A Right Placement Makes a Happy Emulator: a Placement Module for Distributed SDN/NFV Emulation

Giuseppe Lena, Andrea Tomassilli, Frédéric Giroire, Damien Saucez, Thierry Turletti, Chidung Lac

A Right Placement Makes a Happy Emulator: a Placement Module for Distributed SDN/NFV Emulation

Giuseppe Di Lena
Université Côte d'Azur, Orange Labs
Lannion, France
giuseppe.dilena@orange.com

Andrea Tomassilli
Université Côte d'Azur, Inria
Sophia Antipolis, France
andrea.tomassilli@inria.fr

Frédéric Giroire
Université Côte d'Azur, CNRS
Sophia Antipolis, France
frederic.giroire@cnrs.fr

Damien Saucez
Université Côte d'Azur, Inria
Sophia Antipolis, France
damien.saucez@inria.fr

Thierry Turletti
Université Côte d'Azur, Inria
Sophia Antipolis, France
thierry.turletti@inria.fr

Chidung Lac
Orange Labs
Lannion, France
chidung.lac@orange.com

***Abstract*—To handle the ever growing demand of resource intensive experiments distributed, network emulation tools such as Mininet and Maxinet have been proposed. They automatically allocate experimental resources. In this work, we show that resources are poorly allocated, leading to resource overloading and hence to dubious experimental results.**

This is why we propose and implement a new placement module for distributed emulation. Our algorithms take into account both link and node resources and minimize the number of physical hosts needed to carry out the emulation. Through extensive numerical evaluations, simulations, and actual experiments, we show that our placement methods outperform existing ones and allowing to re-establish trust in experimental results.

I. Introduction

The complexity of networks has greatly increased in the last years. Networks currently rely massively on software frameworks and virtualization, and their performances become implementation dependent. This makes them hard to model, or even to simulate, to obtain relevant predictions of the behavior of their protocols. As a complementary approach, we have seen the rise of emulation tools such as Mininet that allow one to experiment network prototypes on a single computer, but using the same real software that would be used in production.

Using a single machine for a rapid emulation prevents resource intensive experiments, e.g., needing heavy memory, processing, input/output, or specific hardware, to emulate, for instance, networks with virtual network functions or artificial intelligence algorithms. To tackle this issue, distributed emulation tools were proposed: Maxinet [1], Mininet Cluster Edition [2], and Distrinet [3].

Carrying distributed emulation rises several challenges. First, facing an experiment, is there a need to distribute it? In other words, how to know if the experiment exceeds the capacity of a single node? Then, if yes, onto how many nodes and on which nodes should it be distributed? Actually, a networking experiment can be seen as a virtual network or a graph with node and link demands in terms of CPU, memory, network capacity, etc. A fundamental problem that arises in this context is how to map virtual nodes and links to a physical network topology while minimizing a certain objective function without exceeding the available resources.

Existing tools have placement modules answering partially these questions. `Mininet Cluster Edition` implements three simple algorithms (Round Robin, Random, and Switch Bin [2]), while `Maxinet` uses an algorithm from `METIS` [4], a library for partitioning graphs. However, these placement methods have several important limitations. Firstly, they do not take into account the nodes' resources and the links' capacities. This means that they do not verify if nodes or links capacities are exceeded. Consequently, experiments may overload links and nodes, leading to unreliable results, as we show in this paper. Secondly, they do not minimize the number of needed machines, and use all machines at their disposal. This is especially important for public clusters, where physical resources are shared. To solve these limitations, we studied placement algorithms that take the experimental infrastructure into account to map an experiment onto a set of physical machines. Our objective consists in minimizing the number of reserved machines to run an experiment, motivated by the fact that scientific clusters such as *Grid5000* [5] require to reserve a group of machines before running an experiment and an excess in these terms may lead to usage policy violations or to a large waiting time to obtain the needed resources.

Contributions. We proposed new tailored placement algorithms and compared them with the ones used in existing tools. We built a *placement module* for distributed emulators to efficiently solve this problem in practice. This module first decides if the experiment has to be distributed. Then, given a pool of available machines, it computes the deployment using the minimum number of machines to run the experiments in such a way that physical resources are not exceeded. To summarize, our contributions are:

- We studied placement algorithms to distribute an experiment onto the machines of a testbed. We proposed several efficient algorithms to deal with the problem.

- We built a placement module for distributed emulators with all the algorithms implemented and used it with `Distrinet` (https://distrinet-emu.github.io).
- We compared our algorithms with the ones implemented in existing tools using extensive simulations.
- We then carried experimentation in a private cluster with the goal of evaluating the impact of such resource overload on the emulation. We show that overloading a link, may lead to important drops of measured bandwidth.

The rest of this paper is organized as follows. In Section II, we review related works on placement methods to carry out distributed emulation. We then formally state the problem and propose algorithms to deal with it in Section III. In Section IV we evaluate the algorithms against the existing placement modules with extensive simulations and experimentation. Last, we conclude and present future work on placement algorithms in Section V.

II. Related Work

Placement for Distributed Emulations. Existing tools for distributed large-scale emulations adopt different strategies to map the virtual topology to the physical one. `Mininet Cluster Edition` [2] provides 3 different placement algorithms: *RoundRobin*, *Random*, and *SwitchBin*.
SwitchBin first distributes the virtual switches (vSwitches) around the infrastructure, such that each physical host has the same amount of vSwitches assigned. It then places the virtual hosts (vHosts) on the host to which its connected vSwitches are assigned.

`Maxinet` uses the Multilevel Recursive Bisectioning algorithm [6] to partition the virtual switches and the virtual hosts into the physical machines. In `Maxinet`, there is no notion about the physical infrastructure (hosts resources or network topology). This means that the partition will not change if we deploy the virtual network in different physical topologies (i.e., spanning tree, clique, etc.). The virtual network, given as an input to the partition algorithm, does not have notion of virtual CPU (vCPU) or virtual RAM (vRAM); i.e., a virtual node (vNode) requiring 1 vCPU is treated like a vNode requiring 10 vCPUs. The partitioning algorithm is not directly implemented in `Maxinet`. `Maxinet` uses `METIS` [4], a set of tools for partitioning graphs. The goal of the algorithm is to find a partitioning of the nodes such that the sum of the nodes weights (e.g., workload) in each partition is balanced and the sum of all the edges in the cuts are minimized.

In all these algorithms, the physical infrastructure is not taken into account. This means that a *physical link or a physical machine can be overloaded and become a bottleneck* for the emulation *without the user being notified*.

Virtual Network Embedding (VNE) Problem. The solution we propose is based on the investigation of a VNE problem. Such problems have been widely studied in the literature. We refer to [7] for a comprehensive survey of the existing works. Many different settings have been considered. Minimization of the resource allocation cost [8], of the energy consumption [9] or revenue maximization [10] are just few examples. In our settings, we aim at finding a mapping which uses the smallest number of substrate nodes. Thus, our objective can be seen as a variant of the energy-aware VNE problem in which we aim to minimize the number of activated substrate nodes.

III. Problem and Algorithms

When emulating large data center networks with hundreds or thousand of nodes, it is necessary to distribute the emulation over multiple physical machines in order not to overload the physical resources.

We study here the problem of mapping virtual nodes and links to a physical network topology, while minimizing the number of used machines and without exceeding the available physical resources (CPU cores, memory, and links' rates). We first define formally the optimization problem which is considered. We then propose algorithms to deal with it.

We consider several specific settings often found in real cluster environments: homogeneous topologies or physical network arranged as a tree.

A. Problem Statement

The VNE problem can be formally stated as follows.
Substrate Network. We are given a substrate network modeled as an undirected multigraph $G^S = (N^S, L^S)$ where N^S and L^S refer to the set of nodes and links, respectively. G^S is a multigraph as there may be multiple links between a pair of nodes. Each node $n^S \in N^S$ is associated with the CPU capacity (expressed in terms of CPU cores) and memory capacity denoted by $c(n^S)$ and $m(c^S)$, respectively. Also, each link $e^S(i,j) \in L^S$ between two substrate nodes i and j is associated with the bandwidth capacity value $b(e^S)$ denoting the total amount of rate that can be supported.
Virtual Network Request. We use an undirected graph $G^V = (N^V, L^V)$ to denote a virtual network, where N^V is the set of virtual nodes and L^V the set of virtual links. Requirements on virtual nodes and virtual links are expressed in terms of the attributes of the nodes (i.e., CPU cores $c(n^V)$ and memory $m(c^V)$) and links (i.e., the rate to be supported $b(e^V)$) of the substrate network. If there are no sufficient substrate resources available, the virtual network request should be rejected or postponed.The problem consists in mapping the virtual network requests to the substrate network, while respecting the resource constraints of the substrate network. The problem can be decomposed into two major components: (1) Node assignment in which each virtual node is assigned to a substrate node, and (2) Link assignment in which each virtual link is mapped to a substrate path.
The combination of nodes and links constraints makes the problem extremely hard for finding a feasible solution. Rost et al. [11] show that the problem of finding a feasible embedding is NP-Complete, even for a single request.

B. Algorithms

We propose three algorithms to tackle this problem. Our algorithms are able to provide near-optimal solutions in a reasonable computation time.

The first two algorithms, K-BALANCED and DIVIDESWAP, have two phases. Firstly, virtual nodes are mapped into the physical topology and, secondly, physical paths are found to map virtual links. The third proposed algorithm, GREEDYPARTITION, mixes both nodes and links mapping.

1) Homogeneous Case:

If the substrate nodes within the cluster are homogeneous in terms of physical resources (or if there is a subset of homogeneous nodes from the entire cluster), an assignment strategy may consist in carrying out a partition of the tasks to be done by the physical machines while minimizing the network tasks that would be necessary to be done. We refer to this algorithm as K-BALANCED. We use as a subroutine an algorithm for the k-balanced partitioning problem. Given an edge-capacitated graph and an integer $k \geq 2$, the goal is to partition the graph vertices into k parts of equal size, so as to minimize the total capacity of the cut edges (i.e., edges from different partitions). The problem is NP-Hard even for $k = 2$ [12]. K-BALANCED solves a k-partitioning problem for $k = 1, \ldots, \min(|N^S|, |N^V|)$, and tests the feasibility of the computed mapping $m : N^V \rightarrow N^S$ of virtual nodes on the substrate network. The smallest k for which a feasible k-partitioning exists will be the output of the algorithm. The corresponding pseudo-code is given in Algorithm 1.

Algorithm 1 K-BALANCED

Input: Virtual network G^V, Substrate network G^S.
Output: a mapping of virtual nodes to substrate nodes $m : N^V \rightarrow N^S$.
for $k = 1, 2, \ldots, \min(|N^S|, |N^V|)$ **do**
    sol $\leftarrow$ Compute an approximate solution of the k-balanced partitioning problem for G^V.
    if sol is feasible (see Sec. III-B3) **then return** sol
return $\emptyset$

The best known approximation factor for the k-balanced partitioning problem is due to Krauthgamer et al. [13] and achieves an approximation factor of $O(\sqrt{\log n \log k})$, with n being the number of nodes in the virtual network. Nevertheless, as their algorithm is based on semi-definite programming and would lead to long execution time, to deal with the problem, we use the $O(\log n)$ approximation algorithm described in [14]. The main idea consists in solving recursively a Minimum Bisection Problem. To this end, we use the Kernighan and Lin heuristic [15].

K-BALANCED has theoretical guarantees of efficiency for its node mapping phase, but only when the number of parts takes some specific values (powers of 2). Indeed, the procedure is based on merging two small partitions until the number of partitions is greater than the desired one.

We thus propose a new algorithm, DIVIDESWAP, efficient for any values of k. The global idea of DIVIDESWAP is to first build an arbitrary balanced partition dividing randomly the nodes in balanced sets, and then swapping pairs of nodes to reduce the cut weight (or required rate for the communications). Its pseudo-code is given in Algorithm 2.

2) General Case:

When the substrate nodes are associated with different combinations of CPU, memory, and networking resources, K-BALANCED and DIVIDESWAP may have difficulties in finding a good assignment of virtual nodes to substrate nodes which respects the different capacities. To prevent this, we define a general procedure referred to as GREEDYPARTITION. Again, we first build a bisection tree. We compute it by recursively applying the Kernighan-Lin bisection algorithm (see the discussion above on bisection algorithm). Then we test the use of an increasing number of physical machines, from 1 to N. We take the j most powerful ones, with powerful defined as a combination of CPU and memory. Next, we perform a BFS visit on the bisection tree. For each considered node, we find a physical node such that the resources are enough and the communication can be performed considering the already placed virtual nodes. If for a node the conditions hold, then the node is removed from the tree with all its subtrees. If not, we consider the next node of the bisection tree. If at any point the tree is empty, we return the solution.

Algorithm 2 DIVIDESWAP

Input: Virtual network G^V, Substrate network G^S.
Output: a mapping of virtual nodes to substrate nodes $m : N^V \rightarrow N^S$.
for $k = 1, 2, \ldots, \min(|N^S|, |N^V|)$ **do**
    Divide the nodes from N^S in k balanced subsets V_1, V_2, ..., V_k.
    Take a random sample of k physical nodes P_1, P_2, ..., P_k
    Assign nodes in V_j to P_j for $j = 1, ..., k$.
    for $i = 1, 2, \ldots, N\_SWAPS$ **do**
        Choose at random two nodes $u, v \in N^V$ assigned to two distinct physical nodes.
        If by swapping u and v the cut weight decreases, swap them in sol and update the cut weight
    if sol is feasible (see Sec. III-B3) **then return** sol.
return $\emptyset$

3) Link Mapping:

DIVIDESWAP and K-BALANCED are based on assigning virtual nodes to physical nodes, then checking the feasibility of the problem by trying to assign all the virtual links between virtual nodes assigned to two different substrate nodes to a substrate path. This problem is solved differently according to the structure of the physical substrate network.

Tree Topologies. Even if the substrate network is assumed to be a tree, there are still decisions to be made in terms of how the network interfaces of a substrate compute node should be used by the virtual nodes. Indeed, if we allow the traffic associated to a single virtual link to be sent using more than one network interface (e.g., 50% on `eth0` and 50% on `eth1`), then multiple links between a compute node and a switch can be considered as a single link with an associated rate which corresponds to the sum of rates on all the node network

interfaces. As a consequence, once the mapping between virtual and substrate nodes has been selected, checking if a substrate compute node has enough resources on its interfaces to send or receive a given set of rates can be done *exactly* in polynomial time with a visit (either BFS or DFS) in time $O(|N^S| + |L^S|)$, as there exists only one path between each source and destination pair. Conversely, if the virtual link rate to be supported can be mapped on only a single network interface, then the situation can be reduced to the *Bin Packing Problem* and is thus NP-hard. To deal with it, we use the First-fit decreasing heuristic which has been shown to be $\frac{11}{9}$-approximated for this problem [16] (i.e., it guarantees an allocation using at most $\frac{11}{9}$OPT + 1 bins, with OPT being the optimal number of bins).

Algorithm 3 GREEDYPARTITION

Input: Virtual network G^V, Substrate network G^S.
Output: a mapping of virtual nodes to substrate nodes $m : N^V \to N^S$.
T $\leftarrow$ Compute a bisection tree of G^V.
for $j = 1, 2, \ldots, |N^S|$ **do**
    Select the j most powerful machines, $P_1, \ldots, P_j$
    Perform a BFS on the bisection tree T.
    for each Node v of T **do**
        if $\exists\ P \in P_1, \ldots, P_j$ with enough resource to host the virtual nodes in v **then**
            v is assigned to P
            Remove v and the subtree rooted at v from T
    if T is empty **then return** sol
return $\emptyset$

General Topologies. Also in this case, we need to distinguish two cases according to the desired strategy for mapping a virtual link of the virtual network to a physical path in the substrate network. If path splitting is supported by the substrate network, then the problem can be solved in polynomial time by using a multi-commodity flow algorithm [10]. On the other hand, if path splitting is not supported, then the situation can be reduced to the *Unsplittable Flow Problem*, which is NP-hard [17]. In such case, we consider the virtual links in non-increasing order. Given the remaining capacities, we find the shortest path in the residual network in which we remove all links with an available rate smaller than the rate to be mapped. If we succeed to find a physical path for all the virtual links to be mapped (i.e., between nodes assigned to distinct physical machines), then the problem is considered feasible.

IV. EVALUATION

We compare our placement algorithms with the ones used by `Mininet Cluster Edition` and `Maxinet`. In order to make the comparison as meaningful as possible and to understand the advantages and disadvantages of each algorithm, we considered different scenarios with *homogeneous* or *heterogeneous* virtual and physical topologies. Scenarios with homogeneous virtual and physical infrastructures are the most favorable for simple placement algorithms. Scenarios with homogeneous physical infrastructures should be the most favorable ones for the placement modules of the existing tools as they do not take into account the physical infrastructure. We show that our algorithms outperform them even in this scenario. The heterogeneous scenario represents a more complex case to show the importance of taking into consideration the capacities of the physical infrastructure.

Physical Topologies. The first physical infrastructure is a simple star topology corresponding to the one of the *Gros* cluster in Grid'5000 [5]. We used 20 physical machines, each equipped with an Intel Gold 5220 (Cascade Lake-SP, 2.20 GHz, 1 CPU/node, 18 cores/CPU) and 96 GB of RAM. The machines are connected by a single switch with 25 Gbps links. The second is represented in Fig. 1: this infrastructure is made of a subset of 25 hosts of the *Rennes* cluster in Grid'5000. There are four types of servers with different numbers of cores (from 8 to 32) and RAM sizes (from 24 GB to 128 GB) for a total of 25 machines. The servers are interconnected using a small network with 4 switches.

Virtual Topologies. We used two different families of virtual topologies: Fat Trees and Random. We chose the first one as it is a traditional family of data center topologies: this corresponds to the homogeneous scenario. Indeed, Fat Trees present symmetries and all servers are usually similar.

Fat Trees. We tested Fat Trees with different parameters:

- *K*: 2, 4, 6, 8 or 10;
- *Number of CPU cores*: the number of virtual cores to assign to each vSwitch or vHost (1, 2, 4, 6, 8 or 10);
- *Memory*: the amount of RAM required by each vSwitch or vHost (100, 1000, 2000, 4000, 8000 or 16000 MB);
- *Links' rates*: the rate associated to each virtual link (1, 10, 50, 100, 200, 500 or 1000 Mbps).

Random Topologies. We used a generator of random topologies which takes as inputs the number of vSwitches and the link density between them. Half of the vSwitches are chosen to be the core network (meaning that no host is attached to them). The other half are the edge switches (vHosts are connected to them). The generator then chooses a random graph to connect the vSwitches, making sure that all of them are connected. The random graph is obtained by generating Erdös-Renyi graphs using the classical `networkx` library till we obtain a connected one that can be used as vNetwork. After setting up the switch topology, a vHost is connected to a single edge switch selected uniformly at random.

A. Placement results

In this section, we extensively study the performances of the different placement algorithms. To this end, we considered more than 70,000 test instances.

Finding a Feasible Solution. When comparing the results of the placement algorithms, we only considered the virtual instances for which at least one of them was able to find a feasible solution. In total, we report the experiments made for more than 5,000 virtual networks.

Table I shows the percentage of instances solved by each algorithm (over the set of feasible instances). We provide the

Cluster	vTopo	GREEDYP	K-BALANCED	DIVIDESWAP	METIS	RANDOM	ROUNDROBIN	SWITCHBIN	INTERSECTION
Gros	vFT	100.0%	91.72%	97.76%	85.02%	72.01%	98.81%	83.18%	68.59%
	vRD	100.0 %	83.31 %	96.64%	58.31%	52.64%	93.15%	37.86%	32.09%
Rennes	vFT	100.0%	83.90%	92.09%	65.81%	43.22%	68.36%	44.49%	34.18%
	vRD	99.98%	73.51%	74.41%	32.75%	20.37%	34.67%	28.40%	11.47%

TABLE I
PERCENTAGE OF SOLUTIONS FOUND USING DIFFERENT ALGORITHMS, VIRTUAL TOPOLOGIES, AND DIFFERENT CLUSTERS.

percentage for each family of virtual topologies: *vFT* (virtual Fat Tree) and *vRD* (virtual Random) topologies. For each family of virtual topology, the tests have been performed on both physical topologies. In particular, the number of feasible solutions analyzed are 761 for *Gros* vFT (Homogeneous-Homogeneous), 4500 for *Gros* vRD (Homogeneous-Heterogeneous), 708 for *Rennes* vFT (Heterogeneous-Homogeneous), and 4436 *Rennes* vRD (Heterogeneous-Heterogeneous). We indicate in the last column of the table (INTERSECTION) the percentage of virtual instances for which all algorithms return a feasible solution. First, we observe that a large number of instances cannot be solved by all the algorithms. Second, the results confirm that heterogeneous (whether virtual or physical) topologies are a lot harder to solve (in particular for the algorithms of the existing tools).

B. Bandwidth Intensive Experiments

From a high level, two of the algorithms proposed in this paper reach the higher success ratio in terms of number of solved instances. In particular, GREEDYPARTITION succeeds to find a feasible solution for almost all the feasible virtual networks when mapped to the *Gros* cluster, and vFT when mapped to the *Rennes* cluster, while it finds a feasible solution for 99.98% of the instances in the vRD case. The second best algorithm is DIVIDESWAP which solved more than 90% of the instances in the *Gros* cluster and in the *Rennes* Cluster for the vFT topology. Note that K-BALANCED has a lower percentage (76.7%). This is expected as this algorithm is efficient when the solution is mapped on specific numbers of physical hosts (powers of 2), a case for which it has some theoretical guarantees.

Then, the algorithm used by `Maxinet` (`METIS`) finds 85.02% of the solutions for *Gros* vFT. As expected, the algorithm drastically changes its performances when the physical environment is non heterogeneous (i.e., *Rennes* vFT), or when the network to emulate has different vNodes requirements or vLinks requirements (i.e., virtualizing a random network in *Gros*). The worst scenario that we have with `METIS` is in the case of homogeneous infrastructure virtualizing a random topology (i.e., *Rennes* vRD) where only 32.75% of the returned solutions are feasible. We tested the 3 other algorithms that are directly implemented in `Mininet Cluster Edition`. `Random` solves 72 % of the instances in *Gros* vFT, while the performances drop drastically in the homogeneous case and when virtualizing a random topology. The same behavior can be observed for `Round Robin` and `Switch Bin`.

Number of Physical Hosts Needed. An important additional advantage of the algorithms that we propose in this paper is that they minimize the number of physical hosts needed to emulate the experiments. This helps reducing the use of testbed resources and even making feasible some large experiments that would not be able to run without well optimizing hardware usage, as we show below.

We report in Fig. 4 the distributions of the number of hosts used by the algorithms over all the virtual topologies for which all algorithms found a feasible solution (INTERSECTION subset). Note that this subset of experiments does not contain many large topologies, as the less efficient placement algorithms were not able to find solutions for them. As expected, the proposed algorithms use much fewer physical hosts. For the *Gros* cluster (Fig. 4), the general tendency is that GREEDYPARTITION uses between 1 and 13 hosts (median is 3) in case it is emulating a vFT, and between 2 and 11 hosts (median is 5) in case it emulates a vRD. `METIS` uses a minimum of 4 instances with a maximum of 20 instances (medians are 4 and 12). `Round Robin` (medians are 13 and 14), `Random` (medians are 7 and 20), and `Switch Bin` (medians are 5 and 15) use in general more hosts than METIS.

The experiment shows the performances of our placement module in a network intensive scenario. The networks we emulate are virtual Fat Trees with K=4 and K=6. They are composed of vHosts and vSwitches requiring 1 vCore and 1 GB of RAM, while all the vLinks are set to 500 Mbps. Half of the vHosts are clients and the other ones are servers. The experiment consists in running TCP `iperf` between each pair of client/server. The total aggregated demand to be served is 4 Gbps and 13.5 Gbps, while the total network traffic generated is 24 Gbps and 81 Gbps, for K=4 and K=6, respectively. The traffic is forwarded in a way guaranteeing that each client is theoretically able to send at full speed. This is possible as a Fat Tree is a permutation network. Each experiment was performed 10 times, enough to obtain reliable results as the variability is small.

Heterogeneous Case. We now consider the heterogeneous physical (yet simple) physical topology of the *Rennes* cluster (see Fig 1). Results using this experimental platform are illustrated in Figs. 2 and 3. The experiments show how the emulation can return unexpected results when the placement of the virtual nodes does not take into account links' rates. Note that, as this topology is not homogeneous like the *Gros* cluster, finding a good placement is significantly harder, as discussed previously.

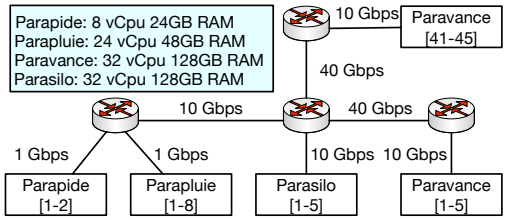

Fig. 1. *Rennes* topology cluster

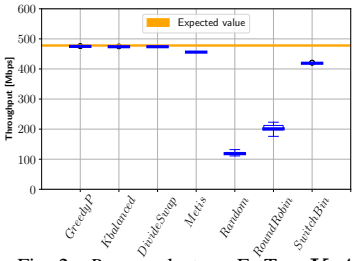

Fig. 2. *Rennes* cluster, vFatTree **K**=4.

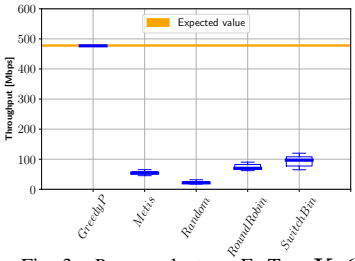

Fig. 3. *Rennes* cluster, vFatTree **K**=6.

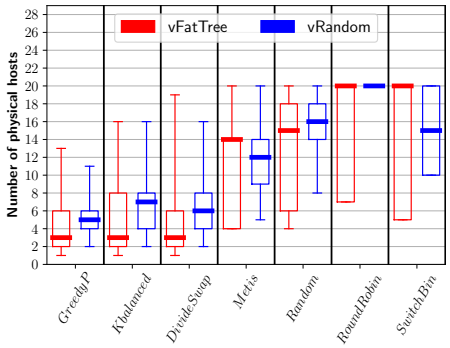

Fig. 4. Number of physical hosts used by the placement algorithms. Homogeneous *Gros* cluster.

The test in *Rennes* consists in emulating a vFT topology K=4 in this cluster using all the algorithms. In this case, the emulation creates a vFT with 16 vHosts and 20 vSwitches. Fig. 2 reports the bandwidth performance of a vFT (with K=4) obtained for the different placement algorithms when the emulation is performed in the *Rennes* cluster. As we can observe, the bandwidth results using GREEDYPARTITION, K-BALANCED, DIVIDESWAP, and METIS are the ones expected from the emulation, while the results obtained by other algorithms, RANDOM and ROUNDROBIN, are far from the expected ones. Indeed, some of the links are overloaded in the placement returned by the latter algorithms. This means that the paths of two demands are using the same links. For these links, the throughput drops to 120 Mbps and 200 Mbps, respectively. When running a larger emulation for a vFT with K=6, we can observe in Fig. 3 that K-BALANCED, DIVIDESWAP do not find a feasible solution and the results returned by METIS and SWITCHBIN are not trustworthy anymore. The measured throughput on some overloaded links has fallen to very low values between 20 and 100 Mbps, to be compared with the expected 478 Mbps. On the contrary, the emulation distributed with GREEDYPARTITION returns exactly the expected results, showing the efficiency and reliability of the proposed algorithm. The experiments can be reproduced following the tutorial at:
https://github.com/jdoe79250/algo_experiments

V. CONCLUSION

In this paper, we propose a placement module for tools enabling distributed network emulation. Indeed, large scale or resource intensive emulations have to be distributed over several physical hosts to avoid overloading hosts carrying out the emulation. A network experiment can be seen as a virtual network with resources needed on nodes (e.g., CPU or memory) and links (e.g., bandwidth). This network has to be mapped to the physical clusters on which the emulation is done. As we show in this paper, poor mapping may lead to an overload of physical resources leading to unreliable experiments. This is the reason why we propose and evaluate in this paper placement algorithms that provide trustworthy mapping ensuring that resources are never overloaded. As a bonus, our algorithms also minimize the number of physical machines required. We show that they outperform existing solutions in all aspects.

REFERENCES

[1] P. Wette, M. Dräxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl, "Maxinet: Distributed emulation of software-defined networks," in *2014 IFIP Networking Conference*, June 2014, pp. 1–9.

[2] "Cluster edition prototype," https://github.com/mininet/mininet/wiki/Cluster-Edition-Prototype.

[3] G. Di Lena, A. Tomassilli, D. Saucez, F. Giroire, T. Turletti, and C. Lac, "Mininet on steroids: exploiting the cloud for mininet performance," in *IEEE CloudNet*, 2019.

[4] "Metis doc." http://glaros.dtc.umn.edu/gkhome/metis/metis/overview.

[5] "Nancy topology," https://www.grid5000.fr/w/Nancy:Network.

[6] G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning," in *SC'98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*. IEEE, 1998, pp. 28–28.

[7] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[8] N. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *IEEE INFOCOM 2009*. IEEE, 2009, pp. 783–791.

[9] J. F. Botero, X. Hesselbach, M. Duelli, D. Schlosser, A. Fischer, and H. De Meer, "Energy efficient virtual network embedding," *IEEE Communications Letters*, vol. 16, no. 5, pp. 756–759, 2012.

[10] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, pp. 17–29, 2008.

[11] M. Rost and S. Schmid, "Charting the Complexity Landscape of Virtual Network Embeddings," in *Proc. IFIP Networking*, 2018.

[12] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified np-complete problems," in *Proceedings of the sixth annual ACM symposium on Theory of computing*. ACM, 1974, pp. 47–63.

[13] R. Krauthgamer, J. Naor, and R. Schwartz, "Partitioning graphs into balanced components," in *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 2009, pp. 942–949.

[14] H. D. Simon and S.-H. Teng, "How good is recursive bisection?" *SIAM Journal on Scientific Computing*, vol. 18, no. 5, pp. 1436–1445, 1997.

[15] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell system technical journal*, pp. 291–307, 1970.

[16] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM Journal on computing*, pp. 299–325, 1974.

[17] S. G. Kolliopoulos and C. Stein, "Improved approximation algorithms for unsplittable flow problems," in *Proceedings 38th Annual Symposium on Foundations of Computer Science*. IEEE, 1997, pp. 426–436.