Mininet on steroids: exploiting the cloud for Mininet performance

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

Mininet on steroids: exploiting the cloud for Mininet performance

Giuseppe di Lena*†, Andrea Tomassilli*, Damien Saucez*, Frédéric Giroire*, Thierry Turletti*, and Chidung Lac†
*Université Côte d'Azur, Inria, CNRS, France
†Orange Labs, France

***Abstract*—Networks have become complex systems that combine various concepts, techniques, and technologies. As a consequence, modelling or simulating them is now extremely complicated and researchers massively resort to prototyping techniques. Among other tools, Mininet is the most popular when it comes to evaluate SDN propositions. It allows to emulate SDN networks on a single computer. However, under certain circumstances experiments (e.g., resource intensive ones) may overload the host running Mininet. To tackle this issue, we propose *Distrinet*, a way to distribute Mininet over multiple hosts. Distrinet uses the same API than Mininet, meaning that it is compatible with Mininet programs. Distrinet is generic and can deploy experiments in Linux clusters or in the Amazon EC2 cloud. Thanks to optimization techniques, Distrinet minimizes the number of hosts required to perform an experiment given the capabilities of the hosting infrastructure, meaning that the experiment is run in a single host (as Mininet) if possible. Otherwise, it is automatically deployed on a platform using a minimum amount of resources in a Linux cluster or with a minimum cost in Amazon EC2.**

I. Introduction

We have witnessed the decline of holistic models and simulations and the rise of prototyping and experiments in real systems. In particular, we have seen the advent of the Mininet network emulator [4] that allows one to experiment network prototypes on a single computer, using the same real software that would be deployed in production.

Problem. Mininet is a fantastic emulator for network experiments with a simple yet powerful Python API [4] that emulates network topologies on the experimenter computer with virtual environment, network namespaces, and virtual switches. This approach is particularly useful as it to allow fast prototyping of network experiments. Unfortunately, *Mininet was not designed for resource intensive experiments* [4] and one computer may not be sufficient to properly execute the tasks.

Challenges. Mininet is used by a large community of researchers and students. The success of this tool resides in its simplicity: it can work directly on a laptop, its installation is trivial, its Python API is simple yet powerful, and it can be used for complex scenarios with the same ease as for basic tests. The question is to understand how to rethink Mininet to support resource intensive experiments by transparently distributing them to a pool of computing resources.

This work has been supported by the French government through the UCA JEDI (ANR-15-IDEX-01) and EUR DS4H (ANR-17-EURE-004) Investments in the Future projects, and by Inria associated team EfDyNet.

The approach. Mininet and its API have proven to be the right tools for researchers. It is essential to keep its usage unchanged for users but we must make it able to scale with good experimental fidelity. We propose *Distrinet*, an extension of Mininet that distributes experiments on multiple hosts. Experiments are defined with the Mininet idiom (e.g., API or CLI) and are transparently deployed on several machines when needed.

Related Work. To speedup experiments, Maxinet [10] proposes to distribute experiments by running multiple instances of Mininet in a cluster. Maxinet philosophy is the same as ours, except that instead of extending Mininet, it uses it and then provides a new API. As a result, Mininet scripts are not directly usable with Maxinet while our approach allows to directly run Mininet scripts in a distributed environment. Yan and Jin took a radically different approach and added virtual time to Mininet [11]. It can significantly improve fidelity of Mininet, even under high load. Regrettably, while in most situations virtual time is a good solution, it prevents users to pair their Mininet experiments with real hardware (e.g., TCP offloading, GPUs) or with real traffic (e.g., external controller), which can be an issue when experimenting new networking technologies (e.g., optical, 5G…). Finally, Containernet extends Mininet to isolate nodes in docker containers in order to emulate Virtual Network Functions (VNFs) [5]. It is possible to combine Maxinet and Containernet to have a distributed container-based SDN/VNF emulator. The main problem is that Maxinet and Containernet do not consider the properties of the physical infrastructure in which they are running. So, a machine or a link in the physical network may be overloaded during the emulation. Thus, there is no guarantee on the reliability of the results. With Distrinet we combine the concepts of Maxinet and Containernet to provide a distributed Mininet implementation which takes into account the characteristics of the logical and physical topologies.

Contributions. A key factor for the adoption of Distrinet is to share the same API as Mininet such that Mininet programs and scripts can readily be used, but with better confidence on the experiment results. Our main contributions to reach this objective can be summarized as follows.

- **Architecture. (Sec. II)** We designed and developed the Distrinet distributed network emulator that can be used for intensive compute and I/O experiments on a large variety of experimental infrastructures (e.g., a single computer, a Linux cluster, or the Amazon EC2 cloud). The implementation is lightweight and robust as it uses

Ansible and LXD to manage experiments and to isolate their components. We use LXD to orchestrate emulated nodes and switches.

- **Compatibility with Mininet.** Distrinet is a superset of Mininet. It means that Mininet experiments are fully compatible with Distrinet, either with the Mininet API or with the Mininet Command Line Interface (i.e., `mn`). Distrinet extends Mininet code: as a result, it can readily benefit from the future advances of Mininet.
- **Optimization. (Sec. III)** Based on the experimental infrastructure information, we optimally allocate resources to the experiments. As the knowledge about the experimental infrastructure can be partial or total, we developed two distinct optimization strategies: (i) when the infrastructure network topology is unknown (e.g., in public clouds such as Amazon EC2) we rely on a Vector Bin Packing with Multiple-Choice optimization scheme, and (ii) when the experimental infrastructure topology is known (e.g., in private testbeds or clusters) we rely on a Virtual Network Embedding (VNE) problem.

II. Architecture

Mininet defines a powerful Python API to implement experiments as Python programs and provides a command-line interface (i.e., `mn`) to help in debugging or to allow interactive manipulations during an experiment. Despite being built with lightweight virtual environment techniques, Mininet cannot properly handle resource-intensive experiments because it runs on a single computer. As a consequence, when heavy load is requested, it is expected to experience resource contention and fidelity to the reality is unclear. To tackle this issue Distrinet extends Mininet to distribute experiments over multiple hosts. Its general architecture is presented in Fig. 1.

In Mininet, hosts are emulated as user-level processes isolated from each others by making use of the Linux cgroup and network namespaces. With Distrinet, we re-use the principle of containerisation but implement it with LXC and LXD to simplify the management of complex scenarios.

Mininet heavily relies on `popen` to interact with the emulated nodes. In order to distribute Mininet, we implemented a new class, `RemotePopen` – that implements the same interface as the native `Popen` class of Python – and extended the `Node` class of Mininet to use our implementation instead of using the one from Python. The `RemotePopen` is implemented with SSH channels from the `Paramiko` library. To control emulated nodes via SSH, a control network interconnecting all nodes is deployed.

Network links are emulated with virtual Ethernet when the two end points of the link are deployed on the same machine. If the end points are deployed on separate machines, then the connection is made over VxLAN. From that point of view Distrinet significantly differs from Mininet as with our implementation packets cross the full network stack (including hardware). Link data rate control is directly inherited from Mininet.

Distrinet provides an infrastructure provisioning mechanism that automatically installs and configures LXD and SSH on each machine to be used during the experiment by the means of Ansible. If the experimental infrastructure is Amazon EC2, Distrinet instantiates first a Virtual Private Cloud (VPC) configured as depicted in Fig. 2 in which the virtual instances running the experiment will be deployed. A NAT gateway is automatically created to provide Internet access to the emulated nodes. Access to the emulated nodes from the experimenter machine is ensured by a master node acting as an SSH relay. The deployment on Amazon EC2 only requires an active Amazon AWS account.

Distrinet determines the number of machines to use for an experiment and how to deploy the emulated network on these machines by first solving an optimization problem (see Sec. III for more details). To plug the optimization mechanism, we introduced the new abstract class `Mapper` that the optimizer must implement in order to be used for deploying experiments. When no optimization is provided, the placement is decided by the LXC scheduler.

III. Resource allocation

An experiment can be seen as a virtual network, modeled as a graph annotated with node and link demands in terms of, e.g., CPU, memory and network capacity. A fundamental question is how to map virtual nodes and links to a physical network topology while minimizing a certain objective function without exceeding the available resources. Two different cases must be considered. Either experiments are run in a fully controlled Linux cluster or they run in a public cloud (e.g., Amazon EC2). These two use cases lead to the following distinct problems.

The Virtual Network Embedding problem. On a fully controlled cluster we can directly manage both the compute resources (i.e., CPU cores, I/Os, and memory) and the network resources (i.e., link rate, path and interfaces). To properly assign resources and paths, we have to solve a *VNE* problem. Our objective consists in minimizing the number of reserved machines to run an experiment, motivated by the fact that scientific clusters such as *Grid5000* [1] require to reserve a group of machines before running an experiment [9] and an excess in these terms may lead to usage policy violations or to a large waiting time to obtain the needed resources.

The combination of nodes and links constraints makes the problem extremely hard even in finding a feasible solution. Rost et al. [6] show that the problem of finding a feasible embedding is NP-Complete, even for a single request. Exact solutions which propose optimal techniques to solve small instances have been proposed (e.g., [3]) but they are not suitable here as they are too slow. We then prefer a heuristic approach running with short execution time. We propose three heuristics that provide near-optimal solutions in a reasonable computation time: (i) *k-balanced* carries out a partition of logical nodes, while minimizing the network capacity needed between physical nodes. We use as a subroutine an algorithm for the k-balanced partitioning problem given

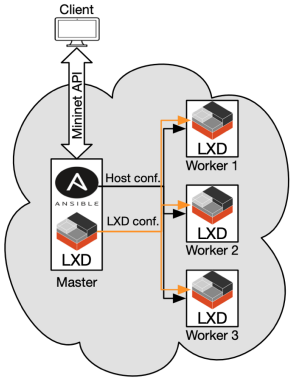

Figure 1: The client executes the script. The master and the workers run the actual experiment.

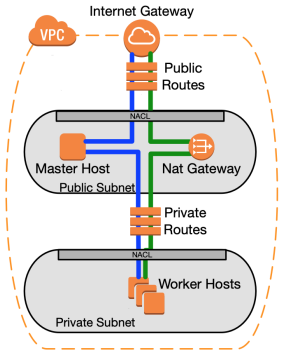

Figure 2: Amazon VPC configuration.

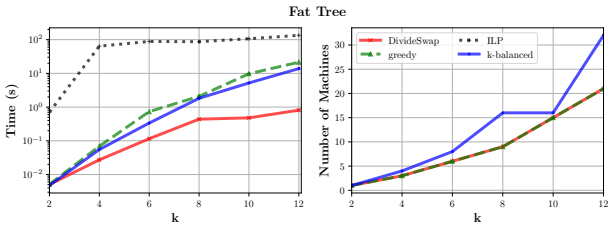

Figure 3: Performance of both the heuristics and the ILP solver. Left: Time needed to find the solution. Right: Number of machines needed to run the experiment as a function of the k-fat tree size.

in [8]; (ii) *Greedy* first builds a bisection tree, where each node is divided into 2 partitions of nodes. Then, the tasks are assigned to the available physical machines by doing a bread first search visit on the tree, assigning to each machine the first fitting subtree; (iii) *DivideSwap* divides the logical nodes into n physical machines and, then, iteratively swap them to reduce the network cut weight.

Vector Bin Packing with Multiple-Choice (VBP-MC). When experiments run in a public cloud, the problem consists in choosing a set of virtual machine instances taken from a set of instance types, which provide different combinations of CPU, memory, disk, and networking. In this case, the natural objective consists in minimizing the cost to run an experiment. This problem is often referred to as the *Vector Bin Packing with Multiple-Choice*. For space reasons, we only briefly present the adopted solutions. We implemented three heuristics approaches in Distrinet: *First Fit Weighted Sum* [7], *Best Fit Dop Product* [7], and *First Fit Ordered Deviation* [2].

Experimental Evaluation. Due to lack of space, we only present here some results for the VNE problem. The experiment is to map data center servers interconnected by a k-fat tree onto the Nancy node of [1] of the academic cluster GRID5000. Each logical switch and server require 2 cores and 8 Gb of memory, and each server is sending 0.2 Mbps of traffic. The physical nodes are machines with 32 cores and 132 Gb of RAM. The physical links are Ethernet links of capacity 10 Gbps. The goal was to minimize the number of machines of the cluster used for the experiment for different data center sizes. We compare the solutions given by three heuristics with the ones obtained using linear programming (with a running time limit set to 2 minutes) in Fig. 3. First, we verify that our solutions map a 2-fat tree onto a single machine as the requirements in terms of cores and memory are low enough. However, this is no longer the case for 4-fat trees and larger topologies. Second, we see that the best proposed algorithms, *DivideSwap* and *Greedy*, reach optimum or close to optimum solutions within a few seconds, showing they can be used for fast experimental deployment.

IV. Conclusion

To adapt Mininet to resource-intensive experiments, we proposed Distrinet that extends Mininet to make it able to distribute experiments on an arbitrary number of hosts. Distrinet automatically determines the number of hosts required for an experiment. Our implementation is flexible and experiments can be run on a single Linux host, a Linux cluster, or even in the Amazon EC2 cloud. Distrinet automatically provisions hosts and launches Amazon instances such that experimenters do not have to know the details of the infrastructure they will use for their experiment. We are developing methods to automatically compensate the variability of cloud resources.

Distrinet is compatible with Mininet and its source code is available on https://distrinet-emu.github.io.

References

[1] Grid5000. https://www.grid5000.fr/w/Grid5000:Home.

[2] B. T. Han, G. Diehr, and J. S. Cook. Multiple-type, two-dimensional bin packing problems: Applications and algorithms. *Annals of Operations Research*, 50(1):239–261, 1994.

[3] I. Houidi, W. Louati, W. B. Ameur, and D. Zeghlache. Virtual network provisioning across multiple substrate networks. *Computer Networks*, 55(4):1011–1023, 2011.

[4] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *ACM SIGCOMM Workshop HotNets*, New York, NY, USA, 2010. ACM.

[5] M. Peuster, H. Karl, and S. van Rossem. Medicine: Rapid prototyping of production-ready network services in multi-pop environments. In *IEEE NFV-SDN*, pages 148–153, Nov 2016.

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

[7] P. Silva, C. Perez, and F. Desprez. Efficient heuristics for placing large-scale distributed applications on multiple clouds. In *IEEE/ACM CCGrid*, 2016.

[8] H. D. Simon and S.-H. Teng. How good is recursive bisection? *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997.

[9] P. Vicat-Blanc, B. Goglin, R. Guillier, and S. Soudan. *Computing networks: from cluster to cloud computing*. John Wiley & Sons, 2013.

[10] 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*, pages 1–9, June 2014.

[11] J. Yan and D. Jin. Vt-mininet: Virtual-time-enabled mininet for scalable and accurate software-define network emulation. In *ACM SIGCOMM Symposium SOSR*, New York, NY, USA, 2015. ACM.