# Demo Proposal - Distrinet: a Mininet implementation for the Cloud

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

# Demo Proposal – Distrinet: a Mininet implementation for the Cloud

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*—*Abstract*—Networks became so complex and technical that it is now hard if not impossible to model or simulate them. Consequently, more and more researchers rely on prototypes emulated in controlled environments and Mininet is by far the most popular tool. Mininet implements a simple, yet powerful API to define and run network experiments on a single machine. In most cases, running experiments on one machine is adequate but for resource intensive applications one machine may not be sufficient. For that reason we propose *Distrinet*, a way to distribute Mininet over multiple hosts. Distrinet uses the same API than Mininet, granting full compatibility with Mininet programs. Distrinet is generic and can optimally deploy experiments in Linux clusters or in public clouds and automatically minimizes the resource consumed in the experimental infrastructure.**

## I. Introduction

Network emulation is becoming the candidate of choice when it comes to SDN evaluation because modern networks are too complex and implementation-dependent to be modeled or simulated. In this field of research, Mininet [3] is by far the most popular network emulator. It provides a simple yet powerful API to emulate networks on one machine thanks to lightweight virtualization techniques. The success of Mininet comes from its simplicity: it can work directly on a basic laptop or a powerful workstation, its installation is trivial and its Python API is simple yet powerful. Most of the time, running experiments in one machine is fine but for resource intensive applications it may not be adapted anymore as one computer may not offer enough resources [3].

Mininet has been designed to run on one single machine assuming that all resources are shared. Unfortunately, when multiple machines are used to run an experiment, this assumption doesn't hold true anymore, making it hard to implement the Mininet API in a way that keep all the code compatible with legacy Mininet code.

We propose to extend Mininet with *Distrinet*. Distrinet keeps the Mininet idiom (e.g., API and CLI) and keeps untouched the Mininet core implementation but it transparently deploys experiments on multiple machines.

**Related Work.** Maxinet [6] distributes experiments by running multiple instances of Mininet in a cluster. Maxinet and Distrinet share the same philosophy but have an important technical difference: Mininet scripts are not directly usable with Maxinet while our approach allows to directly run Mininet scripts in a distributed environment. Yan and Jin added virtual time to Mininet [7] to improve fidelity of Mininet experiments. Virtual time is a right approach in most cases, but it prevents users to pair their Mininet experiments with real hardware or with real traffic. Finally, Containernet extends Mininet to better isolate nodes, by means of Docker containers [4]. To benefit from this isolation, Maxinet has proposed an extension running with Containernet.

**Our contributions.** Distrinet [2] is a superset of Mininet designed to be used for resource intensive (e.g., compute, I/Os) experiments requiring multiple machines to be executed properly. It is fully compatible with Mininet and can be run on a large variety of experimental infrastructures (e.g., a single computer, a Linux cluster, Amazon EC2).

Experimental infrastructure information is taken into account by Distrinet to optimally allocate resources to the experiments such that the cost of running experiments is minimized.

## II. Architecture

In Mininet a network node is basically a bash process isolated from the rest with cgroups. Mininet interacts with the process by reading and writing its standard outputs and input directly via their file descriptors. Unfortunately, the file descriptors and PIDs spaces are not shared when multiple machines are used. To make the bridge we use SSH.

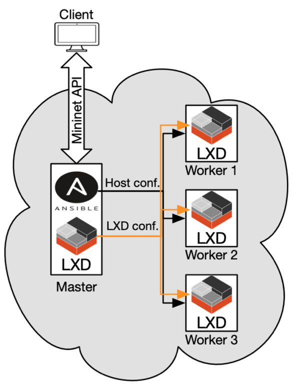

Figure 1. The client executes the script. The master and the workers run the actual experiment.

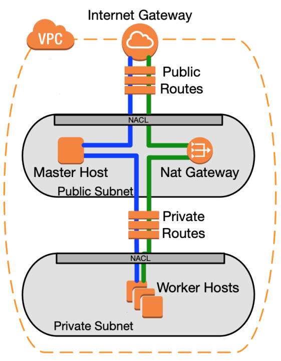

Figure 2. Amazon VPC configuration.

We could have leveraged SSH options or even to use RPCs as in Maxinet. However, our objective is to be compatible with Mininet, not only from a user API point of view but also from a code developer point of view. For that reason, we kept the Mininet idea of relying only on the read and write primitives. That's why we create a node by starting a bash process in the exact same way as in Mininet, except that it is started in a LXC container instead of directly on the host.[1] The standard input and outputs are bound to local file descriptors that are connected to the actual emulated node via SSH. As a result, all the Mininet core code can seamlessly be used to interact with the emulated nodes. Network links are emulated with VxLAN tunnels.

Distrinet provides an infrastructure provisioning mechanism that uses Ansible to automatically install and configure LXC, SSH, and all the required dependencies on the machines as shown in Fig. 1.

Distrinet is designed to be easy to deploy either in physical or cloud infrastructure. In physical infrastructure, the Distrinet client (the machine where Distrinet is installed) must be able to connect via ssh to the master node, and the master node must be able to connect via ssh to the workers. Distrinet will configure Ansible and LXD automatically in the hosts. Another requirement is that the master and the workers have Internet access. To automatically deploy an infrastructure compatible with Distrinet in Amazon EC2, Distrinet creates a new Virtual Private Cloud (VPC) with public and private subnet Fig. 2. The master host is placed in the public subnet in order to be accessible by the Distrinet client, while the workers are placed in the private subnet. In order to give Internet access to the worker hosts, a NAT Gateway is created in the public subnet and the private route table is updated to forward all the public traffic via the virtual NAT.

## III. Resource allocation

Ultimately, a Mininet experiment can be seen as an annotated graph and Distrinet uses it to determine 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.

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 free fully controlled Linux cluster or they run in a costly public cloud (e.g., Amazon EC2). These two use cases lead to the following distinct problems.

The first one is a virtual network embedding problem where the objective is to find a valid mapping between the virtual topology used for the experiment and the resources in the infrastructure. In such a situation, Distrinet minimizes 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 [5] and an excess in these terms may lead to usage policy violations or to a large waiting time to obtain the needed resources.

The second case is a vector bin packing with multiple-choice problem. 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, Distrinet objective consists in minimizing the cost to run an experiment.

[1]The container is used to guarantee that all emulated hosts run in the same Linux environment, regardless of the host.

## IV. Conclusion

We proposed Distrinet that extends Mininet to make it able to distribute experiments on multiple hosts. Distrinet determines the number of hosts required for an experiment to minimize the overall consumed resources. Our implementation is flexible and can be run on any pool of Linux hosts or even in the Amazon EC2 cloud. Distrinet seamlessly provisions hosts and launches cloud instances when needed. Distrinet is compatible with Mininet and its source code is available on http://distrinet-emu.github.io.

## References

[1] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec. Adding virtualization capabilities to the Grid'5000 testbed. In I. I. Ivanov, M. van Sinderen, F. Leymann, and T. Shan, editors, *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer International Publishing, 2013.

[2] 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*, volume 8. IEEE, 2019.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.