Federating EdgeNet with Fed4FIRE+ and Deploying its Nodes Behind NATs

Berat Can Senel, Maxime Mouchet, Justin Cappos, Timur Friedman, Olivier Fourmaux, Rick Mcgeer

Federating EdgeNet with Fed4FIRE+ and Deploying its Nodes Behind NATs

Berat Can Şenel
Sorbonne Université, France
berat.senel@lip6.fr

Maxime Mouchet
Sorbonne Université, France
maxime.mouchet@lip6.fr

Justin Cappos
New York University, United States
jcappos@nyu.edu

Timur Friedman
Sorbonne Université, France
timur.friedman@sorbonne-universite.fr

Olivier Fourmaux
Sorbonne Université, France
olivier.fourmaux@sorbonne-universite.fr

Rick McGeer
engageLively, United States
rick.mcgeer@engagelively.com

***Abstract*—EdgeNet is a globally-distributed edge cloud testbed. It aims to provide an experimentation platform to computer networks and distributed systems researchers. However, distributed testbed providers encounter difficulties with node provisioning, access, and maintenance when establishing an edge cloud consisting of ubiquitous nodes. EdgeNet extends Kubernetes to the edge and addresses these challenges. Employing an industry-standard container orchestration system enables researchers to straightforwardly deploy experiments across many vantage points, maximizing their time for collecting and analyzing measurement data. This paper describes three features that we have developed as liberally-licensed, free, open-source extensions to Kubernetes: federation, nodes in home networks, and easy node deployment. We also discuss remote node maintenance as future work.**

***Index Terms*—Edge Cloud, Testbed, Kubernetes, Home Network, Federation, Internet Researchers**

I. Introduction

A growing number of internet users and their use of services such as video streaming and gaming, along with data produced by IoT devices, are placing heavy demands on cloud networks. In recent years, edge computing, which can be broadly defined as the processing of data where it is being generated [1], [2], has been introduced to address this issue as well as to process data in real-time. With edge computing comes edge clouds, which aim to provide computing services ensuring lower latency and jitter with decreased bandwidth pressure on the core network [3].

This novel computing paradigm captures the attention of researchers in launching experiments from such environments. While providing an edge cloud testbed at scale is possible to satisfy this demand, it poses a challenge since the nodes are located and maintained at the edge and they require physical and remote access, thus incurring additional costs. If these nodes are located behind network address translation (NAT) boxes, it introduces an extra burden in terms of remote access. Taken together, the issues of delivering, maintaining, and accessing nodes at the edge, also considering hindrances that come from NAT boxes, impair the long-term viability of edge cloud testbeds due to economic constraints.

We grouped the challenges of delivering edge nodes into three categories: provision, access, and maintenance. Geographically distributed infrastructure conceived for edge computing causes an increase in the complexity of provisioning nodes. In order to alleviate this complexity, an easy node deployment procedure is necessary. However, this kind of procedure does not address economic and organizational obstacles to a provider establishing ubiquitous edge clouds that run at scale. To overcome such economic and organizational difficulties, a collaboration between edge cloud providers, where they open up their infrastructures to each other's users, possibly through a federation, can be an efficient method.

The edge infrastructure mentioned above also limits physical access to the nodes. This physical access issue is primarily related to maintenance and also urges collaboration across organizations providing nodes. On the other hand, remote access is a technical problem. Nodes behind NATs do not hold a public IP address but a private IP address to communicate; thus, they become unreachable outside the local network remotely by default. That is to say, a system's control plane cannot reach such a node to take necessary actions such as service deployment. Being physically inaccessible and having connectivity issues raises maintenance challenges for edge nodes.

We tackle these three issues: providing, accessing, and maintaining edge nodes. We do so on EdgeNet, an edge cloud testbed that we develop and operate. EdgeNet consists in both a production globally-distributed edge cloud testbed[1] for researchers; and the liberally-licensed, free, open-source software[2] that powers the testbed. This code consists in a set of

EdgeNet got its start thanks to an NSF EAGER grant, and now benefits from a VMware Academic Program grant and a French Ministry of Armed Forces cybersecurity grant. *(Corresponding authors: Berat Can Şenel; Maxime Mouchet; Timur Friedman; Olivier Fourmaux.)*

Berat Can Şenel, Maxime Mouchet, Timur Friedman, and Olivier Fourmaux are with the LIP6 CNRS laboratory, 75005 Paris, France.

Berat Can Şenel, Maxime Mouchet, and Timur Friedman are also with the LINCS laboratory, 91120 Palaiseau, France.

[1]The EdgeNet testbed https://edge-net.org/

[2]The EdgeNet software https://github.com/EdgeNet-project

extensions to Kubernetes,[3] the de facto industry standard tool for orchestrating the deployment of containers to the cloud.

The multitenancy framework already offered by EdgeNet empowers different researcher groups to conduct experiments in parallel, thus opening up its infrastructure to a broader community with the aim of achieving high resource utilization in the cluster. However, other providers' users cannot use it without registering or EdgeNet users cannot benefit from other providers' infrastructure through this framework. Regarding node deployment, our straightforward procedure, called the multi-provider feature, to add nodes to the cluster required a public IP address. This requirement prevented nodes behind NAT boxes from joining the cluster.

This paper makes three contributions:

- *Federation:* We believe that a federation of edge clouds, each cloud being offered by a different provider, is essential to establish heterogeneity and enable numerous vantage points. Such a federation can virtually address two problems discussed above: provision and maintenance. As multiple providers deliver edge nodes, a federation can ensure an infrastructure at scale. It also facilitates the maintenance of the nodes through physical operation, thanks to a shared workload between the providers.
  We developed an aggregate manager (AM), which empowers researchers to use EdgeNet through Fed4FIRE+ [4] as they do for the other federated testbeds. That is to say, it removes the requirement of registering with the testbed to conduct measurements on our globally distributed edge cloud.
- *Home Networks:* Since the advent of edge computing, home networks have drawn heavy attention from internet researchers for the past few years. With home networks, the presence of NAT boxes is an important deterrent to deployment. Thus, it is not a trivial task to manage the life-cycle of an experiment, such as deployment and version updates, that launches measurements from a node behind a NAT box.
  We offer a virtual private network solution that makes it possible for nodes at home networks to take part in a cluster, handling the access issue, as explained in Sec. IV. An agent running on each node configures the VPN network by actions such as assigning an IPv4/IPv6 pair of addresses. Current deployment ensures a VPN tunnel is established between two public nodes or between a public node and a NATted node. In future releases, we plan to support communication between two NATted nodes through the VPN network.
- *Node Deployment:* A fundamental necessity is a simple procedure that safely sets up a node and makes it join a cluster. By providing nodes in such a forthright manner, a contributor is less likely to give up during the process. EdgeNet facilitates node contributions to the cluster via its deployment procedure. We now include the installation of the node agent mentioned above in this procedure to automate VPN establishment.

This paper is organized as follows: Sec. II presents related work regarding similar testbed platforms; Sec. III introduces the architecture of our Fed4FIRE+ aggregate manager; Sec. IV describes how EdgeNet extends to home networks; Sec. V describes the node deployment procedure; Sec. VI discusses future work to maintain and recover nodes remotely; and Sec. VII is the conclusion.

II. Related Work

The networking and distributed systems research communities have provided various edge cloud testbeds typically spanning broad geolocations such as PlanetLab [5], PlanetLab Europe [6], GENI [7], G-Lab [8], V-Node [9], and SAVI [10] in the past decades. All of these testbeds required dedicated hardware and delivered custom software. These two design decisions hindered efficiency and sustainability.

First, dedicated hardware has caused an increase in maintenance and scaling costs because of a need for on-site support and initial purchase investment. Thus, contributors abandoned nodes to their fate over time. Second, typically, these testbeds have been supported by researchers writing custom software. This introduces a heavy workload on coding and preparing tutorials for those who maintain that testbed. Furthermore, it commonly obliges an experimenter to learn a new control framework for each testbed.

EdgeNet's philosophy is different from these testbeds in two respects. It encourages contributors to supply virtual machines as a node instead of dedicated hardware, in order to decrease the cost of providing and maintaining the testbed. And to reduce programming and documentation workload, it adapts industry-standard open-source software for the needs of the testbed. Thus, we strive to attract potential contributors to the cluster and contribute back to the open-source community.

III. Federation

The EdgeNet testbed provides a platform for running Next-Generation Internet (NGI) experiments. It currently provides more than 50 nodes scattered around the world, with access to the internet and private network connectivity between each nodes. EdgeNet is based on Kubernetes and experimenters deploys Docker containers. In order to provide a unified way of accessing the testbed, we implement the GENI Aggregate Manager (AM) API v3 mandated by the Fed4FIRE+ project. Through this API, experimenters can deploy and access containers on nodes of their choice.

The Kubernetes API is natively a declarative API: users define the desired state of a resource (e.g., a container) and a control-loop (also called the *controller*) keeps the resource in sync. In contrast, the AM API is imperative by nature: users perform actions that change the state of a resource, such as *allocate*, *provision*, *shutdown*, etc. In order to reconcile the two paradigms, we have created an AM API that manages Kubernetes objects on the behalf of the users.

[3]Kubernetes https://kubernetes.io/

A. Mapping GENI resources to Kubernetes resources

The AM specification defines three main kinds of resources: users, slices, and slivers. Slivers are collections of compute resources and users are given rights to create slivers in slices. In our case, we seek to offer to experimenters SSH access to Docker containers running on EdgeNet. Thus, a sliver maps to a collection of three Kubernetes objects:

- A *Deployment* object defines the specification of the container: image, node and CPU architecture. Kubernetes will ensure that a container matching these specifications will always be running.
- A *Service* object maps an available TCP port of the host node, to the SSH port of the container.
- A *ConfigMap* object holds the SSH keys of the user and is mounted on `~/.ssh/authorized_keys`.

Users can choose a specific Docker image, node and CPU architecture (aarch64 or x86_64). If none are specified, the AM API will choose a default image and Kubernetes will create the container on any available node.

B. Resource expiration

Slivers have an expiration time, which can be extended by performing the renew action. When a sliver expires, the associated resources must be deleted. Kubernetes has currently no way of specifying expiration dates for objects and automatically deletes them (excepted for Jobs resources). To work around this, we run a garbage collector goroutine which periodically checks for the expiration of the resources and deletes them.

C. Object naming

Object names are derived from the first 8 bytes of the SHA512 hash of the sliver name. This allows the creation of objects with names that are valid in the GENI AM specification, but not in Kubernetes which allows only alphanumeric chars.

D. Non-standard TLS certificated workaround

As per the specification, clients are authenticated using client TLS certificates. The certificates provided by Fed4FIRE+ contain non-standard OIDs (Object IDentifiers) which cannot be parsed by the Go X.509 parser. Specifically, the *Authority Information Access* OID contains numbers larger than 32-bits. This makes it impossible to authenticate clients using Go code and prevents the use of reverse proxies written in Go such as the popular Caddy[4] and Traefik[5] proxies. Upon discussion with the Fed4FIRE+ administrators, it became clear that there is no immediate plan for the Fed4FIRE+ OID format to change. To work around this issue, we place an NGINX[6] reverse proxy in front of our AM API server. This proxy performs client TLS authentication and forwards the request to the AM API server by including the certificate in a custom `X-Fed4Fire-Certificate` HTTP header. This information is then passed to external tools by the AM API, such as `xmlsec1` to validate credentials and authorize users.

E. Deployment

Our AM API is publicly deployed[7] and its source code is available on GitHub.[8] It can easily be deployed on any Kubernetes cluster to federate that cluster with Fed4FIRE+. That is to say, the API that we provide is general to Kubernetes and is not specific to just the EdgeNet Kubernetes cluster.

IV. HOME NETWORKS

Kubernetes is designed for centralized data centers, and on that basis, the system assumes that cluster nodes share a local network. Put another way, it does not provide an off-the-shelf solution for nodes on different networks without public IP addresses. This introduces two communications problems:

- A node behind a NAT box can access control plane nodes, but a control plane node cannot access that node.
- Containers on a node behind a NAT box are unreachable from other cluster nodes.

Kubernetes has an extensible architecture that allows developing and using plugins. A container network interface (CNI) plugin typically establishes networking between pods. EdgeNet employs VMware's Antrea CNI[9] for this purpose. Antrea uses the OVS bridge on every node. Furthermore, it forms a virtual ethernet (veth) pair for each pod, a gateway (gw) to the node subnet, and a tunnel (tun) for inter-node communications.[10]

The OVS bridge forwards packets using veth pairs on the node regarding local pod traffic. If traffic is toward an external destination, packets to be routed are forwarded through the gateway port. Antrea benefits from source network address translation (SNAT) so that the pod IP address is preserved. In terms of inter-node communication, tunnels encapsulate and decapsulate packets. Fig. 1a depicts the traffic flow where every node has a public IP address.

However, if a node is behind a NAT box on another network, it blocks inter-node communication. To overcome this problem, we set up a VPN tunnel between every pair of nodes in the cluster. We settled on using the WireGuard VPN [11] for multiple reasons:

- its performance: it offers throughput 5 times higher than OpenVPN and 1.1 times higher than IPsec on the same configuration;[11]
- its simplicity: it only requires generating of public/private pair of keys for each client and does not requires a PKI infrastructure as OpenVPN certificate-based authentication does;

[4]Caddy https://caddyserver.com
[5]Traefik https://doc.traefik.io/traefik
[6]NGINX https://www.nginx.com

[7]EdgeNet AM API https://fed4fire.edge-net.io.
[8]EdgeNet AM software https://github.com/EdgeNet-project/fed4fire.
[9]VMware Antrea https://antrea.io/
[10]The VMware Antrea architecture https://antrea.io/docs/main/docs/design/architecture/
[11]The WireGuard benchmarking https://www.wireguard.com/performance

- its integration in the Linux kernel: it is natively integrated in the kernel since Linux 5.6, requiring no additional kernel modules.

Our solution provides IPv4 and IPv6 peer-to-peer communication for every nodes of the cluster (with the exception of NAT-to-NAT communications, see Sec. IV-B) over the public IPv4 internet, as seen in Fig. 1b.

A. Bootstrapping VPN peers

A node must have established VPN connectivity with the rest of the cluster before Kubernetes starts. To achieve this, an agent present on each node performs the following actions on boot:

1) It checks if a public/private key pair has ever been generated. If none, it generates one and saves it for subsequent boots.
2) It checks if an IPv4/IPv6 pair of address has ever been generated. If none, it queries the cluster to get the list of used IP address in the VPN network, and chooses a random pair of addresses amongst the ones available. Randomization allows to reduce the risk of two new nodes choosing the same IP address if booted at the same time.
3) It publishes its public key, its private IP address pair, and its public IP address to the cluster by creating a *VPNPeer* Kubernetes resource.
4) It queries the list of *VPNPeer* resources and configures the tunnel interface to add each peer.

B. NAT-to-NAT communications

In our current deployment, a VPN tunnel can be established between two public nodes, or between one public node and one NATted node. Establishing a connection between two NATted nodes requires the use of an external server to exchange port numbers and perform UDP hole punching. We will implement such a technique in future iterations.

V. Node Deployment

Anyone can contribute a node to the EdgeNet public cluster. Deploying a node involves setting up Kubernetes, the container runtime, the network, and joining the cluster. In order to make this process as easy and as transparent as possible, we developed a set of Ansible playbooks[12] that automatically perform all of these steps. Our current deployment procedure is as follows and will spawn a node in under 5 minutes:

- The user runs the `bootstrap.sh` script on the target node. This script will install Ansible, fetch the deployment playbook and run it.
- The playbook will install Kubernetes, the container runtime, and a dedicated *node agent* written in Go.
- The node agent will start and configure the VPN as described in Sec. IV-A, and it will join the cluster.

This architecture is very flexible:

- The bootstrap script works on any recent Debian or RedHat-based Linux distribution, on aarch64 or x86_64 architectures, and it doesn't requires any preinstalled software.
- The bootstrap script URL can be passed to cloud-init[13] to automatically set up the instances on first boot.
- The Ansible playbooks can be used in a standalone fashion for bulk deployment or node maintenance.
- The Ansible playbooks can be used together with Packer[14] to create VM images with our software pre-installed.

In our current implementation, no input is required from the user and anyone can contribute a node without an EdgeNet account.

VI. Node Robustness

In comparison to other testbeds such as PlanetLab, EdgeNet nodes are not expected to be maintained by system administrators. A node can be deployed in a user's home with limited debugging time and knowledge. As such, we must ensure that the nodes are able to self-heal in case of problems. We have currently identified two main issues: unresponsive nodes and file system corruption. We describe below two tentative solutions that we will try to implement in the next iteration of EdgeNet.

A. Unresponsive nodes

A node can become unresponsive if some application consumes all of its resources, or if an excessive amount of network traffic saturates the network interface and the CPU. We are investigating the use of the hardware watchdog present on Raspberry Pi and ODROID single-board computers to automatically reboot unresponsive nodes. The kernel periodically sends heartbeats to the watchdog. If the watchdog stops receiving heartbeats, it power cycles the node. This procedure can fix unresponsive nodes without any user intervention.

B. File system corruption

Single-board computers often use flash-based memory such as SD cards or eMMCs. These memories are prone to failure as they are usually not designed for continuous random writes over long period of times. When these memories fail, the file system is corrupted and the systems stops working properly. It is also possible that a user improperly changes the configuration of a node.

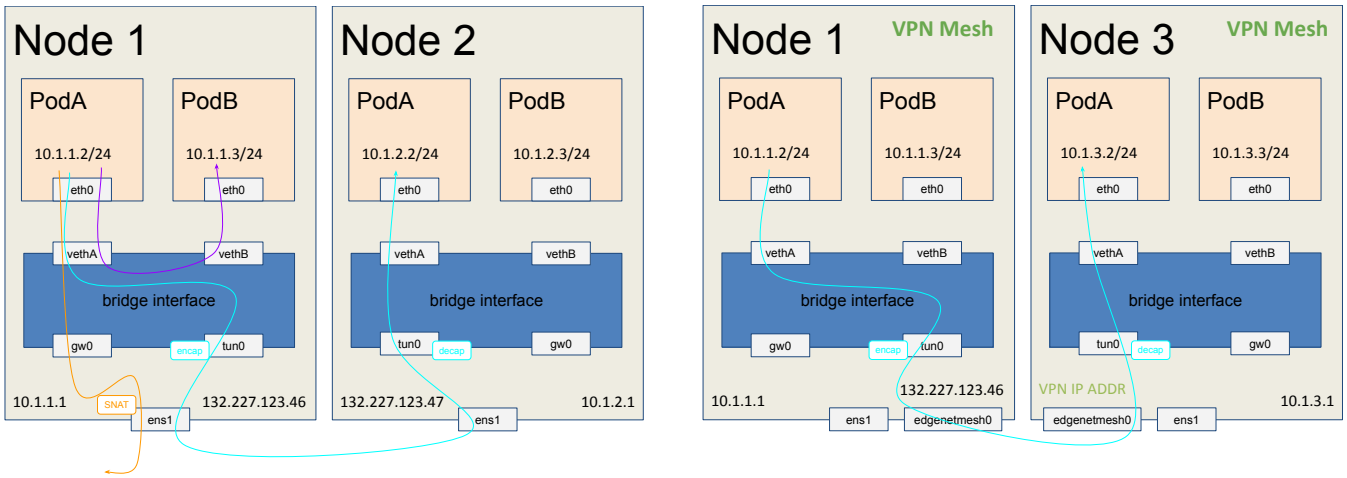
To handle this issue, we are exploring the possibility of booting nodes over the internet. The petitboot[15] bootloader can boot a kernel and a live disk image over the internet. The disk image and the kernel would live in RAM, and the node's flash storage would only be used to store container data. If the flash memory fails, the node would still be accessible and the user would only have to replace the SD card.

[12] EdgeNet node provisioning https://github.com/EdgeNet-project/node.

[13] cloud-init ttps://cloud-init.io

[14] Packer https://www.packer.io

[15] petitboot https://github.com/open-power/petitboot



(a) Pod traffic where every node has a public IP address. Orange is for pod-to-external, cyan is for inter-node, and purple is for intra-node traffic.

(b) Pod traffic where a node has a private IP address. Cyan shows that traffic is routed through edgenetmesh both on the source and destination nodes.

Fig. 1. Traffic flow in EdgeNet. The drawings are inspired by VMware architecture documentation.[10]

This method also has the benefit of making system updates very easy: deploy a new disk image on the server and reboot the remote nodes. If the update fails, roll back the disk image on the server. The main concern with this approach is security and how to authenticate the boot server as well as the disk images.

VII. Conclusion

We have introduced challenges related to nodes at the edge that distributed testbed providers face: provision, access, and maintenance. Three contributions, federation, home networks, and node deployment, addressed these issues. Our aggregate manager (AM), which federates EdgeNet into Fed4FIRE+, shows that our testbed is capable of running in concert with other testbeds. A node agent configures a virtual private network, thus enabling nodes behind NAT boxes to participate in a cluster with the help of the native VPN peer controller in Kubernetes. Finally yet importantly, a node can join a cluster in less than ten minutes via our node deployment procedure that includes installing the above-mentioned agent. In conclusion, these three features allow the testbed to reach out to a wider community and scale up the cluster, including edge nodes typically blocked by NAT boxes. This paper also discusses how to tackle remotely maintaining edge nodes as future work.

References

[1] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016. [Online]. Available: https://doi.org/10.1109/MC.2016.145

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016. [Online]. Available: https://doi.org/10.1109/JIOT.2016.2579198

[3] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017. [Online]. Available: https://doi.org/10.1109/MC.2017.9

[4] Fed4FIRE+, "Fed4FIRE+ testbeds portal," 2022. [Online]. Available: https://www.fed4fire.eu/

[5] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir, "Experiences building PlanetLab," in *Proc. 7th Symposium on Operating Systems Design and Implementation (USENIX OSDI)*, 2006. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/1298455.1298489

[6] S. Fdida, T. Friedman, and T. Parmentelat, "OneLab: An open federated facility for experimentally driven future internet research," in *New Network Architectures: The Path to the Future Internet*, ser. Studies in Computational Intelligence, T. Tronco, Ed. Springer, 2010, vol. 297, pp. 141–152. [Online]. Available: https://doi.org/10.1007/978-3-642-13247-6

[7] R. McGeer, M. Berman, C. Elliott, and R. Ricci, Eds., *The GENI Book*. Springer, 2016. [Online]. Available: https://doi.org/10.1007/978-3-319-33769-2

[8] P. Mueller and S. Fischer, "Europe's mission in next-generation networking with special emphasis on the German-Lab project," in *The GENI Book*, R. McGeer, M. Berman, C. Elliott, and R. Ricci, Eds. Springer, 2016, ch. 21. [Online]. Available: https://doi.org/10.1007/978-3-319-33769-2_21

[9] A. Nakao and K. Yamada, "Research and development on network virtualization technologies in Japan: VNode and FLARE projects," in *The GENI Book*, R. McGeer, M. Berman, C. Elliott, and R. Ricci, Eds. Springer, 2016, ch. 23. [Online]. Available: https://doi.org/10.1007/978-3-319-33769-2_23

[10] A. Leon-Garcia and H. Bannazadeh, "SAVI testbed for applications on software-defined infrastructure," in *The GENI Book*, R. McGeer, M. Berman, C. Elliott, and R. Ricci, Eds. Springer, 2016, ch. 22. [Online]. Available: https://doi.org/10.1007/978-3-319-33769-2_22

[11] J. A. Donenfeld, "WireGuard: Next Generation Kernel Network Tunnel," in *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, 2017. [Online]. Available: https://doi.org/10.14722/ndss.2017.23160