EdgeNet: A Multi-Tenant and Multi-Provider Edge Cloud

Berat Can Şenel
Sorbonne Université
Paris, France

Maxime Mouchet
Sorbonne Université
Paris, France

Justin Cappos
NYU Tandon School of Engineering
New York, United States

Olivier Fourmaux
Sorbonne Université
Paris, France

Timur Friedman
Sorbonne Université
Paris, France

Rick McGeer
US Ignite
Washington, DC, United States

ABSTRACT

EdgeNet is a public Kubernetes cluster dedicated to network and distributed systems research, supporting experiments that are deployed concurrently by independent groups. Its nodes are hosted by multiple institutions around the world. It represents a departure from the classic Kubernetes model, where the nodes that are available to a single tenant reside in a small number of well-interconnected data centers. The free open-source EdgeNet code extends Kubernetes to the edge, making three key contributions: multi-tenancy, geographical deployments, and single-command node installation. We show that establishing a public Kubernetes cluster over the internet, with multiple tenants and multiple hosting providers is viable. Preliminary results also indicate that the EdgeNet testbed that we run provides a satisfactory environment to run a variety of experiments with minimal network overhead.

CCS CONCEPTS

• **Computer systems organization → Distributed architectures**.

KEYWORDS

Edge Computing, Edge Cloud, Distributed Systems, Kubernetes

ACM Reference Format:
Berat Can Şenel, Maxime Mouchet, Justin Cappos, Olivier Fourmaux, Timur Friedman, and Rick McGeer. 2021. EdgeNet: A Multi-Tenant and Multi-Provider Edge Cloud. In *4th International Workshop on Edge Systems, Analytics and Networking (EdgeSys'21), April 26, 2021, Online, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3434770.3459737

1 INTRODUCTION

Traditional cloud architectures are concerned with providing on-demand access for external users to compute and storage resources located in centralized data centers. This model is challenged with the emergence of new applications, such as content delivery, peer-to-peer multicast, distributed messaging, and the Internet of Things (IoT). These applications are sensitive to latency and they benefit from compute resources that are geographically close to the user.

Edge clouds complement centralized clouds by placing computation and storage resources close to users or data sources, to offer high bandwidth and low latency between cloud computing resources, data producers, and data consumers. For well over a decade, the networking and distributed systems research communities have deployed a series of wide-area edge cloud testbeds, such as PlanetLab Central [21], PlanetLab Europe [6], Geni [16], G-Lab [17], V-Node [18] and Savi [14]. These testbeds degraded over time for two reasons: they relied on dedicated hardware, which required on-site support, and they used custom control frameworks.

The requirement for dedicated hardware has led to maintenance and scalability issues. On one hand, the cost of purchase and replacement of servers discouraged people from contributing to the testbeds. On the other hand, the human resources required to maintain servers over the long term were costly. In the long term, the testbeds were not able to scale.

The testbeds also relied on custom software for managing the nodes and the experiments. These control frameworks were typically written and maintained by a small team of researchers, and used by a relatively small community of distributed-systems experimenters. Such software gets quickly outdated, and is only improved by the small communities of the original developers and dedicated experimenters. This lack of standardization resulted in a waste of resources, as each testbed has to be documented individually, and experimenters had to learn testbed-specific knowledge.

We argue that the solution to make the next generation of distributed testbeds viable is to rely on widely used control frameworks and on inexpensive virtual machines. This approach reduces the cost per site, as the virtual machines can be created for free on an existing infrastructure, and requires almost no maintenance. This solves the scalability problem by lowering the entry barrier for contributing new nodes. In addition, there is no need to maintain extensive testbed-specific documentation and software, as most of it is reused from external projects. This also benefits the users of a testbed, as by learning how to use it, they gain industry-valuable knowledge. However, today's common cloud frameworks treat nodes as homogeneous entities in a centralized data center, whereas a key feature of edge testbeds is their heterogeneity and geo-diversity. Up until now been no production-ready framework for edge cloud testbeds.

In this article, we introduce the EdgeNet free open-source software that allows an edge cloud to be deployed onto virtual machines as worker nodes, with Kubernetes as the control framework. EdgeNet offers a novel architecture for edge computing, which directly addresses the sustainability and maintenance issues described above. Since an EdgeNet worker node is a VM running at a site's local cloud, the expense of maintaining a dedicated hardware resource disappears; in fact, an EdgeNet VM is just another VM among many running at that cloud, requiring no marginal maintenance commitment. Using Kubernetes directly addresses the maintenance, upgrade, and training issues of control frameworks mentioned above. A worldwide community of developers maintains and extends Kubernetes, and extensive documentation and training resources are available on the internet.

The challenge that EdgeNet overcomes is that the use of Kubernetes as an edge cloud control framework breaks Kubernetes' central design assumptions in three important areas:

- Kubernetes was designed for homogeneous nodes, where computation could be rapidly moved from one node to another in the cluster. For EdgeNet, a node's physical location is a first-class design parameter, and so nodes are heterogeneous in physical location. Further, experimenters must be able to choose where their worker nodes are placed.
- Kubernetes was designed so that head nodes and all worker nodes were within the same cluster, so communication was on layer 2 and latencies were on the order of microseconds. In our deployment, layer-2 connectivity is not available, and inter-node latencies are on the order of 10s of milliseconds.
- Kubernetes was designed for a single-tenant deployment. In our cluster, there are mutually-untrusting multiple tenants.

Kubernetes is widely used in central data centers for container organization. EdgeNet brings it to the edge with three key contributions: (1) A node selection feature that makes it possible to deploy containers on nodes based on their locations. It is easy to configure deployments to schedule pods to cities, countries, continents, or latitude-longitude polygons. (2) A single-command node installation procedure lowers the entry barrier for the institutions wishing to contribute nodes to the cluster, thus simplifying the establishment of an edge cloud. (3) Multi-tenancy at the edge, providing isolation between tenants and sharing limited resources fairly, so that multiple organizations can concurrently benefit from the edge cloud.

EdgeNet's open-source software is freely available on GitHub[1], and the testbed that it supports[2] is open to researchers worldwide.

In Sec. 2 we review similar efforts for bringing Kubernetes to the edge. In Sec. 3 we present how EdgeNet is implemented in terms of Kubernetes custom resources. In Sec. 4 we describe the current status of the platform, and in Sec. 5 we evaluate the performance of EdgeNet.

2 RELATED WORK

Much work has been done to adapt container technology to edge clouds [2, 19, 20]. A growing number of publications address various aspects, such as IoT task offloading, enabling long-running functions in containerization for IoT devices, and designing a scheduler for Kubernetes in Industrial IoT, which allows edge cloud nodes to consume less energy and to deploy applications in less time than usual. [4, 11, 12].

In Cloud4IoT [22], the authors discuss a platform using containers to deploy, orchestrate, and dynamically configure software components related to IoT and data-intensive applications while providing scalability in the cloud layer. The main difference with EdgeNet is that it concentrates on IoT solutions, meaning the device edge, whereas EdgeNet's core use case is for more somewhat more powerful nodes positioned at the network edge.

Kristiani et al. [13] provide an implementation of an edge computing architecture by taking advantage of OpenStack and Kubernetes to cover the cloud, the edge cloud, and the device edge cloud. This implementation reduces the workload on the cloud side by assigning data processing tasks to the edge front to be performed at the edge of the network. The focus is different from EdgeNet's, looking at communication between three layers (Cloud, Edge, and Device Edge) and on task offloading.

KubeEdge [26], a project that is incubating within the Cloud Native Computing Foundation, offers a Kubernetes-based infrastructure that brings specific cloud capabilities to the edge. It aims to overcome edge computing challenges such as limited resources and non-connectivity. KubeEdge uses Docker as its containerization technology, Kubernetes as the orchestrator, and Mosquitto for IoT devices talking to edge nodes. Again, the focus is different from EdgeNet's selective deployment, easy node installation, and multi-tenancy contributions.

Another project that brings Kubernetes to the edge is Rancher's Lightweight Kubernetes [10], which focuses on lightweight Kubernetes for resource-constrained nodes, which is a feature that we would like to provide through EdgeNet, but which is not the subject of the present study.

3 KUBERNETES EXTENSIONS

Kubernetes is an orchestrator: its main goal is to deploy containers on nodes, and to ensure that the desired number of containers is running at any given time. It is based on the concept of *resources* and *controllers*. A resource defines the state of the system, and a controller ensures that the system stays in the desired state. By default, Kubernetes provides a number of resources, such as *deployments* which define a desired quantity of containers. When a deployment object is created, the associated controller will select available nodes, and create the *pod* objects that represent a group of containers running on a single node. The pod will then run the actual containers on the node. If a node were to fail, the deployment controller would create new pod objects on suitable nodes.

Kubernetes can be extended in many ways. The original EdgeNet prototype [3, 9] was built as an overlay *on top* of Kubernetes. A custom web server would handle EdgeNet-specific features such as user registration, and create the appropriate Kubernetes object using the Kubernetes `kubectl` command-line tool. The current EdgeNet version extends Kubernetes in a more idiomatic way, by defined its own custom resources and controllers (Fig. 1). This makes it possible to interact with EdgeNet by using nothing beyond standard Kubernetes tools.

The EdgeNet philosophy is to stick as close to possible to vanilla Kubernetes. This allows its users to benefit from the wealth of Kubernetes documentation available online, and to leverage their existing knowledge. Users such as students also gain skills that will be transferable to industry. EdgeNet's developers have a smaller base of custom code to maintain over time. All EdgeNet development is done in the Go language, used by Kubernetes and many other projects in the community, so as to make it easy to understand and to contribute to.

[1]https://github.com/EdgeNet-project
[2]https://edge-net.org/

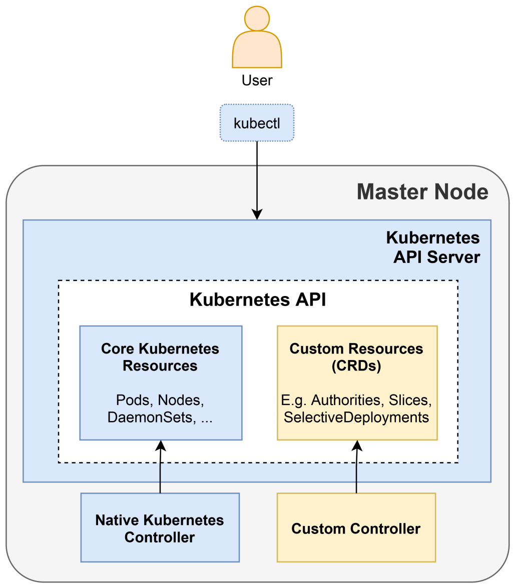
User
kubectl
Master Node
Kubernetes API Server
Kubernetes API
Core Kubernetes Resources
Pods, Nodes, DaemonSets, ...
Custom Resources (CRDs)
E.g. Authorities, Slices, SelectiveDeployments
Native Kubernetes Controller
Custom Controller

Figure 1: EdgeNet extends Kubernetes by using custom resources and custom controllers. The user interacts with EdgeNet through the standard Kubernetes API.

In using the EdgeNet software to power a testbed, we aim to provide:

- a service that makes it easy for users to take advantage of geographic node diversity for node selection (Sec. 3.1);
- an easy node deployment procedure, to lower the entry barrier for new contributors and increase the diversity and the geographic coverage of the cluster's nodes (Sec. 3.2);
- a multi-tenant structure that allows different groups to register to use the cluster and obtain a share of its resources for their applications (Sec. 3.3).

3.1 Location-based node selection

EdgeNet's main value as compared to pure Kubernetes is its ability to deploy containerized software to a widely distributed set of nodes rather than to nodes that are all grouped in a centralized datacenter. To do so, the users must be able deploy containers based upon a node's location. Note that Kubernetes offers the ability to choose specific nodes based on *labels*, but it is up to the cluster administrators to attach the relevant labels to the nodes. EdgeNet achieves location-based deployments with two components: (1) a service that geolocates nodes and attaches the appropriate labels to them; (2) a *selective deployment* resource, which allows the users to select amongst nodes based on geographic criteria.

3.1.1 Node labeler. In order to be able to select nodes by their location, EdgeNet attaches multiple labels to the nodes according to their city, state/region, country, continent, and coordinates. This is done by the *node labeler*, a controller that watches the cluster for new nodes, or for node IP updates, and geolocates the nodes. By default, the nodes are located by IP address, using the MaxMind GeoLite2 database.If the node is running at a known cloud provider, we use the location of the data center in which the node is running, obtained from the instance metadata. For example, we assign the following labels to a node located at Stanford:

- edge-net.io/city=Stanford
- edge-net.io/state-iso=CA
- edge-net.io/country-iso=US
- edge-net.io/continent=North_America
- edge-net.io/lat=n37.423000
- edge-net.io/lon=w-122.163900

3.1.2 Selective deployment. Once the labels are attached to the nodes, they can be used to select specific nodes when creating resource objects such as deployments. However Kubernetes' built-in selectors are limited, consisting only of equality comparison (*city = Stanford*), and set inclusion (*city in (Paris, Stanford)*). In order to enable additional selection criteria, we introduce the *selective deployment* resource.

A selective deployment is comprised of a resource type (a deployment, for example), and of geographic queries associated with a number of nodes. It can target nodes by continent, country, region, and city, as well as polygons that are described using latitudes and longitudes. In order to efficiently determine which nodes lie with a polygon, we use the Point-in-Polygon algorithm [8]. When a selective deployment is created, the controller finds the relevant nodes and creates the appropriate resource objects. If a node goes down, the controller re-configures its resource objects in order to start a new pod on a new node in the same geographic area.

3.2 Node contribution

An EdgeNet node is a machine, physical or virtual, that runs `kubelet`, the Kubernetes agent, and a container runtime. EdgeNet currently uses Docker as the container runtime, although this configuration will be deprecated in future Kubernetes versions. EdgeNet will be updated to use `containerd` instead.

While installing Docker and kubelet is relatively easy for a user comfortable with the command line, we seek to make the process as easy and error-proof as possible in order to encourage contributions. To do so we describe an EdgeNet configuration through a set of Ansible playbooks. Ansible is a popular configuration management tool, and is commonly used to deploy Kubernetes clusters (see, for example, the Kubespray project [23]). By using Ansible, we can reuse community-maintained playbooks for deploying Docker and kubelet, and we can benefit from the ecosystem of tools that integrate with Ansible. Most notably, we make use of the Packer tool to build ready-to-use virtual machines from the playbooks.

An EdgeNet node can be deployed in under 5 minutes, as detailed in Sec. 5.2. We currently support the deployment of nodes on the major Linux distributions (CentOS, Fedora, and Ubuntu) on x86 machines with a public IPv4 address. We will extend support to ARM hosts and machines with non-public IP addresses in the future.

3.2.1 Pre-built cloud images. We provide prebuilt cloud images for the major cloud providers (Amazon Web Services, Google Cloud Platform and Microsoft Azure). These images allow any user of these clouds to deploy an EdgeNet node with no configuration required. On first boot, a `NodeContribution` object is created and the node is allowed to join the cluster.

3.2.2 Bootstrap script. For users who want to deploy nodes on their own machines, we provide a *bootstrap* script that installs Ansible, downloads the playbooks, and runs them. Note that users comfortable with Ansible can directly use the EdgeNet playbooks to deploy a node.

3.3 Multi-tenancy

Edge clouds aim to provide nodes that are geographically close to the end users. However, achieving good geographic coverage as a single tenant can be complicated, as it requires obtaining access to nodes across the world. A contribution in exchange for consumption model, whereby multiple tenants contribute nodes, is a solution to this problem. In this section we describe the current multi-tenancy model of EdgeNet and we discuss an improved model.

3.3.1 Current multi-tenancy model. EdgeNet offers multi-tenancy at the edge, in which authorizations to use the platform are handed out hierarchically. EdgeNet administrators approve the establishment of *authorities*, which represent an institution or a research team. The administrator of the authority, in turn, approves the creation of individual user accounts for the local users who they know. For structuring an authority, EdgeNet borrows two concepts from the PlanetLab Europe [5] and Fed4FIRE [7] platforms: *teams* and *slices*. A team under an authority allows its members to independently create slices even if they are not authorized to do so at authority scope. A slice allows its participants to create their workload resource objects to deploy the pods to the cluster. Team members and slice participants may belong to different authorities, thus allowing the collaboration across authorities. Each authority is assigned a total resource quota, and slices can only be created within the available quota, ensuring that resources are shared amongst authorities.

3.3.2 Future multi-tenancy model. We plan to update the current multi-tenancy model, inherited from the realm of networking and distributed systems testbeds, to bring it closer to the to the work of Kubernetes multi-tenancy working group. First, we will align the naming, for instance replacing "authority" with "tenant". Next, we would create a *core namespace* for each tenant and merge slices and teams into *subnamespaces*. The core namespace would allow each tenant to create workloads directly, as opposed to the current design, which obliges a tenant to create a slice first. The subnamespaces could be arbitrarily nested, starting from the core namespace, and each would inherit the RBAC (Role-Based Access Control) and network policy settings from its parent. This would allow a more fine-grained and hierarchical control over the resources, in comparison to the slices and teams model. We hope to achieve three main objectives: (1) position EdgeNet closer to the Kubernetes multi-tenancy working group in better contribute our work back to the community; (2) simplify the EdgeNet code base; (3) simplify the onboarding of new users by reducing the number of steps required to run experiments on the platform.

4 PLATFORM STATUS

EdgeNet is up and running at over 40 nodes worldwide including 5 in Europe, 1 in Brasil, 1 in Australia, and the others in the United States. The current nodes are hosted by universities and the Geni [16] testbed in the USA. In addition, several experiments have been conducted on EdgeNet over the past year:

CacheCash (NYU Tandon School)
CacheCash [1] is a blockchain-based CDN that involves the end users themselves into the network to serve content through their own machines. 30 EdgeNet nodes have been used to deploy CacheCash and perform extensive latency, throughput, and resource usage measurements.

Internet scale topology discovery (Sorbonne Université)
The Multilevel MDA-Lite Paris Traceroute [25] tool, an evolved version of the well known `traceroute` tool, was used for continuous surveys the internet from EdgeNet nodes. Also, Diamond-Miner [24], which conducts high speed internet-scale route traces, has been deployed on 7 EdgeNet nodes as part of a production internet topology measurement system.

Cyberlab Honeypot Experiment (University of Ljubljana)
The honeypot experiment uses EdgeNet to expose fake SSH servers on the internet and detect malicious activities.

Reveal topologies of remote networks
(Université de Liège - Institut Montefiore)
This experiment sends ICMP probes from EdgeNet nodes to perform internet topology discovery.

Darknet Watch (University College Dublin)
This bandwidth-intensive experiment conducts measurement on the I2P anonymous network.

NDT Client (M-Lab)
EdgeNet supports continuous measurements by the M-Lab NDT (Network Diagnostic Tool) [15] client that measures download and upload speeds.

The EdgeNet cluster currently consists of 72 vCPUs and 96 GB of memory. A typical EdgeNet node has 2 GB of memory, 2 vCPUs, and 15 GB of storage. Since its start, EdgeNet has supported up to 7 parallel experiments. Scaling the system is ongoing work: currently, if a new experiment requires a node that does not have enough available resources to handle a new experiment, the system does not deploy the experiment on that node. In future work, we plan to fix this limitation by automatically instantiating a new EdgeNet node on the overloaded site (if resources are available). The node contribution process (Sec. 3.2) that automates the deployment of a new node goes in that direction.

5 BENCHMARKS

EdgeNet is a global Kubernetes cluster with nodes all over the world communicating over the internet. This differs from the classic Kubernetes use case with nodes located in well-interconnected data centers. In this section we study how the cluster performs in terms of deploying containers, adding new nodes, and networking.

5.1 Time to deploy an experiment

We measured the time necessary to schedule and run pods using selective deployments in 5 use cases: 1 pod anywhere in the European Union, 5 pods anywhere in the United States, 20 pods anywhere in the United States, and 1 and 20 pods in a polygon with 18 vertices provided in the GeoJSON format. We also simulated node failures by stopping the Kubernetes agent, *kubelet*, on them. The results are presented in Tab. 1.

We first see that creating a selective deployment, including the node selection, is done in a very short time, always under half a second. Thus, selecting nodes geographically incurs almost no overhead over the classical Kubernetes selector.

Next, when the number of pods is small, the time to create the daemon set and to get all the pods up and running is under 10 seconds. However, when 20 pods are requested, this time jumps up to 80 seconds. This is explained by the lack of resources on some nodes of the cluster. Pods typically are scheduled in under 10 seconds, but if the node where a pod is scheduled has a bottleneck of the resources in terms of CPU and memory, it spikes the time for the pod to be up and running.

The cluster can detect a node failure in under 45 seconds, and in all cases but one that we have tested, restart the pods on a new node in under 10 seconds. In the case of 5 pods in the US, it took 80 seconds to recover the pods. In that instance, the Kubernetes scheduler removed all pods from nodes and redeployed them even though only one node was changed and others remained the same in the selector. We assume that changing the selector, the node affinity, in the workload spec caused this unexpected behavior. Further investigation is needed to understand the underlying cause.

These findings raise questions about the ability of the default Kubernetes scheduler to meet edge cloud requirements and indicates the need for an edge scheduler for edge-specific applications. A health-check mechanism between the master and worker nodes could also be put in place to reduce the time needed to detect a node failure.

5.2 Time to deploy a node

An EdgeNet node can be deployed in two ways: from a pre-built cloud image, or from a *bootstrap* shell script in a dedicated virtual machine. We perform our measurements for both methods on an AWS (Amazon Web Services) *t2.small* instance with 1 vCPU and 2 GB of memory, in the *eu-west-1* (Ireland) region.

With the bootstrap script, counting from the launch of the script, it takes 2 minutes and 50 seconds to install Docker and Kubernetes, 3 minutes and 5 seconds for the node to be detected by the cluster, and 3 minutes and 50 seconds to be ready to deploy containers.

With the prebuilt AMI (Amazon Machine Image), counting from the instance creation, it takes 40 seconds for the node to be detected by the cluster, and 1 minute 20 seconds for the node to be ready to deploy containers.

5.3 Cluster network performance

EdgeNet relies on the Calico network plugin to enable intra-cluster communications, that is, communications between the pods of the cluster. Calico establishes a private *pod network* on each node, and exchanges routes through full-mesh BGP peerings. Packets between the pod networks are sent over the internet using an IP-in-IP encapsulation. In this section we investigate the performance impact of this encapsulation in terms of throughput and RTT. Due to space constraints, we only present a relevant subset of our measurement results. The full dataset is available online[3].

[3]https://github.com/EdgeNet-project/benchmarks

Throughput. We used the standard iPerf3 tool to measure the throughput between nodes of the cluster and a node in Paris, and between nodes and a public server hosted by an ISP in France. This allows us to assess the impact of the IP-in-IP encapsulation on the network throughput. The results are presented in Fig. 2.

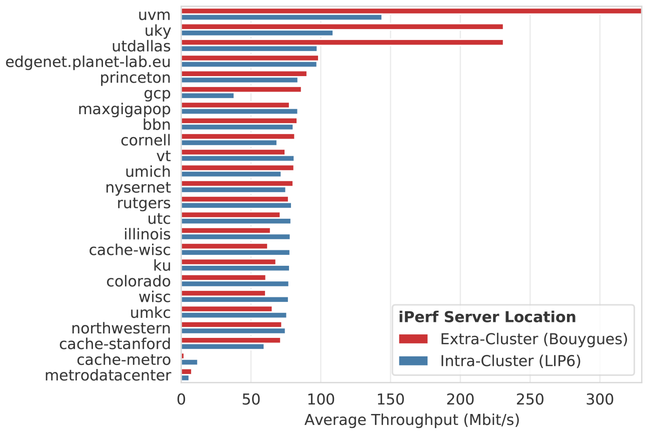

Figure 2: Average throughput between intra and extra-cluster targets. The throughput was measured over 10 seconds with iPerf 3. *Bougyues* is a 10Gbit/s iPerf server hosted by an ISP, and *LIP6* is an EdgeNet node, both located in Paris, France. Measurement towards the EdgeNet node are routed through the Calico IP-in-IP tunnel.

In almost every case, the intra and extra cluster throughput is similar. However, for the *uvm*, *uky*, and *utdallas* nodes the extra-cluster throughput is much higher than the intra-cluster one. Similarly, the *gcp* node located in the Google Cloud Platform, has significantly worse intra-cluster performance. Further investigations are needed to determine if this is due to congestion in the network, different IP routes for the two destinations, or to the IP-in-IP encapsulation.

Round-trip time. Similarly to the throughput measurements, we measure the round-trip time between the nodes of the cluster and a node in Paris, France. The measurements are done towards the external IP of the node, in which case the packets are directly sent over the internet, and towards the internal IP of the node, in which case the packets are encapsulated before being sent. The results are presented in Fig. 3.

In all cases the minimum RTT is identical between the external and internal IP, indicating that the IP-in-IP encapsulation has no effect on the RTT. This makes EdgeNet suitable for research on computer networks, as the platform overhead is minimal. We have also done this measurement towards 3 nodes in the USA, and 1 node in Brazil. We observe no overhead for the nodes in the USA, but for the node in Brazil the intra-cluster delay is consistently longer by 4 milliseconds. Further investigation is needed to understand why this is the case for this node.

6 CONCLUSION

In this paper we introduced EdgeNet, a multi-tenant and multi-provider edge cloud based on Kubernetes. EdgeNet extends Kubernetes through custom resources and controllers, making it usable

Table 1: Time in seconds to schedule and run pods on EdgeNet using selective deployments.

Selection	Selective Deployment	Daemon Set	Pod	Node Failure Detection	Daemon Set Recovery	Pod Recovery
1 pod in the EU	0.42	3.9	3.4	36.7	4.6	3.5
5 pods in the US	0.19	10.1	10.5	45.5	79.9	79.2
20 pods in the US	0.47	60.6	79.6	39.8	8.2	8.3
1 pod in a polygon	0.18	6.7	6.2	41.1	8.3	8.3
20 pods in a polygon	0.33	57.2	56.8	37.4	9.8	9.8

Selective Deployment is the time to create a selective deployment object and select the nodes. *Daemon Set* is the time for the selective deployment to create the daemon set and its pods. *Pod* is the time between the creation of the first pod and having the last pod in the running state. *Node Failure Detection* is the time to detect a node failure. The last two columns are the time to reconfigure the daemon sets and to recreate the pods after the node failure.

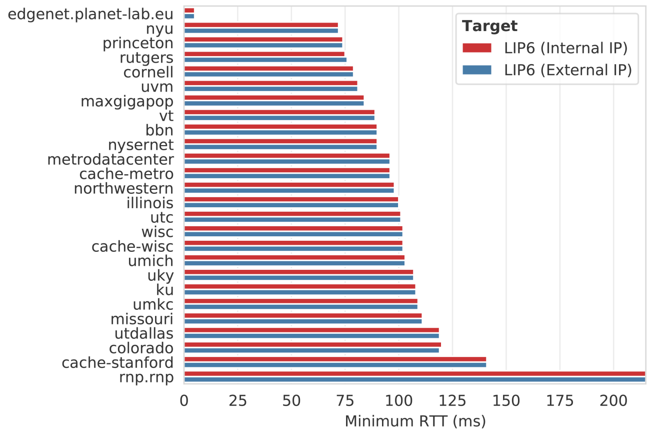

Figure 3: Minimum RTT between EdgeNet nodes and a node in Paris, France. The RTT was computed over 100 ICMP ping measurements. Pings towards the external IP are directly sent over the internet, while pings towards the internal IP are routed through the Calico IP-in-IP tunnel.

with nothing else than the standard Kubernetes tools. In its current state, the EdgeNet cluster offers reasonable performance with minimal overhead, and is suited to all kinds of experiments on networked systems. EdgeNet shows that a public Kubernetes cluster with nodes distributed over the world works.

Several steps can be taken to improve EdgeNet, and more generally Kubernetes at the edge. First, an edge-specific scheduler for Kubernetes could be designed so as to minimize the deployment times and to better take into account the limitations of each edge node. Second, EdgeNet does not currently support nodes with a private IP address and located behind a NAT. This scenario is common for edge nodes and should be addressed in a future EdgeNet update.

ACKNOWLEDGMENTS

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. The authors also wish to thank Kevin Vermeulen for his review of the paper draft and his valuable comments.

REFERENCES

[1] G. Almashaqbeh. 2019. *CacheCash: A Cryptocurrency-based Decentralized Content Delivery Network*. Ph.D. Dissertation. Columbia University.
[2] A. Bavier, R. McGeer, and G. Ricart. 2016. PlanetIgnite: A Self-Assembling, Lightweight, Infrastructure-as-a-Service Edge Cloud. In *Proc. ITC '16*. 130–138.
[3] J. Cappos, M. Hemmings, R. McGeer, A. Rafetseder, and G. Ricart. 2018. EdgeNet: A global cloud that spreads by local action. In *Proc. SEC 2018*. IEEE, 359–360.
[4] C. Dupont, R. Giaffreda, and L. Capra. 2017. Edge computing in IoT context: Horizontal and vertical Linux container migration. In *Proc. GIoTS 2017*. 1–4.
[5] PlanetLab Europe. 2021. PLE. https://www.planet-lab.eu/
[6] S. Fdida, T. Friedman, and T. Parmentelat. 2010. OneLab: An Open Federated Facility for Experimentally Driven Future Internet Research. In *New Network Architectures: The Path to the Future Internet*, Tania Tronco (Ed.). Studies in Computational Intelligence, Vol. 297. Springer-Verlag, 141–152.
[7] Fed4FIRE. 2021. Fed4FIRE. https://www.fed4fire.eu/
[8] D. Rex Finley. 2021. Point-in-Polygon Algorithm—Determining Whether a Point Is Inside a Complex Polygon. http://alienryderflex.com/polygon/
[9] T. Friedman, R. McGeer, B. C. Senel, M. Hemmings, and G. Ricart. 2019. The EdgeNet System. *Proc. ICNP '19* (2019), 1–2.
[10] K3s. 2021. K3s: Lightweight Kubernetes. https://k3s.io/
[11] P. Karhula, J. Janak, and H. Schulzrinne. 2019. Checkpointing and Migration of IoT Edge Functions. In *Proc. EdgeSys '19* (Dresden, Germany). Association for Computing Machinery, New York, NY, USA, 60–65.
[12] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and M. Atiquzzaman. 2020. KEIDS: Kubernetes-Based Energy and Interference Driven Scheduler for Industrial IoT in Edge-Cloud Ecosystem. *IEEE Internet of Things Journal* 7, 5 (2020), 4228–4237.
[13] E. Kristiani, C.-T. Yang, Y. Wang, and C.-Y. Huang. 2019. *Implementation of an Edge Computing Architecture Using OpenStack and Kubernetes: ICISA 2018*. 675–685.
[14] A. Leon-Garcia and H. Bannazadeh. 2016. SAVI Testbed for Applications on Software-Defined Infrastructure. In *The GENI Book*. Springer-Verlag, New York, Chapter 22.
[15] M-Lab. 2021. Network Diagnostic Tool. www.measurementlab.net/tests/ndt/
[16] R. McGeer, M. Berman, C. Elliott, and R. Ricci (Eds.). 2016. *The GENI Book*. Springer International Publishing.
[17] P. Mueller and S. Fischer. 2016. Europe's Mission in Next-Generation Networking With special emphasis on the German-Lab Project. In *The GENI Book*. Springer-Verlag, New York, Chapter 21.
[18] A. Nakao and K. Yamada. 2016. *Research and Development on Network Virtualization Technologies in Japan: VNode and FLARE Projects*. 563–588.
[19] C. Pahl, S. Helmer, L. Miori, J. Sanin, and B. Lee. 2016. A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters. In *Proc. FiCloud '16*. 117–124.
[20] C. Pahl and B. Lee. 2015. Containers and Clusters for Edge Cloud Architectures – A Technology Review. In *Proc. FiCloud '15*. 379–386.
[21] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir. 2006. Experiences building PlanetLab. In *Proc. OSDI '06*. USENIX Association.
[22] D. Pizzolli, G. Cossu, D. Santoro, L. Capra, C. Dupont, D. Charalampos, F. De Pellegrini, F. Antonelli, and S. Cretti. 2016. Cloud4IoT: A Heterogeneous, Distributed and Autonomic Cloud Platform for the IoT. In *Proc. CloudCom '16*. 476–479.
[23] Kubernetes SIGs. 2021. kubespray. github.com/kubernetes-sigs/kubespray
[24] K. Vermeulen, J. P. Rohrer, R. Beverly, O. Fourmaux, and T. Friedman. 2020. Diamond-Miner: Comprehensive Discovery of the Internet's Topology Diamonds. In *Proc. NSDI '20*. USENIX Association, Santa Clara, CA, United States, 479–493.
[25] K. Vermeulen, S. D. Strowes, O. Fourmaux, and T. Friedman. 2018. Multilevel MDA-lite Paris traceroute. In *Proc. IMC '18*. 29–42.
[26] Y. Xiong, Y. Sun, L. Xing, and Y. Huang. 2018. Extend Cloud to Edge with KubeEdge. In *Proc. SEC 2018*. 373–377.