



## **Arena: a kubernetes-based testbed for evaluating application deployment across the computing continuum**

Chih-Kai Huang, Konstantinos Krouti, Stella Markopoulou, Konstantinos Tserpes,  
Georgios Bouloukakis

### **► To cite this version:**

Chih-Kai Huang, Konstantinos Krouti, Stella Markopoulou, Konstantinos Tserpes, Georgios Bouloukakis.  
Arena: a kubernetes-based testbed for evaluating application deployment across the computing continuum.  
IEEE International Conference on Communications (ICC), IEEE, May 2026, Glasgow, United Kingdom. ⟨hal-05517164v2⟩

**HAL Id: hal-05517164**

**<https://inria.hal.science/hal-05517164v2>**

Submitted on 18 Feb 2026

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons CC BY 4.0 - Attribution - International License

# Arena: A Kubernetes-based Testbed for Evaluating Application Deployment across the Computing Continuum

Chih-Kai Huang<sup>†\*</sup>, Konstantinos Krouti<sup>‡</sup>, Stella Markopoulou<sup>‡</sup>, Konstantinos Tserpes<sup>§</sup>, and Georgios Bouloukakis<sup>¶\*</sup>

<sup>\*</sup>Télécom SudParis, <sup>†</sup>Télécom Paris, Institut Polytechnique de Paris, France

<sup>‡</sup>AEGIS IT Research GmbH, Germany

<sup>§</sup>National Technical University of Athens, Greece

<sup>¶</sup>University of Patras, Greece

Email: chih-kai.huang@telecom-paris.fr, {c.krouti, s.markopoulou}@aegisresearch.eu, tserpes@mail.ntua.gr, gbouloukakis@upatras.gr

**Abstract**—This paper introduces Arena, a Kubernetes-based testbed for evaluating application deployment across computing continuum environments (IoT/Edge/Cloud). Arena enables the emulation of diverse computing nodes using Docker containers and leverages Kubernetes for testbed management. Arena integrates the Chaos Mesh framework to simulate network characteristics and Prometheus with Grafana tools for monitoring and visualization purposes. Experiments on the Grid’5000 platform with a Google microservice application demonstrate that Arena’s container-based emulation achieves similar resource usage patterns to virtual machine-based nodes, and its network chaos injection effectively enforces network constraints. Results highlight Arena’s capability to provide a practical and reproducible environment for testing containerized applications across diverse computing continuum nodes.

**Index Terms**—Computing Continuum, Kubernetes, Testbed, Emulation, IoT, Cloud, Edge

## I. INTRODUCTION

The rapid advancement of the Internet of Things (IoT) industry has led to a widespread increase in IoT devices across different domains, including smart cities, industrial automation, and intelligent transportation systems. These devices continuously produce massive volumes of data that may require low-latency processing and real-time decision-making from applications/services processing this data. Relying on traditional cloud computing for deploying such applications/services is often insufficient as the long-distance data transmission causes high latency and bandwidth consumption [1].

Edge and Fog computing have emerged as paradigms that place computing resources, such as Edge servers, closer to data sources, thereby enabling lower latency and reducing network workload [2]. By deploying applications at the Edge, faster response times can be achieved, and the amount of data transmitted to cloud data centers is reduced. Nevertheless, the computing and storage capacity of Edge resources is typically limited. To address more demanding workloads, cloud platforms may complement system deployment by

providing scalable resources. The combination of Edge and Cloud resources gives rise to the concept of the **computing continuum** [3], in which computation for data from IoT devices is flexibly distributed across heterogeneous resources to balance performance, scalability, and cost.

While the computing continuum is promising, developers need reliable methods to test and evaluate their applications before actual deployment, as running directly in such environments can be costly, time-consuming, and impractical because fully integrated IoT-Edge-Cloud platforms are rarely available. To achieve this, two main methods are commonly used: simulation and emulation-based approaches. Several simulation frameworks have been proposed, including iFogSim [4] and its newer version iFogSim2 [5], MobFogSim [6], and Edge-CloudSim [7]. Emulation-based methods can better reflect the application behavior by executing actual software components in controlled environments. Representative examples include FogBed [8], EmuFog [9], and iContinuum [10].

However, simulation typically relies on models and assumptions, which may limit the accuracy of application behavior and lack system-level realism. Emulation-based approaches address these issues by executing real software components. However, they still face several challenges. In particular, existing works often lack integration with mainstream orchestrators, which restricts the emulation of realistic application lifecycle management, or depend on Virtual Machine (VM) deployments that may introduce additional overhead and longer provisioning times.

To evaluate the deployment of applications across the computing continuum in terms of latency, scalability, and performance, this paper introduces Arena, a Kubernetes-based testbed that enables the systematic measurement of metrics such as CPU and memory usage, request latency, and throughput in a single host before deploying them in real-world IoT-Edge-Cloud resources. Arena relies on Docker containers to emulate IoT-Edge-Cloud computing nodes and leverages Kubernetes to manage both container-based nodes and system

designers' applications. To simulate the dynamic network conditions of the computing continuum and provide comprehensive monitoring functionality, Arena integrates Chaos Mesh for network simulation and Prometheus with Grafana for monitoring and performance visualization.

To demonstrate the usefulness of Arena, we conduct two experiments via real-world application deployment in the Grid'5000 platform [11]. The first experiment compares container-based and VM-based node emulation to verify whether Arena can have similar resource utilization patterns. VM-based nodes are considered the reference configuration, as they are commonly used in production deployment [12], [13]. The second experiment applies network bandwidth constraints through Chaos Mesh to emulate dynamic network conditions and observe their impact on application throughput. The results confirm that Arena provides a reliable and reproducible application evaluation. Moreover, the Arena testbed has been adopted within the PANDORA EU project (<https://pandora-heu.eu>), highlighting its relevance and maturity in realistic pilot cases involving computing continuum resources [14].

The remainder of this paper is organized as follows: Section II discusses the overview of background and related work. In Section III, we describe the design and components of the Arena testbed. We evaluate our solution in Section IV, and summarize the paper's conclusions and future work in Section V.

## II. RELATED WORK

Computing continuum is a distributed, heterogeneous, and dynamic environment that creates significant challenges for developers to evaluate their applications directly on real-world infrastructures. To study and validate applications across such environments, developers typically rely on two main approaches: **simulation**, which allows cost-effective and scalable exploration by running software models under different scenarios, and **emulation**, which enables realistic evaluation on actual or virtualized infrastructures while leveraging partially controlled simulation, such as network conditions and failures. **Simulation-based Approaches.** *iFogSim* toolkit [4] is based on the *CloudSim* [15] framework to model and simulate IoT and Edge infrastructures with hierarchical topologies and is able to configure CPU, RAM, uplink/downlink bandwidth, and busy/idle power to estimate energy consumption. To support mobility and microservice orchestration, an extension version of this simulation toolkit *iFogSim2* [5] was proposed.

Another related extension from *iFogSim* is *MobFogSim* [6], which enables the modeling of device mobility and service migration in fog computing environments. Similar to the *iFogSim*, *EdgeCloudSim* [7] is also based on the *CloudSim* framework. It supports Wireless Local Area Network (WLAN) and Wide Area Network (WAN) network settings, user mobility models, and application offloading strategies, enabling it to evaluate the latency-sensitive mobile edge computing scenarios.

However, these simulation frameworks remain limited in their ability to capture the complexity and dynamics of real-world edge computing environments. In particular, they lack system-level realism and basically do not allow for testing real applications or runtime behavior, making them insufficient for replicating real deployment scenarios.

**Emulation-based Approaches.** *FogBed* [8] leverages an emulation-based approach to prototype cloud-fog systems in virtualized environments. *FogBed* is built on Docker containers and the Mininet network emulator [16], where Docker containers act as fog nodes and Mininet provides flexible network topology emulation. Similarly, *EmuFog* [9] is another emulator for edge/fog environments, built on Docker and the MaxiNet, an extension of Mininet. The framework supports flexible placement of fog nodes and enables the evaluation of latency-sensitive services under different network conditions.

However, neither *FogBed* nor *EmuFog* is integrated with mainstream container orchestrators such as Kubernetes. This limits their ability to reproduce realistic application lifecycle management, automated scaling, and resource scheduling that are common in real deployments. Moreover, without integration into the Kubernetes ecosystem, they cannot easily benefit from its continuously evolving set of tools, which restricts their extensibility and long-term maintainability.

*iContinuum* [10] is an intent-based emulator that combines Software-Defined Networking (SDN), using the Open Network Operating System (ONOS) [17] controller and Mininet, with Kubernetes to model cloud-edge continuum environments. Nevertheless, *iContinuum* relies on VMs to emulate diverse IoT-Edge-Cloud computing nodes. The use of VMs introduces additional virtualization overhead and slower provisioning, which reduces hosting efficiency during experimentation.

In contrast, Arena leverages Docker containers as computing nodes managed by Kubernetes. This design offers better scalability, lower overhead, and faster provisioning. Moreover, it allows developers to evaluate application deployments under realistic conditions by leveraging Kubernetes' scheduling and auto-scaling capabilities, the programmable network behaviors provided by Chaos Mesh, and integrated monitoring and performance visualization through Prometheus and Grafana.

## III. ARENA TESTBED

This section presents an overview of the Arena testbed and introduces its main components.

Figure 1 presents the overview of the Arena testbed. The design is driven by the need to provide a platform for reproducing computing environments with diverse resources. Arena users can emulate a wide range of deployment scenarios by selecting the number of computing nodes and configuring their CPU and memory capacities to represent IoT, Edge, or Cloud environments. They can also specify latency, bandwidth, and jitter values between nodes or applications, enabling the testbed to simulate the network conditions ranging from 4G/5G telecom networks to wired-connected and stable networks. Thanks to Kubernetes, Arena uses simple YAML

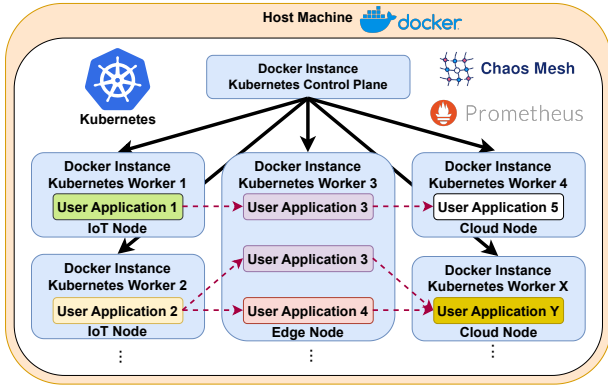


Fig. 1. An overview of Arena Testbed.

files to describe application deployment and network settings. Therefore, experiments can be repeated or adjusted easily, and Arena users can share configurations to ensure consistent evaluation. Arena allows users to study application behavior and orchestration strategies, while monitoring tools support observation and analysis under heterogeneous and dynamic computing continuum environments. The following discusses the main components of Arena.

**Kubernetes:** Kubernetes is a de facto and open-source container orchestrator that has been widely adopted in industry and academic research [18], [19]. We use Kubernetes in Arena to automate the deployment and management of containerized applications. It contains two main roles: the control plane and the worker node. The control plane manages the overall cluster state, while the worker node provides the computing resources to run containerized workloads. Kubernetes relies on container runtimes, such as containerd and CRI-O, to execute containers on each node [20]. In Arena, each node is emulated as a Docker container, inside which a container runtime is installed to support the execution of containerized applications. This *containers-in-containers* design enables the emulation of diverse computing nodes in a single host while maintaining the same operational semantics as a real Kubernetes cluster, thereby ensuring portability and realism.

To configure the computing resources of each emulated node for representing IoT, Edge, and Cloud devices, Arena configures the node’s CPU and memory capacities through Kubernetes’ allocatable mechanism. This defines the portion of node resources available for pods<sup>1</sup> after reserving system overhead, allowing Arena to emulate heterogeneous computing capabilities.

**Chaos Mesh:** Chaos Mesh is an open-source chaos engineering framework designed for a Kubernetes cluster [21]. It provides a wide range of fault injection mechanisms, including pod failures, network constraints, I/O errors, and resource stress. Chaos Mesh enables its users to inject specific failure scenarios that can be consistently reproduced across

experiments. In Arena, Chaos Mesh is integrated to emulate the dynamic and unstable conditions of the networking and applications in the computing continuum. For example, Arena users can configure network latency and bandwidth to simulate edge-to-cloud links or execute node crashes to evaluate the resilience of orchestration strategies. Moreover, this framework is a Cloud Native Computing Foundation (CNCF) incubating project, which demonstrates its growing maturity and community support [22].

**Prometheus and Grafana:** Prometheus [23] and Grafana [24] are open-source monitoring and visualization tools that work well in the cloud-native ecosystem. Prometheus provides a time-series database and a flexible query language (PromQL) for collecting and analyzing metrics from a Kubernetes cluster and its running applications. It continuously scrapes<sup>2</sup> metrics from targets, storing them in its databases. In Arena, Prometheus is deployed to monitor its worker nodes and application resource utilization. These data can then be the foundation for evaluating orchestration policies and detecting anomalies during experiments. Moreover, Prometheus is a CNCF graduated project, demonstrating its maturity and reliability, and it has been widely adopted in several research works [25]–[27].

The Grafana tool complements the Prometheus framework by providing an interactive and highly customizable dashboard system. It allows Arena users to visualize metric trends, explore data correlations, and obtain deeper insights from experimental results.

#### IV. EVALUATION OF ARENA TESTBED

This section presents the experimental evaluation of the proposed Arena testbed. The Arena testbed code is available at: <https://github.com/satrai-lab/arena>.

##### A. Experimental Setup

The experiments are running on the Grid’5000 large-scale platform [11]. To simulate an Arena end-user testing application deployment across the computing continuum, we run all experiments on a server equipped with dual Intel Xeon E5-2630L v4 CPUs (10 cores each) and 128 GB of RAM, running Debian 11.11 with the 5.10.0-35-amd64 kernel as the host operating system. Arena is implemented using kind [28] (v0.22.0), and experiment deployment includes Kubernetes v1.33.2, containerd v2.1.3 with Runc v1.3.0, Cilium v1.17.6, kube-prometheus-stack v75.18.1, Chaos Mesh v2.28.0, Docker v28.0.4, and QEMU v5.2.0. Each experiment is repeated five times, and the results are reported as the average values across all runs.

##### B. Emulation Fidelity of Container-based vs. VM-based Nodes

To evaluate the fidelity of Arena’s container-based node emulation, we compare the resource usage of container and VM-based node deployments. This experiment aims to verify whether container-based node emulation in Arena can

<sup>1</sup>A pod is the smallest deployable unit in Kubernetes.

<sup>2</sup>In Prometheus, the term “scrape” refers to collecting or pulling monitoring data from a target such as applications at regular intervals.



reproduce the performance characteristics of VM-based nodes, which are commonly used in realistic computing environments [12], [13].

In this experiment, we deploy an Arena testbed (containers as Kubernetes nodes) and a VM-based Kubernetes cluster using identical configurations. Our test application is Online Boutique (v0.10.3), a microservice demo developed by Google [29], comprising eleven microservices and one load generator, implemented in various programming languages listed in Table I. To minimize network interference and inter-node communication, each cluster is limited to a single worker node and one control plane node, both configured with 4 vCPUs and 16 GiB of memory. All microservices and essential system components are scheduled on the worker node, while monitoring and management components run on the control plane.

We use the *loadgenerator* to emulate user activities. It continuously sends HTTP requests to the *frontend*, where each virtual user randomly performs actions such as visiting the homepage, browsing products, changing currencies, adding items to the cart, viewing the cart, and checking out. We change both the number of virtual users and the spawn rate with 1, 100, and 200 to evaluate microservice performance. For data collection, we use the Arena built-in Prometheus framework configured with a 5 second scrape interval to monitor resource consumption. Each experiment runs for 10 minutes.

TABLE I  
MICROSERVICES AND THEIR PROGRAMMING LANGUAGES

Service	Language
adservice	Java
cartservice	C#
checkoutservice	Go
currencyservice	Node.js
emailservice	Python
frontend	Go
loadgenerator	Python/Locust
paymentservice	Node.js
productcatalogservice	Go
recommendationservice	Python
redis	C
shippingservice	Go

1) *Results of Microservice Resource Usage:* Figure 2 shows the results of the 200 virtual users and compares the CPU and memory consumption of twelve microservices when deployed in Arena’s container-based node and VM-based node. Overall, the container-based deployment exhibits slightly higher CPU utilization for most microservices, with a few exceptions such as *cartservice* and *currencyservice*, where VM-based execution incurs higher overhead. We can find that *frontend* has the highest CPU usage, with the container-based consuming about 193.5 m<sup>3</sup> on average 17.6% higher than VM-based at 164.6 m.

Among all microservices, the largest difference in CPU usage between container-based and VM-based nodes is ob-

served in *checkoutservice* (66.0% higher than in the VM-based deployment), followed by *recommendationservice* (59.5%), *redis* (51.0%), and *adservice* (47.5%). Although the relative percentage differences appear large, the absolute differences in CPU usage remain small, typically within a few tens of millicores. For instance, the CPU usage of the *checkoutservice* is 14.1 m in the container-based node and 8.5 m in the VM-based node. The difference is only 5.6 m, which corresponds to 0.0056 CPU cores and can be regarded as being within the error margin.

Similarly, Figure 2 shows that memory utilization is highly similar across both environments. Although individual microservices may exhibit slight variations, the overall memory consumption trends remain stable and consistent.

These results demonstrate that container-based nodes behave very similarly to VM-based nodes in terms of CPU and memory resource usage of applications. This indicates that using containers as nodes can capture the performance characteristics of widely adopted VM-based environments with only minor variations.

2) *Results of Computing Node Resource Usage:* Figures 3 and 4 compare the CPU and memory utilization of computing nodes in container-based and VM-based environments under an increasing number of users. CPU utilization of the worker node rises continuously in both environments as the workload scales. However, container-based nodes show slightly higher usage in all cases. We can see that the CPU usage of worker nodes in container-based/VM-based deployments under workloads of 1, 100, and 200 users are 3.5% / 1.8%, 18% / 10%, 20% / 16%, respectively. While the CPU usage of each microservice in the container-based setup is close to the VM-based one, the overall usage of the worker node remains higher. This may indicate that the container-as-node approach introduces some overhead, even though per-application behavior appears similar. Further investigation of these effects is left for future work. Regarding memory usage, the results remain almost unchanged across different numbers of users in both environments, and the overall usage is nearly the same between the container-based and standard VM-based setups.

The scalability of the Arena testbed is limited by host resources, and running multiple Arena testbeds concurrently on the same machine may cause interference. To ensure clean and reproducible experiments, we therefore recommend deploying a single Arena testbed per host. Supporting multiple testbeds on a single machine is left for future work.

### C. Validation of Network Chaos Injection

To verify the functionality of network chaos injection provided by Chaos Mesh, we run an experiment using a data pipeline composed of IoT data producers, Kafka (v3.7.0), Logstash (v8.17.3), and Elasticsearch (v8.17.3).

In this experiment, we launch an Arena testbed with one control plane node (8 vCPUs and 16 GiB of memory) and three worker nodes. The three worker nodes are set with 2 vCPUs/4 GiB (IoT), 4 vCPUs/8 GiB (Edge), and 8 vCPUs/16 GiB of

<sup>3</sup>Kubernetes uses “m” (millicores) to represent CPU usage, where 1000m equals one full CPU core.

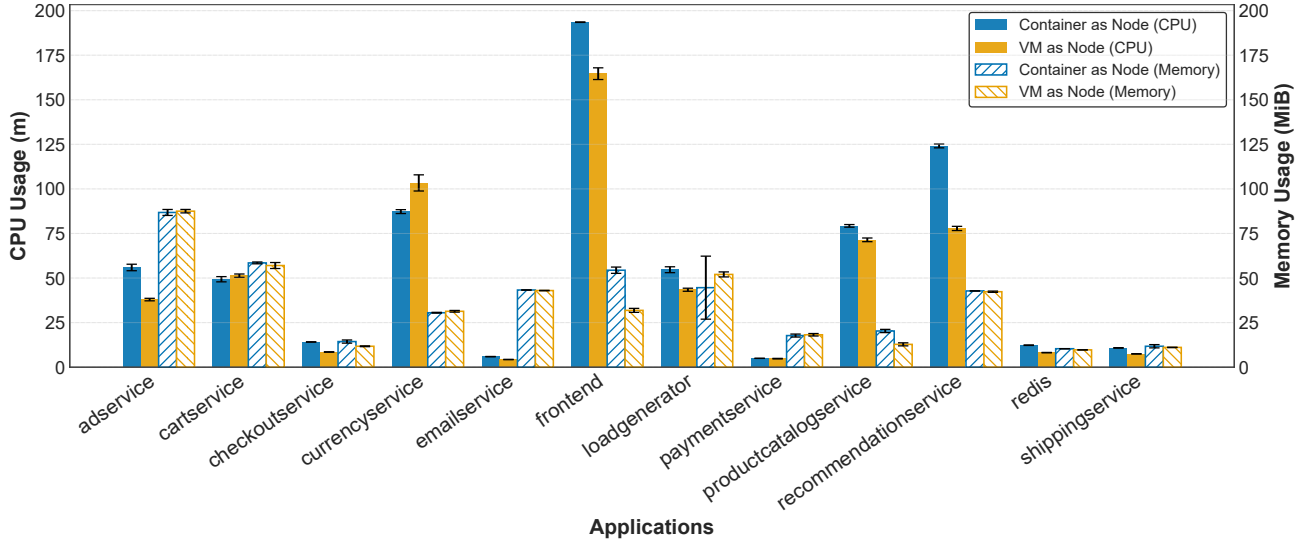


Fig. 2. CPU and memory utilization of twelve microservices running on container-based and VM-based nodes under a workload of 200 virtual users.

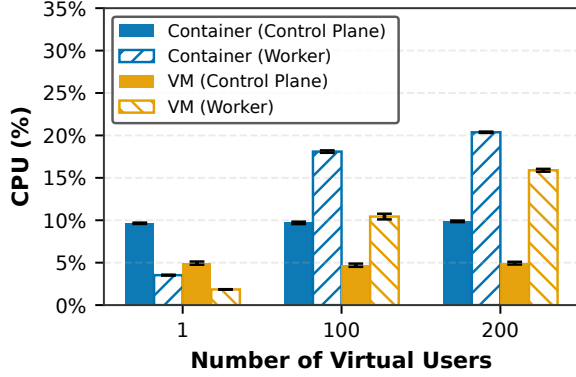


Fig. 3. CPU utilization of container-based and VM-based nodes under different numbers of virtual users.

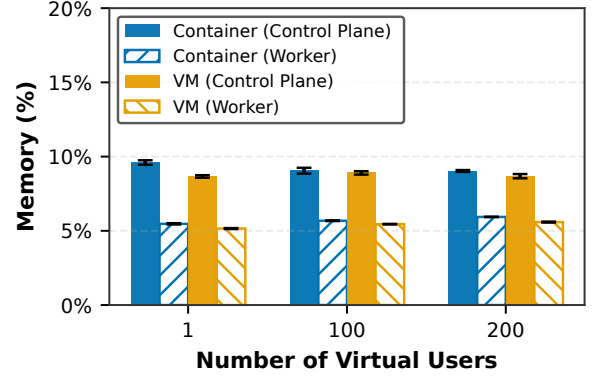


Fig. 4. Memory utilization of container-based and VM-based nodes under different numbers of virtual users.

memory (Cloud), respectively, to emulate heterogeneous environments across the computing continuum. The IoT worker hosts the IoT data producers, the Edge node runs Kafka and Logstash, and the Cloud node deploys Elasticsearch.

The IoT data producers continuously send data to Kafka at a rate of about 16 Mbps. Kafka forwards them to Logstash for processing, and finally, the processed data are sent and stored in Elasticsearch. We inject network bandwidth constraints into Logstash, where the outbound bandwidth is configured to 1 Mbps, 2 Mbps, 5 Mbps, 10 Mbps, and a baseline case without limitation. In all configurations, the buffer size and limit parameters<sup>4</sup> are both set to 10,000 bytes. To verify that the bandwidth limitation takes effect, we continuously monitor the number of documents<sup>5</sup> in Elasticsearch every second using a Python script for ten minutes, and calculate the overall

<sup>4</sup>Buffer and limit define the temporary data buffer and the maximum queue length for network traffic, respectively.

<sup>5</sup>Each document corresponds to one data record inserted into Elasticsearch.

pipeline throughput as the total number of indexed documents divided by the experiment duration.

Figure 5 shows the average throughput of the data pipeline under different outbound bandwidth limitations injected into Logstash. As the bandwidth decreases, the overall throughput drops significantly, confirming that the Chaos Mesh bandwidth constraint is effectively enforced. We can see that the throughput reduces from 41.9 docs/s without limitation to 5.7 docs/s at 1 Mbps and gradually recovers as the bandwidth increases. By using Chaos Mesh, Arena's users can easily emulate various network conditions, such as constrained bandwidth, latency, and packet loss, to evaluate the behavior and resilience of distributed applications under realistic network environments across the computing continuum.

## V. CONCLUSIONS AND FUTURE WORK

This paper presents Arena, a Kubernetes-based testbed designed to evaluate the deployment of containerized appli-

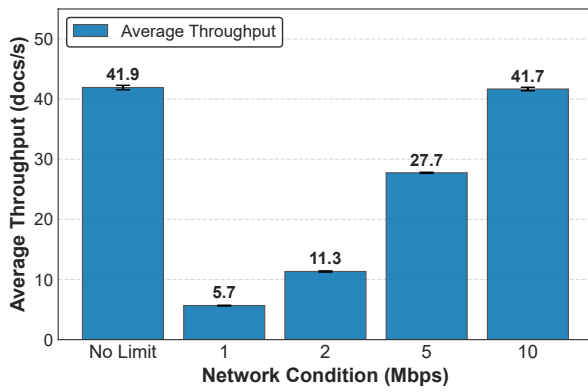


Fig. 5. Average throughput of the data pipeline under different outbound bandwidth limitations applied to Logstash.

cations across emulated computing continuum environments. Through its container-based node emulation and integration with Kubernetes and its ecosystem, Arena provides a unified environment for evaluating application performance and behavior under heterogeneous settings. The experiments on the Grid'5000 platform with actual deployment confirm that Arena has resource usage patterns similar to standard VM-based Kubernetes deployment and that its network control mechanisms operate as expected.

In future work, we plan to pursue three directions. First, the current version of Arena supports only single-host deployment. We will extend it to multi-host environments to enable large-scale application evaluations. Second, we aim to develop a topology-aware network modeling framework that allows users to specify application-level topologies so that Arena can automatically configure network characteristics such as latency, bandwidth, and packet loss between components. Third, we intend to explore heterogeneous hardware support in Arena, including the emulation of ARM-based and GPU nodes under x86 hardware.

#### ACKNOWLEDGMENT

This work has received funding from the European Union's Horizon Europe research and innovation actions under grant agreements No. 101135775 (PANDORA) and No. 101168465 (MEDIATE). Experiments presented in this paper were partly carried out using the Grid'5000, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see <https://www.grid5000.fr>).

#### REFERENCES

- [1] R. K. Naha *et al.*, "Fog Computing: Survey of Trends, Architectures, Requirements, and Research Directions," *IEEE Access*, vol. 6, 2018.
- [2] F. C. Andriulo *et al.*, "Edge Computing and Cloud Computing for Internet of Things: A Review," in *Informatics*, vol. 11, no. 4, 2024.
- [3] A. Ullah *et al.*, "Orchestration in the Cloud-to-Things Compute Continuum: Taxonomy, Survey and Future Directions," *Journal of Cloud Computing*, vol. 12, no. 1, 2023.
- [4] H. Gupta *et al.*, "iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in the Internet of Things, Edge and Fog Computing Environments," *Software: Practice and Experience*, vol. 47, no. 9, 2017.
- [5] R. Mahmud *et al.*, "iFogSim2: An Extended iFogSim Simulator for Mobility, Clustering, and Microservice Management in Edge and Fog Computing Environments," *Journal of Systems and Software*, vol. 190, 2022.
- [6] C. Puliafito *et al.*, "MobFogSim: Simulation of Mobility and Migration for Fog Computing," *Simulation Modelling Practice and Theory*, vol. 101, 2020.
- [7] C. Sonmez *et al.*, "EdgeCloudSim: An Environment for Performance Evaluation of Edge Computing Systems," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 11, 2018.
- [8] A. Coutinho *et al.*, "FogBed: A Rapid-Prototyping Emulation Environment for Fog Computing," in *Proc. IEEE Intl. Conference on Communications*, 2018.
- [9] R. Mayer *et al.*, "EmuFog: Extensible and Scalable Emulation of Large-Scale Fog Computing Infrastructures," in *Proc. IEEE Fog World Congress*, 2017.
- [10] N. Akbari *et al.*, "iContinuum: An Emulation Toolkit for Intent-based Computing Across the Edge-to-Cloud Continuum," in *Proc. IEEE Intl. Conference on Cloud Computing*, 2024.
- [11] D. Balouek *et al.*, "Adding Virtualization Capabilities to the Grid'5000 Testbed," in *Cloud Computing and Services Science*, 2013, vol. 367.
- [12] Google Cloud. (2025) GKE Cluster Architecture - About the Nodes. [Online]. Available: <https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture>
- [13] Amazon Web Services. (2025) Create a Managed Node Group for Your Cluster. [Online]. Available: <https://docs.aws.amazon.com/eks/latest/userguide/create-managed-node-group.html>
- [14] G. Bouloukakis *et al.*, "Unlocking AIoT Efficiency in the Computing Continuum-the PANDORA framework," in *Proc. Intl. on the Internet of Things*, 2025.
- [15] R. N. Calheiros *et al.*, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and experience*, vol. 41, no. 1, 2011.
- [16] K. Kaur *et al.*, "Mininet As Software Defined Networking Testing Platform," in *Proc. Intl. Conference on Communication, Computing & Systems*, 2014.
- [17] P. Berde *et al.*, "ONOS: Towards an Open, Distributed SDN OS," in *Proc. Workshop on Hot Topics in Software Defined Networking*, 2014.
- [18] V. Silverthorne and S. Hendrick, "Cloud Native 2024: Approaching a Decade of Code, Cloud, and Change," Cloud Native Computing Foundation and The Linux Foundation, Tech. Rep., March 2025.
- [19] C.-K. Huang and G. Pierre, "UnBound: Multi-Tenancy Management in Scalable Fog Meta-Federations," in *Proc. IEEE/ACM Intl. Conference on Utility and Cloud Computing*, 2024.
- [20] The Kubernetes Authors. (2025) Container runtimes. [Online]. Available: <https://kubernetes.io/docs/concepts/containers/#container-runtimes>
- [21] The Chaos Mesh Authors. (2025) Chaos Mesh. [Online]. Available: <https://chaos-mesh.org/>
- [22] Cloud Native Computing Foundation. (2025) Chaos Mesh Maturity Level. [Online]. Available: <https://www.cncf.io/projects/chaosmesh/>
- [23] The Prometheus Authors. (2025) Prometheus. [Online]. Available: <https://prometheus.io/>
- [24] The Grafana Authors. (2025) Grafana. [Online]. Available: <https://grafana.com/>
- [25] C.-K. Huang and G. Pierre, "Aggregate Monitoring for Geo-Distributed Kubernetes Cluster Federations," *IEEE Transactions on Cloud Computing*, vol. 12, no. 4, 2024.
- [26] C.-K. Huang and G. Pierre, "AdapPF: Self-Adaptive Scrape Interval for Monitoring in Geo-Distributed Cluster Federations," in *Proc. IEEE Symposium on Computers and Communications*, 2023.
- [27] M. A. Tamiru *et al.*, "mck8s: An Orchestration Platform for Geo-Distributed Multi-Cluster Environments," in *Proc. Intl. Conference on Computer Communications and Networks*, 2021.
- [28] The kind Authors. (2025) kind. [Online]. Available: <https://kind.sigs.k8s.io/>
- [29] The Online Boutique Authors. (2025) Online Boutique. [Online]. Available: <https://github.com/GoogleCloudPlatform/microservices-demo>