EasyChair Preprint

№ 12901

iContinuum: an Emulation Toolkit for Intent-Based Computing Across the Edge-to-Cloud Continuum

Negin Akbari, Adel N.Toosi, John Grundy, Hourieh Khalajzadeh, Mohammad Sadegh Aslanpour and Shashikant Ilager

April 8, 2024

iContinuum: An Emulation Toolkit for Intent-Based Computing Across the Edge-to-Cloud Continuum

***Abstract*—The Internet of Things (IoT) has led to a surge in smart devices, generating vast data volumes. Cloud computing offers scalability but does not suffice for many real-time and privacy-sensitive IoT applications, prompting the rise of edge computing. However, many IoT applications require a blend of both edge and cloud resources, creating the need for seamless integration, known as the *"compute continuum"*. Testing applications within this continuum is vital but can be very complex. Simulation and emulation are preferred methods, with emulation providing more accurate representations of real-world environments. In this paper, we introduce *iContinuum*, a novel emulation toolkit facilitating an intent-based platform for edge-to-cloud testing and experimentation. Leveraging Software-Defined Networking (SDN) and containerization, *iContinuum* enables experimentation and performance evaluation while aligning application requirements with actual performance. We present our detailed architecture, implementation, and evaluation of *iContinuum*, showcasing how our proposed toolkit bridges the gap between simulation and real-world deployment within the compute continuum environments. We also present a use case demonstrating the effectiveness of Intent-Based Scheduling. We further validate our emulation toolkit by comparing its performance against a real-world setup.**

I. Introduction

The rise of the Internet of Things (IoT) has led to an explosion of smart devices, generating massive amounts of data. As the number of IoT applications continues to grow, network-connected devices are collecting and exchanging vast amounts of data with each other [1]. Cloud computing provides scalable storage, processing power, and analytical toolkits, enabling seamless integration and processing of this data from interconnected devices. However, cloud computing may not be the optimal choice for supporting certain types of applications, especially those requiring real-time analysis or those with privacy concerns [2]. To meet the demand for low-response time and on-device data processing, a new computing paradigm, such as edge computing, has emerged. Edge computing provides computations and communication resources at the network edge. It promises to surpass the limitations of traditional cloud computing models by bringing computing closer to the data source [3].

While edge computing offers advantages such as low latency, reduced bandwidth usage, and data processing closer to the source, cloud computing provides proven scalability, flexibility, and extensive shared computational resources [4]. However, many emerging IoT-based applications are not suited to the use of either cloud-only or edge-only setups. They often require a combination of edge and cloud resources to meet their needs effectively. This necessity for hybrid deployment models has led to the emergence of the *"compute continuum"* concept [5], which emphasizes the seamless integration of edge and cloud resources. Spanning from the edge to the cloud, this continuum enables applications to seamlessly operate across a diverse spectrum of resources [5].

A thorough testing of applications leveraging the compute continuum is essential before deployment in a production environment. This ensures seamless integration with the unique features of both edge computing and cloud infrastructure, preempting potential issues and optimizing real-world performance. However, testing applications within the compute continuum presents significant challenges. These are due to its complex network setups, resource heterogeneity, widespread distribution of resources, and diverse environmental factors [6]. Additionally, it demands specialized tools and personnel expertise with a deep understanding of the infrastructure and application configurations. Limited access to real-world testing and experimentation environments further complicates these evaluation processes. Experimentation in a real environment is also costly due to the substantial resources and time required for deployment and execution under varying loads. Moreover, the unpredictability of external variables renders experimental results non-repeatable.

Simulation and emulation are used to test and evaluate the performance of large-scale cloud and edge applications. While simulation relies on abstract models to represent software and hardware entities for performance assessment, emulation employs the actual software deployed on testbed hardware to emulate real-world infrastructure configurations during evaluation [7].

Popular simulation toolkits such as iFogSim [8], Cloudsim [9], and EdgeCloudSim [10] have become widespread for testing and developing application management strategies in edge-to-cloud environments. However, simulation using these tools presents numerous challenges, from constraints on authenticity to concerns regarding accuracy. Authenticity challenges stem from the complexity of continuum, which may necessitate oversimplification or overlook certain aspects of real-world environments, potentially leading to unrealistic outcomes or disregarding critical factors [10]. Accuracy concerns arise due to various factors, including model assumptions, data quality, parameter settings, and human errors [11]. Additionally, many cloud and edge simulation toolkits lack detailed network simulation capabilities, failing to adequately capture dynamic interactions and the impact of communications among various components of the system [10].

Emulation, on the other hand, resembles real-world environments [12], providing a more accurate representation of

edge-to-cloud settings. Thus, this approach leads to more dependable testing outcomes, and applications can be more accurately tested for real-world deployment scenarios. Moreover, it provides a practical testing environment compared to simulation and minimizes deployment risks. However, conventional emulation toolkits often focus solely on testing computing or networking functionalities, or on monitoring and testing low-level metrics. In contrast, intent-based approaches enable both joint testing and the assessment of high-level objectives derived from them. With intent-based emulation, the emphasis shifts from merely replicating the behavior of individual components or systems to achieving specific high-level objectives or intents.

In this paper, we propose an emulation toolkit for building intent-based edge-to-cloud computing testing and experimentation platforms called *iContinuum*. *iContinuum* allows developers or end-users to set up an emulated testbed for their applications and perform experimentation and performance evaluation. The proposed emulation toolkit comprises multiple distinct layers with a set of components to build all the layers from infrastructure to applications. We employ an advanced approach to emulate our network environment by leveraging Software-Defined Networking (SDN) that decouples the data plane and control plane in IoT, enabling controllers to effectively regulate network flow while considering resource usage conditions of both cloud and edge servers [13]. Additionally, we utilize containerization and orchestration technologies to efficiently manage the deployment and operation of applications across the emulated infrastructure. This ensures comprehensive consideration of both networking and computing parameters within the edge-to-cloud environment. Moreover, the toolkit supports users in specifying their desired requirements or intents for their applications, such as target response time or energy consumption, and maintains continuous alignment between the desired state of the applications and their actual performance.

Our evaluation demonstrates that *iContinuum* accurately emulates edge-to-cloud environments, capturing application, networking, and computing-level metrics. Validation confirms its ability to closely match real-world performance. Additionally, it proves to be valuable for implementing intent-based methods in these environments.

Section II provides a motivational scenario illustrating the necessity of an emulated testing toolkit for the edge-to-cloud environment. In section III, we present our approach along with the system architecture underlying *iContinuum*, and then in Section IV we describe the design and implementation of *iContinuum* and discuss the key tools utilized in creating this emulation framework. Section V presents a performance evaluation and results analysis of *iContinuum* as well as a use case for Intent-based scheduling. Section VI shows the validation of our emulation toolkit with the real-world environment, and in section VII we discuss about the limitation of our toolkit. Section VIII presents key related work, and we finish with the conclusion in section IX.

II. Motivation

Consider a smart surveillance application in a smart factory environment. Utilizing IoT devices throughout the factory, the system captures real-time data on factory operation. Such an application typically encompasses a range of software components, including video preprocessing, computer vision algorithms (e.g., object detection), alerting and notification systems, user interfaces, and dashboards. These components can be deployed across diverse computing and storage resources, spanning from edge devices to cloud infrastructure which presents many challenges due to resource diversity and environmental conditions. Variability in factory requirements and network conditions, including failures and traffic fluctuations, also adds complexity [14]. Moreover, with some application tasks processing data at the edge and others in the cloud, ensuring seamless integration and optimal performance becomes essential.

This system can leverage IoT devices, such as cameras across the factory floor, to capture real-time data on various aspects of production. These cameras monitor production lines, packages, products, and worker activities, providing valuable insights into the manufacturing processes. At the edge of the network, edge compute nodes process and analyze the data locally, enabling quick decision-making and reducing latency. Meanwhile, cloud computing nodes aggregate and store this data for further analysis, long-term storage, and integration with other enterprise systems.

Comprehensive testing and experimentation are crucial to effectively address the complexities associated with deploying such applications in this environment. Direct deployment onto factory premises carries risks due to uncertainties surrounding application performance and potential impacts on the network and devices. Emulation provides a practical alternative by replicating real-world conditions within a controlled environment. We need to emulate diverse network conditions, resource capacities, software configurations, IoT devices, and operational scenarios. This enables developers to gain insights into application behavior under different circumstances, fine-tuning performance, and ensuring adaptability to the dynamic factory ecosystem. Thus, our aim is to provide developers with a testing toolkit to bridge the gap between controlled testing environments and complex edge-cloud settings, ensuring the successful deployment and operation of such smart surveillance applications and, in general, various edge-cloud setups.

III. Our Approach

Figure 1 illustrates the key architectural layers comprising our proposed emulation toolkit. We mainly emphasize the middleware layer for developing the *iContinuum* tool.

Application layer: In this layer, users define their application structure and configurations, choosing between a chain of services, a Directed Acyclic Graph (DAG), or other application models. Moreover, they can specify their high-level objectives, like optimizing response time or minimizing energy consumption. The application structure and objectives can be defined in the form of *intent* using formats such as YAML

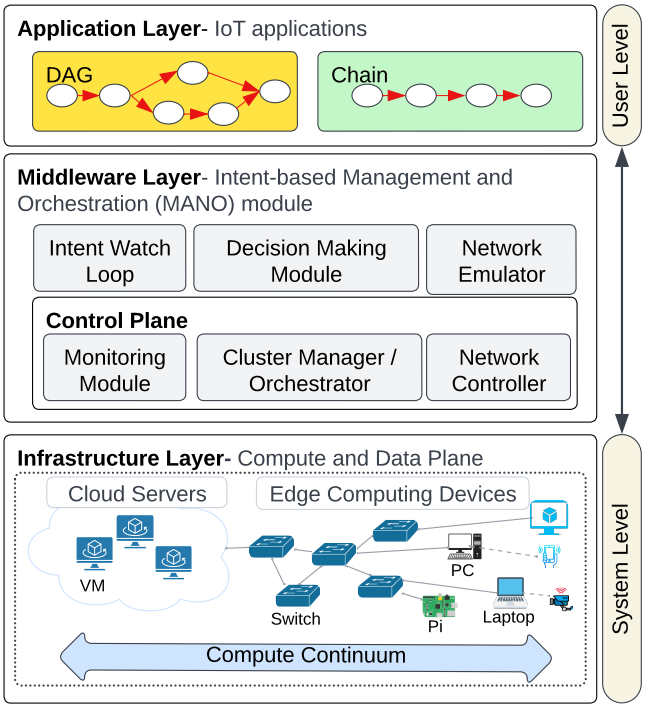


Fig. 1: An overview of the proposed architecture

or JSON. These intents are monitored through a watch loop method integrated within the middleware layer.

Middleware layer: This layer encompasses the primary components responsible for deploying and monitoring the application in the infrastructure layer. It features an *Intent Watch Loop Module*, tasked with detecting any deviations from predefined intents. In the event of an unsatisfied intent, the system initiates appropriate actions through the *Decision-Making Module* to rectify the issue. Positioned at the heart of the middleware layer, is a central control plane integrating both the *Network Controller* for network management and the *Cluster Manager* for orchestrating and managing the computing cluster and its resources. These components work closely with an integrated *Monitoring Module*, responsible for overseeing various computing and networking metrics, including CPU, memory, bandwidth utilization, latency, etc. While it may be relatively straightforward to emulate compute nodes or utilize real-world computing resources within an emulator toolkit, simulating intricate networks across the continuum presents a more formidable challenge. This gap is evident in many simulation tools. Hence, we propose incorporating a *Network Emulator* module into our middleware. This component will be tasked with emulating networks and constructing network topologies within the infrastructure layer.

Infrastructure layer: This layer hosts a diverse array of computing and networking resources, including IoT devices, edge and cloud computing devices, as well as networking devices such as network switches and routers, building the network infrastructure. Some of these devices serve as networking nodes, others as computational nodes, and certain units perform the dual role of network and computation nodes. The network topology within the infrastructure layer is constructed using the network emulator in the middleware layer. The network emulator creates virtual switches or network elements to connect various devices in the infrastructure layer. In this layer, IoT devices like CCTV cameras are attached to edge servers, with the data generated by the IoT device undergoing processing within the compute continuum via microservices or containers built as part of the application. The compute nodes in the continuum can range from conventional computers or physical servers to specialized Single Board Computers (SBCs) like Raspberry Pi at the edge to Virtual Machines (VM) in the cloud.

IV. *iContinuum* Toolkit

This section presents a proof of concept emulation toolkit called *iContinuum* based on our proposed architecture and concepts. We describe key design and technology choices we made and how we realized key elements of the emulation platform.

A. Network Controller

We leverage SDN controller and orchestrator technologies in the control plane to replicate a large-scale compute continuum environment, addressing both networking and computing components. The primary tool we used for network management purposes is the Open Network Operating System (ONOS),[1]. We chose this platform as it excels in SDN for edge computing. ONOS provides centralized control for managing the network with high scalability and flexibility. Its open-source and flexible nature enables support for network programmability. In addition, ONOS features built-in support for "*intent-based networking*", such as *ConnectivityIntent*, simplifying network management by allowing administrators to articulate high-level policies while abstracting away low-level configurations [15]. ONOS can support QoS requirements and has additional recovery protocols to recover from lost crashes due to controller updates [16], and offers east and westbound interfaces for distributed controllers, ensuring network resilience by connecting each data plane element to multiple controllers, with one as master and others as backups. These attributes empower efficient management of resources, seamless integration with diverse network infrastructure, and the essential low-latency communication required for compute continuum applications.

B. Network Emulator

We utilize Mininet[2] to emulate network elements such as swithces as well as relevant network topologies. We chose to use Mininet that provides a powerful and flexible network emulator. A *Traffic Control utility* (TCLink) allows precise specification of bandwidth limits, delay, loss, maximum queue length, and other parameters. It supports the configuration

[1]https://opennetworking.org/onos/
[2]https://mininet.org/

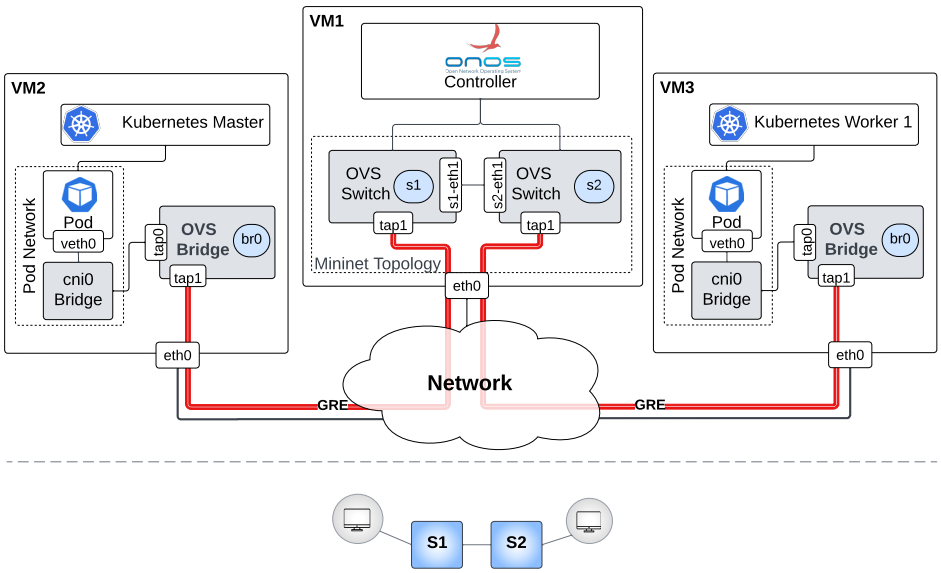


Fig. 2: A Kubernetes cluster with two edge nodes connected via a linear network topology and two switches in Mininet.

of switches such as *Open vSwitch*[3] (OVS) and supports the *OpenFlow* protocol, thus ensuring seamless integration and management of SDN environment. While Mininet effectively emulates switches, its capability to emulate hosts is limited due to its minimalistic shell environment, which prevents the full execution and deployment of applications. Alternatively, tools like *Containernet*,[4] a fork of Mininet, address this issue by supporting containers as hosts within the network topology. However, they lack compatibility with VMs and physical hardware. To address this limitation, we implement a different approach that allows us to seamlessly integrate external computing nodes such as VMs, Raspberry Pis, or physical servers as hosts into the Mininet network topology. These hosts can be used as the Kubernetes cluster masters and workers nodes.

As shown in Figure 2, our novel approach focuses on providing connectivity between the external hosts and the Mininet-created network topology through Generic Routing Encapsulation (GRE) tunneling.[5] To accomplish this, each external host is configured with an OVS bridge featuring two virtual interfaces, *tap0* and *tap1*. The first interface acts as an internal interface (*tap0*), assigning an IP address within the range allocated by Mininet to the network hosts. The second interface (*tap1*), configured as a GRE interface, is linked to a tap port on an OVS switch in the Mininet topology managed by the SDN controllers such as ONOS. Note that OVS bridges on hosts are not managed by the SDN controller and solely connect the external hosts to the switches emulated by Mininet via a GRE tunnel. These external hosts play the role of Kubernetes nodes as explained in the following section. This configuration ensures a bidirectional connection among the external hosts and network switches in the Mininet topology.

C. Container Orchestration

We use containerization as the foundation for application packaging and deployment due to numerous benefits that align well with the requirements of the compute continuum, including resource efficiency, portability, flexibility, isolation and security, as well as orchestration and management capabilities. We integrate Kubernetes as the primary tool for cluster management and orchestration. In our prototype, we use K3s,[6] which is the lightweight version of Kubernetes, suitable for compute continuum use cases. Kubernetes offers a rich set of features supporting container deployment and dynamic resource management [17]. This open-source tool automates the deployment, scaling, and management of containerized applications, facilitating rapid deployment without complex configuration or installation steps [18] which makes it a suitable choice for our system. It seamlessly scales applications to meet changing demands, optimizes resource utilization, ensures high availability, and enables portability across various environments.

D. IoT Devices

To simulate IoT devices, such as sensors, CCTV cameras, and other IoT devices, we utilize Locust[7]. We chose to use Locust as it is a robust load generator capable of generating HTTP or MQTT requests directed toward the application, effectively replicating real-world traffic scenarios. In our use case scenario, Locust operates similarly to CCTV cameras, capturing images within the factory at defined intervals and sending images for processing to the network.

[3]https://www.openvswitch.org/
[4]https://containernet.github.io/
[5]https://www.cloudflare.com/en-gb/learning/network-layer/what-is-gre-tunneling/
[6]https://k3s.io/
[7]https://locust.io/

E. Monitoring Tool

We employ a monitoring tool known as *sFlow-RT*.[8] This provides a real-time monitoring solution designed to gather telemetry data from industry-standard sFlow Agents integrated into network devices or hosts. We utilize the open-source *Host sFlow* agent[9] for performance monitoring of hosts and servers. This agent exports performance metrics such as CPU and memory utilization from both physical and virtual servers using the sFlow protocol. It offers scalable, multi-vendor, and multi-OS performance monitoring capabilities, ensuring minimal impact on the systems under observation. We also utilize standard sFlow agents to monitor Open vSwitches within the Mininet network topology, enabling efficient network monitoring. These agents transmit real-time telemetry data to the *sFlow-RT* central collector, facilitating comprehensive analysis. With all these agents in place, we are able to gather a wide range of metrics, including networking data such as bandwidth and delay and host-level metrics like CPU and memory usage.

F. Monitoring Microservices

Given the diverse microservices (pods) that might be available in the application structure, it is essential to monitor them individually. To achieve this, we leverage sidecar containers[10] with sFlow agents to operate alongside the main application container within the same Pod. They serve to augment or extend the functionality of the primary application container by offering additional services such as logging, monitoring, security, or data synchronization—without necessitating direct modifications to the primary application code. We also utilize the *Prometheus Exporter*,[11] an application integrated into the *sFlow-RT* analytics platform. This exporter seamlessly converts real-time telemetry streams from sFlow agents on hosts and switches into metrics accessible via a REST API and a format compatible with *Prometheus*,[12] enabling *Prometheus* to retrieve and utilize these metrics. We utilize *Prometheus* as a robust time series database and alerting system for persistent storage of real-time monitoring data collected by *sFlow-RT*. Additionally, we leverage Grafana,[13] an open-source analytics and monitoring solution that seamlessly integrates with *Prometheus* for enhanced visualization. This integration allows us to efficiently collect, query, visualize, and alert on metrics data.

G. Platform Automation

We have automated the entire setup of *iContinuum* including configuration, application deployment, and monitoring described above using the Ansible platform,[14] with all associated codes accessible in our GitHub repository.[15] Ansible is an open-source automation tool for configuring and managing computers, software applications, and network devices. This capability empowers users to effortlessly establish an emulation environment for a comprehensive compute continuum, encompassing the entire spectrum of applications, control layers, and infrastructure, all with a single click and a few settings. Ansible is widely used in IT operations, DevOps, and system administration for its simplicity, scalability, and extensibility.

V. Evaluation

In this section, we provide a comprehensive evaluation of *iContinuum* through testing and experimentation of sample applications with mixed cloud and edge components. We begin by describing a sample scenario setup as an example target application and experimental testbed. Following this, we conduct a detailed analysis of our evaluation results to offer a thorough understanding of *iContinuum*'s performance and capabilities, providing valuable insights.

A. Experimental Setup

Our experimental setup consists of five Virtual Machines (VMs) hosted on the Nectar cloud,[16] with detailed configurations provided in Table 1. Four VMs are designated as edge servers within a Kubernetes cluster, comprising one Master node for the control plane and three Worker nodes. Additionally, another VM functions as the SDN controller, equipped with essential monitoring modules including the sFlow-RT collector, Prometheus, and Grafana, alongside Mininet serving as the network emulator.

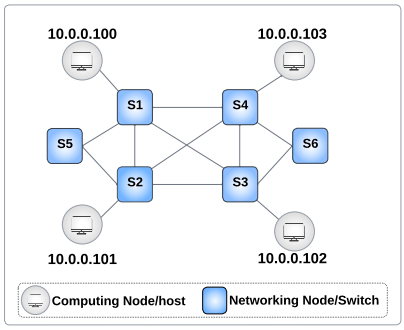
VM	OS	Architecture	RAM	vCPU
Edge Servers	Ubuntu 20.04 LTS	amd64	8GB	4
SDN Controller	Ubuntu 20.04 LTS	amd64	16GB	8

Table 1: Configuration of VMs

Figure 3(a) illustrates the network topology generated by Mininet, as visualized in the ONOS Graphical User Interface (GUI). Each switch-to-switch connection delivers a 200Mbps bandwidth without any additional delay settings, with such negligible network latency that it can be disregarded. The edge nodes connect to the switches via GRE configuration. In Figure 3(b), we showcase the configuration of OVS bridge, named br1, on individual edge servers, along with their respective virtual interfaces (*tap* interfaces). The first interface (*tap0*) functions as the internal interface, featuring IP addresses within the 10.0.0.0/8 subnet range, seamlessly aligned with the Mininet hosts range.

Table 2 shows a detailed breakdown of the edge server IP addresses associated with the *tap0* interface. The second virtual interface (*tap1*) operates as a GRE type, with its remote IP address configured to the ONOS controller's IP address. Also, switches connected to hosts in Figure 3(a) are equipped with a GRE interface, with the remote IP address set to one of the edge nodes. This configuration streamlines centralized cluster management through the ONOS controller.

[8] https://sflow-rt.com/
[9] https://sflow.net/about.php
[10] https://kubernetes.io/docs/concepts/workloads/pods/sidecar-containers/
[11] https://blog.sflow.com/2019/04/prometheus-exporter.html
[12] https://prometheus.io/
[13] https://grafana.com/
[14] https://www.ansible.com/
[15] Removed for anonymous peer review.
[16] https://ardc.edu.au/services/ardc-nectar-research-cloud/



(a) Network Topology

```
#sudo ovs-vsctl show
5d505666-91cd-424b-b4fe-c1a407d4d8da
Bridge br1
Port tap1
Interface tap1
type: gre
options:{remote_ip="ONOS CONTROLLER IP ADDR"}
Port br1
Interface br1
type: internal
Port tap0
Interface tap0
type: internal
ovs_version: "2.13.8"
```

(b) OVS Configuration

Fig. 3: Experimental Setup

Edge Server/ Kubernetes cluster	tap0 IP Address
Master Node	10.0.0.100/8
Worker1 Node	10.0.0.101/8
Worker2 Node	10.0.0.102/8
Worker3 Node	10.0.0.103/8

Table 2: Configuration of edge servers' virtual interface(tap0)

We utilize Locust to send HTTP requests to our application. In Figure 4, we showcase our containerized image processing application built on a chain of services. Docker images for the application can be found in the following Docker Hub link at the footnote.[17] This application comprises four distinct microservices (Pods), each with a unique responsibility. Upon receiving a request, the first pod resizes the image and forwards it to the second microservice, which converts it to a black-and-white version. This transformed image then proceeds to the third microservice, tasked with object detection. Finally, the output is passed to the fourth microservice, which triggers an alarm upon detecting special objects of interest and records their count. This comprehensive setup resembles our factory operations, discussed earlier, by effectively identifying faulty or non-standard objects on the production line. Customized special object definitions further adapt to user requirements or specific production environment needs, ultimately empowering management with greater control over the factory operations.

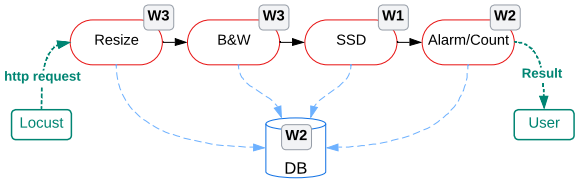


Fig. 4: Image Processing Application

Microservices log the reception and completion times of requests, calculate their processing times, and subsequently transmit this data to a containerized database. The database also logs any failures encountered by the microservices. The microservices are deployed within the Kubernetes cluster and managed by the default scheduler. To prevent overloading, we designate the Master node as non-deployable, acknowledging its critical role as the system's control-plane. Therefore, the microservices are distributed across the worker nodes in the cluster, as depicted in Figure 4 with labels attached to each microservice. In this Figure, 'W1' to 'W3' corresponds to Worker1 to Worker3, respectively.

[17]Removed for anonymous peer review.

B. Results and Analysis

In this section, we will showcase the experiments conducted using *iContinuum*. Given that the proposed platform handles both networking and computing parameters, we undertake diverse experiments at the application level, computing level, and networking level. This approach aims to demonstrate the flexibility and comprehensiveness of *iContinuum*.

Application Level: To stress-test the application, Locust sends synchronous HTTP requests to the application for a duration of 360 seconds with various numbers of concurrent users. Each request carries a JPEG image sized at 499.7kB. Our Locust setup accurately replicates real-world conditions by simulating various concurrency levels with different numbers of threads, resembling multiple CCTV cameras. The spawn rate, set at 1 user per second, determines the rate at which users are generated during the testing phase. We measure the Response Time (RT) of a request, starting from the initial receipt of a request by Microservice 1 to the final completion of processing by Microservice 4. Figure 5 illustrates the CDF of Response Time across various numbers of users, demonstrating the accurate functionality of performance under varying system loads. The smaller variance between the results obtained from 10 and 20 users, in comparison to other concurrency levels, indicates that at a concurrency level of 10, the system is not fully saturated, thereby failing to achieve its maximum throughput.

Table 3 shows the throughput of the application and the number of processed requests for the same experiments, providing insights into performance across varying demand levels. As the results show, *iContinuum* exhibits a reasonable variation in response time and inversely correlated throughput, affirming the reliability of the system under evaluation.

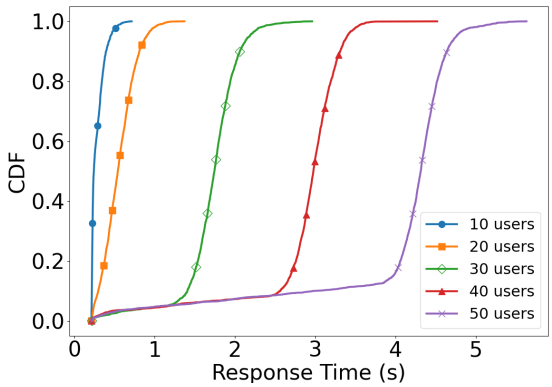


Fig. 5: CDF of response time for different concurrency levels at the workload generator

No. Users	10	20	30	40	50
Throughput	4.27	7.23	7.70	7.75	7.66
No. Processed Requests	1537	2712	2783	2814	2790

Table 3: Throughput and number of processed requests for different concurrency levels at the workload generator

Computing Level: We conduct tests on the system to demonstrate *iContinuum*'s capability to measure computational parameters, such as CPU and memory utilization. The CPU and memory utilization diagrams for each node and pod in the cluster during traffic simulation are depicted in Figure 6. Throughout the 33-minute experiment, data is collected every 15 seconds. Annotations on the top of the plots mark the occurrence of various injected events into the system. The initial phase of the experiment begins before deploying the application. Then, we deploy the application at $e1$, initializing each pod. Following this, at $e2$, we send requests to the application through Locust with 50 concurrent users at a spawn rate of 1 second persisting for a total of 360 seconds up to $e3$. Upon the completion of this traffic, all microservices are terminated at $e4$.

The Figure clearly illustrates that the *Master* node displays higher CPU and memory utilization, serving as the control plane responsible for task scheduling and application deployment within the cluster. Additionally, *Worker3*, which hosts both *Pod1* and *Pod2*, emerges as the second node with the highest of resource utilization, with *Pod1* acting as the initial entry point for the application. Moving forward, *Worker1* houses *Pod3*, which requires substantial processing time for object detection. In contrast, *Worker2* demonstrates lower CPU and memory usage, as it hosts *Pod4* and the database (DB). These microservices demand less processing time, with *Pod4* responsible for generating alarms upon detecting special objects from the previous Pod and then counting them, while the DB focuses primarily on data storage.

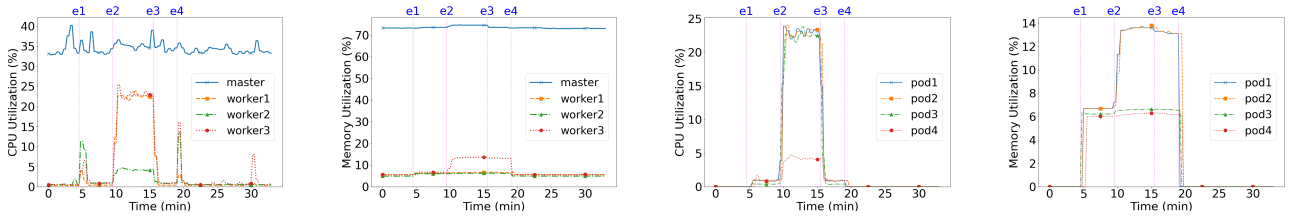
Networking Level: Utilizing Locust for a duration of 360 seconds, we simulate HTTP traffic through the application with 50 users and a spawn rate of 1 user per second. The goal was to show how varying bandwidth and delay settings impact the application's performance. Figure 7(a) presents the application's response time under various bandwidth configurations. Initially, we measured the application response time with all links set to a bandwidth of 50Mbps, configured using the Mininet TCLink utility. Subsequently, we repeated the experiment with bandwidth reduced to 40, 30, 20 and 10Mbps. Figure 7(b) shows the throughput of the application during the experiment with different bandwidth setups. As expected, the higher the bandwidth the lower the response time.

Figure 7(c) shows how different delays can affect the application RT with the same configuration of Locust. This experiment is conducted with a fixed bandwidth set to 50Mbps. We changed the delay of all links by the Traffic Control (tc)[18] utility in Linux to 5ms, 10ms, 15ms, and 20ms, respectively. Figure 7(d) represents the throughput of the application with different delays configured on the links. These experiments underscore the adaptability and flexibility of *iContinuum* across varied networking conditions. Results demonstrate *iContinuum*'s capability to dynamically adjust network parameters for more accurate emulation of real-world conditions.

C. A Use Case: Intent-based Scheduling

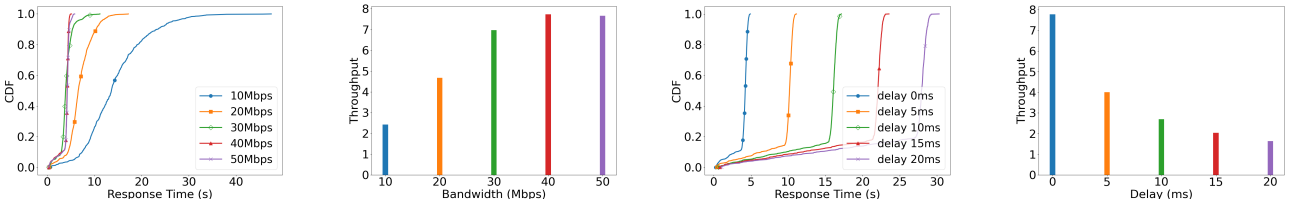
In this section, we present an intent-based scheduling approach as a use case for *iContinuum* emulation as part of our evaluation. The aim is to consistently maintain application RT below a predefined threshold. During a 360-second experiment, Locust simulates traffic with 10 users at a spawn rate of 1 user per second, using the same image size as before. We use the network topology in Figure 3(a). We also set 5ms delay and 50Mbps bandwidth on all switch interconnected links. Each microservice is configured with a limit of 0.5 CPU core and 512MiB memory. As shown in Figure 8, we considered an average target RT threshold below 3 seconds as our intent, denoted as 'Target RT' on the graph. We introduced several events into the system to induce intent non-conformance. At event $e1$, a downtime incident occurred on the direct link between switches *S2* and *S4*. This issue was promptly resolved by rerouting traffic through alternative paths: *S4-S3-S2* in one direction and *S2-S1-S4* in the other, thereby preventing any violation of the intent requirement for response time below 3 seconds. However, this rerouting led to a slight increase in application response time, which was unavoidable. We used *'Reactive Forwarding'* on ONOS controller to promptly redirects traffic via an alternative route. Subsequently, at event $e2$, we intentionally induce syntactic congestion on the newly selected link (*S2-S3*) by injecting *iperf* traffic, resulting in a breach of the response time requirement. However, the intent is promptly restored to the desired level by establishing new flows through the ONOS Controller's REST API. Then, at event $e3$, we increase the concurrent users in Locust from 10 to 20, with a spawn rate of 1 user per second, deliberately triggering another violation of the intent. This is responded by scaling

[18]https://www.linux.com/training-tutorials/tc-show-manipulate-traffic-control-settings/



(a) CPU Utilization of Nodes (b) Memory Utilization of Nodes (c) CPU Utilization of Pods (d) Memory Utilization of Pods

Fig. 6: CPU and Memory utilization for Nodes and Pods (e1: Application Deployment- e2: Traffic Generation- e3: Traffic Completion- e4: Application Termination)



(a) Bandwidth Impact on Application Response Time (b) Bandwidth Impact on Application Throughput (c) Delay Impact on Application Response Time (d) Delay Impact on Application Throughput

Fig. 7: Impact of bandwidth and delay on application performance

up resources, leveraging Kubernetes' scaling mechanism to boost pods for microservice3, which needs more resources to process, from 1 to 8 replicas, effectively reducing the average response times to meet the desired thresholds. The figure clearly illustrates how intent-based scheduling algorithms can be effectively emulated by *iContinuum*.

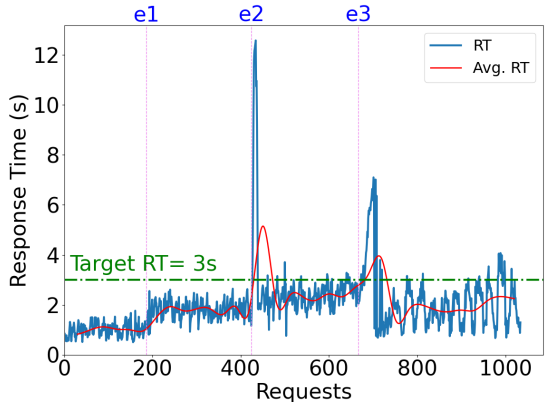


Fig. 8: Application Response Time over the time when different events are induced. Response Time (RT)- Average Response Time (Avg. RT)

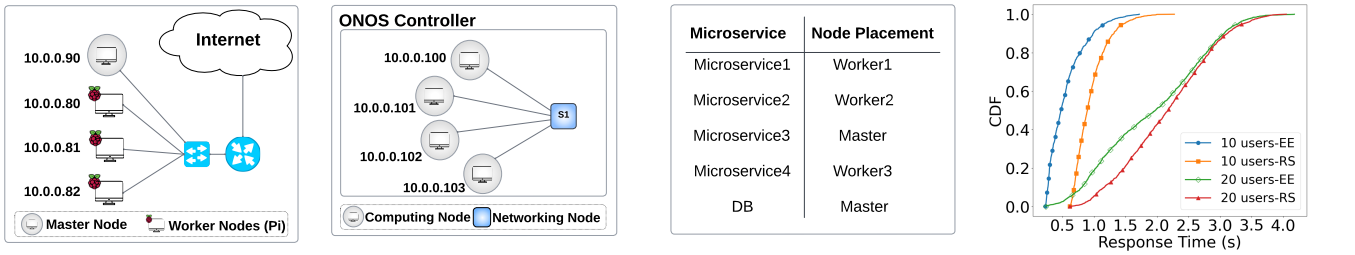
VI. Validation

In this section, we validate our proposed emulation tool by comparing it against a real-world setup. This assessment goes beyond emulation, ensuring that *iContinuum* can closely replicate practical settings found in real-world environments. To achieve this, we configured a Kubernetes cluster over multiple edge devices comprising a Master node on an Intel NUC 8 and three Worker nodes on Raspberry Pis Model 3B, as detailed in Table 4, acting as edge devices. These devices were interconnected through a NetGear switch (Model: ProSafe 16 port Gigabit switch GS116 V2). The network topology of this setup is illustrated in Figure 9(a), reflecting a simple real-world setup. We replicated a similar topology in our proposed emulator, as depicted in Figure 9(b), facilitating a direct comparison between the two environments.

Edge Nodes	Role	OS	Arch	RAM	CPU Cores
Intel NUC	Master	Ubuntu 22.04 LTS	amd64	8GB	8
Pi 3 Model B	Worker1	Pi (Legacy,64-Bit)	arm64	1GB	4
Pi 3 Model B	Worker2	Pi (Legacy,64-Bit)	arm64	1GB	4
Pi 3 Model B	Worker3	Pi (Legacy,64-Bit)	arm64	1GB	4

Table 4: Configuration-Read-world devices

Figure 9(c) illustrates the placement of microservices across various nodes in both the real-world setup and the emulated environment. Microservice 3 and the database are hosted on the Master node, while the remaining microservices are deployed across the Worker nodes. Figure 9(d) presents a comparison of application response times between the real-world setup and the emulated environment, under varying loads generated by Locust. The test scenarios involved 10 users with a spawn rate of 1 user per second, followed by 20 users with the same spawn rate, executed for 360 seconds, with the image size remaining as the same we used in the previous section. Bandwidth and delays on the links in the emulated environment were configured to match those of the real-world setup. However, we introduced CPU limits on the pods in the emulated environment to simulate the lower computational capacity of the Raspberry Pis. Analysis of the



(a) Real-world Setup

(b) The Emulated Environment in *iContinuum*

(c) Placement of Pods

(d) CDF of Response Time for Emulated Environment (EE) and Real-world Setup (RS) with different number of users

Fig. 9: Comparison between the real-world setup and emulated environment

CDF of response times, as depicted in Figure 9(d), shows that both the real-world setup and the emulated environments yield nearly identical results, thus validating the effectiveness of our proposed emulation tool.

VII. Discussion and Limitations

Our evaluation demonstrates that *iContinuum* adeptly emulates intricate edge-to-cloud environments for application deployment. It offers thorough metrics and detailed information essential for testing and experimenting with these applications. Furthermore, its containerization simplifies the packaging and deployment of applications, including those with multiple microservices and dependencies, such as chain and graph structures. Additionally, it provides all the necessary features for implementing advanced intent-based computing and networking within the edge-to-cloud continuum.

Our proposed emulation tool faces several limitations. Its scalability is constrained by the resource limitation on the host, we emulated the network topology using Mininet and other centralized components such as Kubernetes Master node and ONOS controller which are central to the emulation. Additionally, our tool struggles to simulate wireless networks accurately. Our future plan is to incorporate tools such as Mininet-Wifi [19] to address this limitation. While a simplified version of mobility can be incorporated into our toolkit by dynamically connecting the host to various switches during the emulation, the current version does not account for the mobility of computing nodes. These limitations highlight the need for ongoing research to enhance the scalability, mobility, and wireless network emulation of our tool for more accurate edge-to-cloud testing scenarios.

VIII. Related Work

A. Compute Continuum

Edge computing is a relatively new distributed computing paradigm that has gained significant attention in recent years. It is particularly beneficial for applications demanding real-time data processing [20], such as IoT devices, self-driving cars, and remote medical diagnostics. This approach enhances scalability and reduces the strain on cloud-based systems [21], holding immense potential for industries ranging from healthcare and manufacturing to autonomous vehicles and smart cities. While edge computing focuses on processing data as close to its source as feasible, thereby reducing network latency, minimizing bandwidth usage, and improving application performance [2], cloud computing emphasizes remote data storage and processing through centralized servers, providing scalability and accessibility benefits [22]. Despite the significant potential of edge computing in reducing the burden on core networks, its primary limitation lies in its restricted computational and communication capacities compared to cloud computing [23]. These applications rely on both cloud resources and local IoT devices to capture and process data. They have diverse requirements such as low-latency analytics, data privacy, time and location sensitivity, and simultaneous access to distributed sensor arrays. Additionally, they necessitate access to remote, localized, heterogeneous computational resources, as well as seamless multi-cloud computational capabilities on demand [24]. This scenario has led to the development of a new computational paradigm known as the *compute continuum* [5].

The compute continuum is characterized by its diversity, featuring a broad array of capabilities, locations, programming models, and constraints. In this environment, resources encompass computing power, storage capacity, networking capabilities, and specialized components like GPUs, AI, FPGAs, along with services tailored specifically for edge and cloud environments [25]. Therefore, developers and designers need to skillfully utilize these resources throughout the network hierarchy, ensuring smooth migration of applications across different networks and service providers. [25] and [26] discuss the main important features of an edge environment emulator or simulator. These features encompass 1) detailed deployment models for edge networks, incorporating diverse tiers of edge and cloud nodes; 2) dynamic modeling of edge network behavior; 3) mobility of terminals and edge devices; 4) real-time measurement, visualization, and post-analysis of metrics; 5) modeling of failures and reachability; and 6) scalability and extendibility.

B. Emulation and Simulation Tools for Compute Continuum

[25] provides an overview of available emulation and simulation tools, organizing them based on their functionalities. Many solutions focus solely on the IoT sector and may

not readily apply to other areas; We believe that emulation tools should be adaptable across various application domains. iFogSim [8], specializing in fog node placement algorithms within the IoT domain, is an adaptation of CloudSim [9]. However, there is no mobility support in iFogsim [27]. EdgeCloudSim, another adaptation of CloudSim [9], offers modeling capabilities such as network configuration, mobility, and traffic patterns. However, accurately simulating the network presents significant challenges. While Yet Another Fog Simulator (YAFS) [28] does support edge topology simulation, it primarily targets the IoT domain, and its execution time is notably high, consequently leading to increased response times [29].

EmuFog [30] which is an emulation framework for Fog environments, facilitates the simulation of Docker-based applications. It also offers customizable features, allowing users to specify the placement of Fog computing nodes and define their capabilities and workload expectations. However, mobility support is not available in EmuFog [25]. While EmuFog utilizes MaxiNet [31] to track local node events such as CPU and memory consumption, it lacks a universal interface for monitoring global metrics such as response time [27]. Fogify [26] provides a comprehensive fog emulation framework featuring fog topology modeling, dynamic network behavior simulation, KPI monitoring, and seamless integration with edge application workloads. However, its scope is limited to the IoT/Fog domain and does not incorporate integration with external edge resources and compute continuum [25].

The existing literature lacks an emulation framework tailored specifically for edge and cloud environments, allowing users to define their requirements while considering both networking and computing capabilities. Thus, we proposed *iContinuum* to address the needs of the edge-to-cloud environment emulation. The proposed framework employs two widely adopted technologies suitable for the compute continuum environment: SDN and Container Orchestration. With this framework, users have the flexibility to define their requirements, ensuring that the system meets their specific needs.

C. Using SDN in Edge-to-Cloud Continuum

The increasing need for next-generation capabilities, as processing moves to the edge-to-cloud environment, requires effective solutions. In recent years, Software-Defined Networking (SDN) has evolved to offer comprehensive network programmability. SDN separates the control plane and data plane functions, optimizing resource use and traffic flow for reduced latency and improved performance [32]. Its programmability tailors network policies to diverse applications in edge-to-cloud environment. The centralized control plane in SDN simplifies network management and scalability, enabling administrators to focus on strategic tasks [33].

D. Using Container Orchestration in Edge-to-Cloud Continuum

In the realm of cloud and edge computing, where devices and nodes constantly change, manual service management is challenging. Automated orchestration tools like Kubernetes [34], KubeEdge,[19] or ioFog[20] provide a solution. Containerization technology allows developers to package their applications and dependencies into lightweight and portable containers, making it easier to deploy and run applications across different environments. Container Orchestration offers unified management, dynamic scaling, and simplified deployment of microservices-based architectures across diverse edge environments [35]. Additionally, these platforms enhance security, reliability, and performance through built-in monitoring, failover mechanisms, and support for various networking and storage options.

E. Using IBN in Edge-to-Cloud Continuum

While SDN provides flexible network control and intelligence, the discrepancy between business needs and network capabilities requires the underlying network to consistently adapt, protect, and inform across all service-oriented areas. Intent-based networking (IBN) [36] has emerged as a promising solution for addressing the aforementioned gap by capturing business intent and subsequently activating and ensuring it throughout the network [37]. Recent studies are focusing on extending this innovative framework to the computing domain, including edge and edge-to-cloud domains [38], [39], [40], [41]. *iContinnum* provides seamless support for intents to serve as high-level abstractions, empowering systems administrators and developers to articulate their desired outcomes without necessitating explicit instructions on how to achieve them.

IX. Conclusions and Future Work

The rise of edge-to-cloud environment highlights the critical importance of pre-deployment testing for ensuring seamless integration of applications across the edge and cloud infrastructures. While emulation and simulation are commonly employed for testing purposes, emulation stands out for its ability to provide results closely mirroring real-world conditions. In this paper, we proposed *iContinuum*—an emulation toolkit tailored for the edge-to-cloud continuum. Leveraging SDN controller and container orchestrator, *iContinuum* offers a flexible solution to support diverse networking and computation needs, including accommodating user-defined system intents. We provided a comprehensive overview of *iContinuum*'s architectural framework and components, along with evaluations demonstrating its efficacy in emulating the varied requirements of IoT applications in edge-to-cloud environments. We validated our tool against a real-world setup involving multiple edge devices and diverse network settings. Additionally, we presented a use case for Intent-based scheduling using *iContinuum*. In our future work, we aim to enhance our emulation tool by integrating mobility and wireless connectivity support. Furthermore, we plan to propose novel intent-based scheduling methods to optimize resource allocation based on high-level objectives using *iContinuum*.

[19]https://kubeedge.io/

[20]https://iofog.org/

REFERENCES

[1] A. A. Abba Ari, O. K. Ngangmo, C. Titouna, O. Thiare, A. Mohamadou, and A. M. Gueroui, "Enabling privacy and security in cloud of things: Architecture, applications, security & privacy challenges," *Applied Computing and Informatics*, vol. 20, no. 1/2, pp. 119–141, 2024.

[2] G. Agapito and M. Cannataro, "An overview on the challenges and limitations using cloud computing in healthcare corporations," *Big Data and Cognitive Computing*, vol. 7, no. 2, p. 68, 2023.

[3] J. Sakhdari, B. Zolfaghari, S. Izadpanah, S. Mahdizadeh Zargar, M. Rahati Quchani, M. Shadi, S. Abrishami, and A. Rasoolzadegan, "Edge computing: A systematic mapping study," *Concurrency and Computation: Practice and Experience*, vol. 35, no. 22, p. e7741, 2023.

[4] M. Sajid and Z. Raza, "Cloud computing: Issues & challenges," in *International conference on cloud, big data and trust*, vol. 20, no. 13. sn, 2013, pp. 13–15.

[5] G. R. Russo, V. Cardellini, and F. L. Presti, "Serverless functions in the cloud-edge continuum: Challenges and opportunities," in *2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2023, pp. 321–328.

[6] V. Casamayor Pujol, A. Morichetta, I. Murturi, P. Kumar Donta, and S. Dustdar, "Fundamental research challenges for distributed computing continuum systems," *Information*, vol. 14, no. 3, p. 198, 2023.

[7] J. Gustedt, E. Jeannot, and M. Quinson, "Experimental methodologies for large-scale systems: a survey," *Parallel Processing Letters*, vol. 19, no. 03, pp. 399–418, 2009.

[8] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "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, pp. 1275–1296, 2017.

[9] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "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, pp. 23–50, 2011.

[10] C. Sonmez, A. Ozgovde, and C. Ersoy, "Edgecloudsim: An environment for performance evaluation of edge computing systems," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 11, p. e3493, 2018.

[11] S. Svorobej, P. Takako Endo, M. Bendechache, C. Filelis-Papadopoulos, K. M. Giannoutakis, G. A. Gravvanis, D. Tzovaras, J. Byrne, and T. Lynn, "Simulating fog and edge computing scenarios: An overview and research challenges," *Future Internet*, vol. 11, no. 3, p. 55, 2019.

[12] W. Kiess and M. Mauve, "A survey on real-world implementations of mobile ad-hoc networks," *Ad Hoc Networks*, vol. 5, no. 3, pp. 324–339, 2007.

[13] P. K. Sharma, S. Rathore, Y.-S. Jeong, and J. H. Park, "Softedgenet: Sdn based energy-efficient distributed network architecture for edge computing," *IEEE Communications magazine*, vol. 56, no. 12, pp. 104–111, 2018.

[14] S. Trinks and C. Felden, "Edge computing architecture to support real time analytic applications: A state-of-the-art within the application area of smart factory and industry 4.0," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 2930–2939.

[15] S. Jaberi, "Using assl as a method for intent expression to enact autonomic networking," Ph.D. dissertation, Concordia University, 2023.

[16] P. A. D. S. N. Wijesekara and S. Gunawardena, "A comprehensive survey on knowledge-defined networking," in *Telecom*, vol. 4, no. 3. MDPI, 2023, pp. 477–596.

[17] Q.-M. Nguyen, L.-A. Phan, and T. Kim, "Load-balancing of kubernetes-based edge computing infrastructure using resource adaptive proxy," *Sensors*, vol. 22, no. 8, p. 2869, 2022.

[18] S. Pettersson, "Predictive scaling for microservices-based systems," p. 52, 2023.

[19] R. R. Fontes, S. Afzal, S. H. Brito, M. A. Santos, and C. E. Rothenberg, "Mininet-wifi: Emulating software-defined wireless networks," in *2015 11th International Conference on Network and Service Management (CNSM)*. IEEE, 2015, pp. 384–389.

[20] Q. V. Khanh, V.-H. Nguyen, Q. N. Minh, A. D. Van, N. Le Anh, and A. Chehri, "An efficient edge computing management mechanism for sustainable smart cities," *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100867, 2023.

[21] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[22] S. Murugesan and I. Bojanova, "Cloud computing: an overview," *Encyclopedia of cloud computing*, pp. 1–14, 2016.

[23] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 5031–5044, 2019.

[24] A. Ullah, T. Kiss, J. Kovács, F. Tusa, J. Deslauriers, H. Dagdeviren, R. Arjun, and H. Hamzeh, "Orchestration in the cloud-to-things compute continuum: taxonomy, survey and future directions," *Journal of Cloud Computing*, vol. 12, no. 1, p. 135, 2023.

[25] R. Gazda, M. Roy, J. Blakley, A. Sakr, and R. Schuster, "Towards open and cross domain edge emulation–the advantedge platform," in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2021, pp. 339–344.

[26] M. Symeonides, Z. Georgiou, D. Trihinas, G. Pallis, and M. D. Dikaiakos, "Fogify: A fog computing emulation framework," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2020, pp. 42–54.

[27] D. P. Abreu, K. Velasquez, M. Curado, and E. Monteiro, "A comparative analysis of simulators for the cloud to fog continuum," *Simulation Modelling Practice and Theory*, vol. 101, p. 102029, 2020.

[28] I. Lera, C. Guerrero, and C. Juiz, "Yafs: A simulator for iot scenarios in fog computing," *IEEE Access*, vol. 7, pp. 91 745–91 758, 2019.

[29] E. Del-Pozo-Puñal, F. García-Carballeira, and D. Camarmas-Alonso, "A scalable simulator for cloud, fog and edge computing platforms with mobility support," *Future Generation Computer Systems*, vol. 144, pp. 117–130, 2023.

[30] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran, "Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures," in *2017 IEEE Fog World Congress (FWC)*. IEEE, 2017, pp. 1–6.

[31] P. Wette, M. Dräxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl, "Maxinet: Distributed emulation of software-defined networks," in *2014 IFIP Networking Conference*. IEEE, 2014, pp. 1–9.

[32] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "Sdn security: A survey," in *2013 IEEE SDN For Future Networks and Services (SDN4FNS)*. IEEE, 2013, pp. 1–7.

[33] W. Rafique, L. Qi, I. Yaqoob, M. Imran, R. U. Rasool, and W. Dou, "Complementing iot services through software defined networking and edge computing: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1761–1804, 2020.

[34] B. Thurgood and R. G. Lennon, "Cloud computing with kubernetes cluster elastic scaling," in *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems*, 2019, pp. 1–7.

[35] N. Gupta, K. Anantharaj, and K. Subramani, "Containerized architecture for edge computing in smart home: A consistent architecture for model deployment," in *2020 International Conference on Computer Communication and Informatics (ICCCI)*. IEEE, 2020, pp. 1–8.

[36] A. Clemm, L. Ciavaglia, L. Granville, and J. Tantsura, "Intent-based networking-concepts and definitions, 2021," *URL: https://tools. ietf. org/html/draft-irtf-nmrgibn-concepts-definitions-02, last accessed*, vol. 25, 2020.

[37] A. Singh, G. S. Aujla, and R. S. Bali, "Intent-based network for data dissemination in software-defined vehicular edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 8, pp. 5310–5318, 2020.

[38] T. He, A. N. Toosi, N. Akbari, M. T. Islam, and M. A. Cheema, "An intent-based framework for vehicular edge computing," in *2023 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2023, pp. 121–130.

[39] N. Filinis, I. Tzanettis, D. Spatharakis, E. Fotopoulou, I. Dimolitsas, A. Zafeiropoulos, C. Vassilakis, and S. Papavassiliou, "Intent-driven orchestration of serverless applications in the computing continuum," *Future Generation Computer Systems*, vol. 154, pp. 72–86, 2024.

[40] A. Morichetta, N. Spring, P. Raith, and S. Dustdar, "Intent-based management for the distributed computing continuum," in *2023 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2023, pp. 239–249.

[41] A. Zafeiropoulos, E. Fotopoulou, C. Vassilakis, I. Tzanettis, C. Lombardo, A. Carrega, and R. Bruschi, "Intent-driven distributed applications management over compute and network resources in the computing continuum," in *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*. IEEE, 2023, pp. 429–436.