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
**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 $i$Continuum. $i$Continuum 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 $i$Continuum 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 $i$Continuum, and then in Section IV we describe the design and implementation of $i$Continuum and discuss the key tools utilized in creating this emulation framework. Section V presents a performance evaluation and results analysis of $i$Continuum 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 the limitation of our toolkit. Section VIII presents key related work, and we finish with the conclusion in section IX.
Fig. 1: An overview of the proposed architecture

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 network-
as Open vSwitch\(^3\) (OVS) to support the OpenFlow protocol, thus ensuring seamless integration and management within our 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 which allows us to seamlessly integrate external computing nodes such as VMs, 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 connects the external hosts to the switches emulated by Mininet via a GRE tunnels. 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 towards 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/


\(^{6}\)https://k3s.io/

\(^{7}\)https://locust.io/
E. Monitoring Tool

We employ a monitoring tool known as sFlow-RT. 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 sFlow agent\(^8\) 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, as well as 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.

<table>
<thead>
<tr>
<th>VM</th>
<th>OS</th>
<th>Architecture</th>
<th>RAM</th>
<th>vCPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge Servers</td>
<td>Ubuntu 20.04 LTS</td>
<td>amd64</td>
<td>8GB</td>
<td>4</td>
</tr>
<tr>
<td>SDN Controller</td>
<td>Ubuntu 20.04 LTS</td>
<td>amd64</td>
<td>16GB</td>
<td>8</td>
</tr>
</tbody>
</table>

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 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/
\(^12\)https://prometheus.io/
\(^13\)https://grafana.com/
\(^14\)https://www.ansible.com/
\(^15\)Removed for anonymous peer review.

\(^16\)https://arcs.edu.au/services/ardc-nectar-research-cloud/
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.

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.

---

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

<table>
<thead>
<tr>
<th>Edge Server/ Kubernetes cluster</th>
<th>tap0 IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Node</td>
<td>10.0.0.100/8</td>
</tr>
<tr>
<td>Worker1 Node</td>
<td>10.0.0.101/8</td>
</tr>
<tr>
<td>Worker2 Node</td>
<td>10.0.0.102/8</td>
</tr>
<tr>
<td>Worker3 Node</td>
<td>10.0.0.103/8</td>
</tr>
</tbody>
</table>

---

17 Removed for anonymous peer review.
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

---

18https://www.linux.com/training-tutorials/tc-show-manipulate-traffic-control-settings/
by scaling 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 an 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)](image)

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.

<table>
<thead>
<tr>
<th>Edge Nodes</th>
<th>Role</th>
<th>OS</th>
<th>Arch</th>
<th>RAM</th>
<th>CPU Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel NUC</td>
<td>Master</td>
<td>Ubuntu 22.04 LTS</td>
<td>arm64</td>
<td>8GB</td>
<td>8</td>
</tr>
<tr>
<td>Pi 3 Model B</td>
<td>Worker1</td>
<td>Pi (Legacy,64-Bit)</td>
<td>arm64</td>
<td>1GB</td>
<td>4</td>
</tr>
<tr>
<td>Pi 3 Model B</td>
<td>Worker2</td>
<td>Pi (Legacy,64-Bit)</td>
<td>arm64</td>
<td>1GB</td>
<td>4</td>
</tr>
<tr>
<td>Pi 3 Model B</td>
<td>Worker3</td>
<td>Pi (Legacy,64-Bit)</td>
<td>arm64</td>
<td>1GB</td>
<td>4</td>
</tr>
</tbody>
</table>

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
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 Kubernetes Master node and ONOS controller which are central to the emulation. Additionally, our tool struggles to accurately simulate wireless networks. 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 reduce 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 environment, 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 environment

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 environment

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, or iOFog 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 environment

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]. iContinuum 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.

19https://kubeedge.io/
20https://iofog.org/
REFERENCES


