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

Negin Akbari*, Adel N. Toosi*, John Grundy*, Hourieh Khalajzadeh†,
Mohammad S. Aslanpour*, Shashikant Ilager‡
*Department of Software Systems and Cybersecurity, Monash University, Australia,
{negin.akbari, adel.n.toosi, john.grundy, mohammad.aslanpour}@monash.edu
†School of Info Technology, Deakin University, Australia, hkhalajzadeh@deakin.edu.au
‡Institute of Information Systems Engineering, TU Wien, Austria, shashikant.ilager@tuwien.ac.at

***Abstract*—The Internet of Things (IoT) has led to a surge in smart devices, generating vast volumes of data. Cloud computing offers scalability but does not suffice for many real-time and privacy-sensitive IoT applications. This limitation has prompted a blend of both edge and cloud resources, creating the need for seamless integration, known as the *"compute continuum"*. Testing applications and resource management techniques 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 compute continuum environments, and further demonstrate the effectiveness of Intent-Based Scheduling through a specific use case.**

***Index Terms*—Compute Continuum, Emulation, IBN, Intents, SDN, Container Orchestration, Kubernetes.**

## I. Introduction

The rise of the Internet of Things (IoT) has led to an explosion of smart devices generating massive amounts of data, necessitating scalable storage and processing solutions like cloud computing. However, cloud computing is not always optimal for real-time or privacy-sensitive applications, prompting the emergence of "compute continuum," [1] which integrates edge and cloud resources for seamless operation across a diverse spectrum of needs.

A thorough testing of applications leveraging the compute continuum is essential before deployment in a production environment. 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 [2]. Additionally, it demands specialized tools and personnel expertise with a deep understanding of the infrastructure and application configurations. 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 configurations during evaluation [3]. Popular simulation toolkits such as *iFogSim* [4], *Cloudsim* [5], and *EdgeCloudSim* [6] 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. 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 [6]. Emulation, on the other hand, resembles real-world environments [7], providing a more accurate representation of edge-to-cloud settings.

Conventional emulation toolkits often limit their scope to testing specific computing or networking functionalities, or they concentrate on monitoring and assessing low-level metrics. However, intent-based emulation marks a departure from this approach by shifting the focus from mere replication of individual components or systems to the attainment of specific high-level objectives or intents. This shift is particularly crucial in the context of the compute continuum, characterized by the dynamic and heterogeneous nature of edge-to-cloud environments. By prioritizing the intent or desired outcome over low-level configurations, such approaches simplify management, enhance automation, and bolster overall system agility.

In this paper, we propose *iContinuum*, an emulation toolkit designed for constructing intent-based edge-to-cloud computing testing and experimentation platforms. The toolkit consists of multiple layers, each comprising a set of components that span from infrastructure to applications. Leveraging Software-Defined Networking (SDN), we decouple the data plane from the control plane, allowing the network controller to efficiently regulate network flows [8]. Additionally, containerization and orchestration technologies are employed to effectively manage the deployment and operation of applications across the compute continuum. Furthermore, *iContinuum* enables users to specify their desired requirements or high-level intents for their applications, such as target response time, privacy requirements, and energy consumption goals, while ensuring continuous alignment between the desired application state and its actual performance. Our evaluation demonstrates that *iContinuum* accurately emulates edge-to-cloud environments, capturing application, networking, and computing-level metrics. Additionally, it proves to be valuable for implementing intent-based methods in these environments.

## II. Motivation

Consider a smart surveillance application in a smart factory environment. This system uses IoT devices such as CCTV cameras to capture real-time data on factory operations. Such an application encompasses a range of software components, including video preprocessing, computer vision algorithms (e.g., object detection), alerting and notification systems, and user interfaces. These components can be deployed across edge devices and cloud infrastructure, facing challenges due to resource diversity and environmental conditions. Additionally, factory requirements and operations may change over time, introducing an extra layer of complexity. 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. 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 *Service Function Chains* (SFC), 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 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

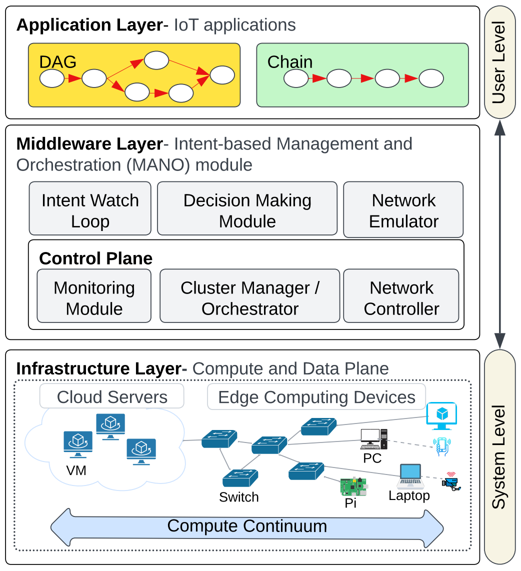


Fig. 1: An overview of the proposed architecture

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 easier to emulate compute nodes or utilize real-world computing resources within an emulator toolkit, simulating network 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 to address this gap.

**Infrastructure layer:** This layer hosts a diverse array of computing resources, including IoT devices, edge and cloud, as well as networking devices such as network switches and routers. 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, and Virtual Machines (VM) to specialized Single Board Computers (SBCs) like Raspberry Pis.

## 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 rely on Open Network Operating System (ONOS)[1] for network management for the SDN controller. ONOS offers centralized control with scalability, flexibility, and support for network programmability. It features built-in support for "*intent-based networking*", such as *ConnectivityIntent*, simplifying network management by abstracting low-level configurations [9]. ONOS also provides east and westbound interfaces for distributed controllers, ensuring network resilience and enabling efficient resource management and low-latency communication for compute continuum applications.

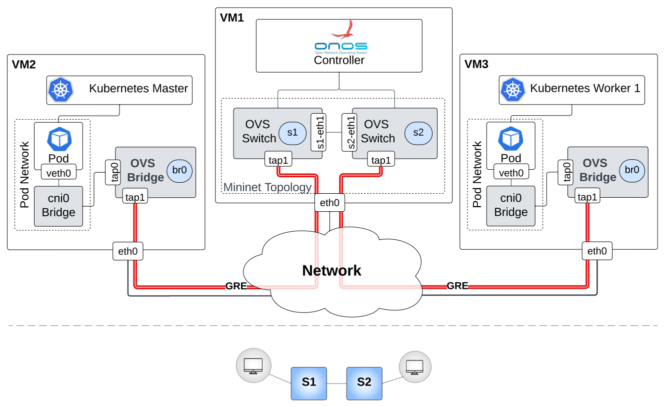


Fig. 2: A Kubernetes cluster with two edge nodes connected via a linear network topology and two switches in Mininet.

### B. Network Emulator

We use Mininet[2] for emulating network elements and topologies, leveraging its *Traffic Control Utility* (TCLink) for precise parameter settings, including bandwidth limits and delay. With support for *Open vSwitch*[3] (OVS) and the *OpenFlow* protocol, Mininet ensures seamless integration of SDN environments. To address host emulation limitations of Mininet, we seamlessly integrate external computing nodes like VMs, Raspberry Pis, or physical servers into Mininet, enabling their use as Kubernetes cluster nodes.

As shown in Figure 2, our novel approach to connect external hosts to Mininet topology focuses on providing connectivity between the external hosts and the Mininet-created network topology through Generic Routing Encapsulation (GRE) tunneling.[4] To accomplish this, each external host is configured with an OVS bridge featuring two virtual interfaces, *tap0* and *tap1*, in which *tap0* acts as an internal interface, assigning an IP address within the range allocated by Mininet to the network hosts, and *tap1*, configured as a GRE interface, linked to a tap port on an OVS switch in the Mininet topology managed by the SDN controllers such as ONOS. These external hosts play the role of Kubernetes nodes as explained in the following section.

### C. Container Orchestration

We leverage containerization for efficient application packaging and deployment, aligning with the needs of the compute continuum. Using Kubernetes for cluster management and orchestration, particularly the lightweight K3s[5] version, we ensure flexibility, security, and seamless scalability. Kubernetes automates deployment, scaling, and management, facilitating rapid deployment and optimization of resources while ensuring high availability and portability [10].

### D. IoT Devices

To emulate IoT devices, such as CCTV cameras we utilize Locust,[6] which is a robust load generator capable of generating HTTP or MQTT requests directed toward the application, effectively replicating real-world traffic scenarios.

### E. Monitoring Tool

We employ *sFlow-RT*[7] as our main monitoring tool, providing 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,[8] supporting sFlow protocol, for performance monitoring of hosts and servers. We also utilize standard sFlow agents to monitor Open vSwitches within the Mininet network topology, enabling efficient network monitoring. 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. Given the application's diverse microservices (pods), individual pod monitoring is vital. We accomplish this by employing sidecar containers[9] with sFlow agents alongside the main application container within each pod. To convert real-time telemetry from sFlow agents into *Prometheus*-compatible metrics, we utilize the *Prometheus Exporter*,[10] integrated with *sFlow-RT*. This integration enables *Prometheus*[11] to retrieve and utilize these metrics via a REST API. For enhanced visualization and efficient management of metrics data, we leverage Grafana.[12]

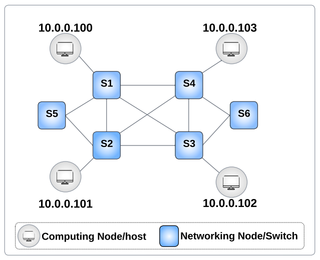
### F. Platform Automation

We have fully automated the setup of *iContinuum* using Ansible,[13] making it incredibly user-friendly. This allows *iContinuum* users to set up a complex edge-to-cloud continuum and application orchestration environment without getting

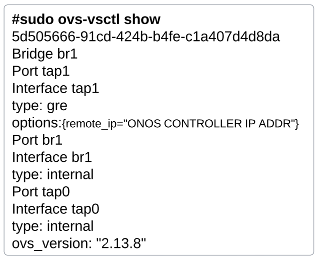
[1]https://opennetworking.org/onos/
[2]https://mininet.org/
[3]https://www.openvswitch.org/
[4]https://www.cloudflare.com/en-gb/learning/network-layer/what-is-gre-tunneling/

[5]https://k3s.io/
[6]https://locust.io/
[7]https://sflow-rt.com/
[8]https://sflow.net/about.php
[9]https://kubernetes.io/docs/concepts/workloads/pods/sidecar-containers/
[10]https://blog.sflow.com/2019/04/prometheus-exporter.html
[11]https://prometheus.io/
[12]https://grafana.com/
[13]https://www.ansible.com/

into the complexities of all the proposed tools. All associated codes are available in our GitHub repository.[14]



(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

## V. Evaluation

We evaluate *iContinuum* through testing and experimentation of sample applications featuring mixed cloud and edge components. We describe a sample scenario and experimental testbed, followed by a detailed analysis of evaluation results to offer insights into *iContinuum*'s performance and capabilities.

### A. Experimental Setup

Our experimental setup consists of five VMs hosted on the Nectar cloud,[15] 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 200 Mbps bandwidth without additional delay settings. 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).

Table 2 shows a detailed breakdown of the edge server IP addresses associated with the *tap0* interface in the range of 10.0.0.0/8. 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 allows centralized cluster management through the ONOS controller.

[14]https://github.com/disnetlab/iContinuum
[15]https://ardc.edu.au/services/ardc-nectar-research-cloud/

| 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 use Locust to send HTTP requests to the application. Our image processing application, illustrated in Figure 4, comprises four microservices available on Docker Hub.[16] These microservices resize images, convert them to black-and-white, detect objects, and trigger alarms for specific objects, mirroring factory operations. Microservices log request times, processing, and failures, sending data to a centralized database. Deployed on Kubernetes, microservices run on worker nodes (labeled W1' to W3' in Figure 4), avoiding overloading the critical Master node.

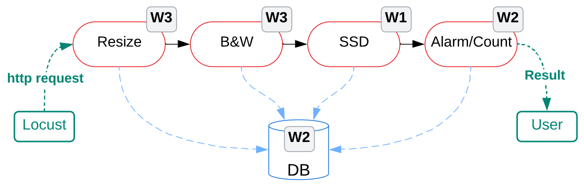


Fig. 4: Image Processing Application

### B. Results and Analysis

In this section, we 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*.

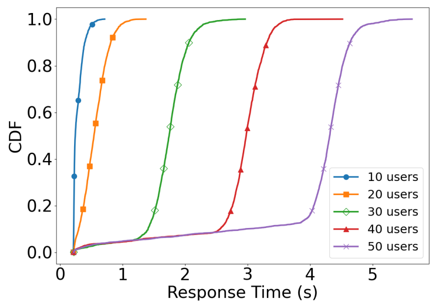


Fig. 5: CDF of response time for different concurrency levels at the workload generator

**Application Level:** To stress-test the application, Locust sends synchronous HTTP requests for 360 seconds with varying concurrent user counts, each carrying a 499.7kB JPEG image. Our setup mirrors real-world conditions by simulating different concurrency levels, similar to multiple CCTV cameras. A spawn rate of 1 user per second dictates data generation during testing. We measure Response Time (RT) from request

[16]https://hub.docker.com/repositories/negin67

receipt by Microservice 1 to completion by Microservice 4. The CDF of RTs across user counts in Figure 5 demonstrates performance under varying loads. Smaller variance between results at 10 and 20 users suggests 10-user concurrency does not saturate the system, limiting maximum throughput.

Table 3 displays application throughput and processed requests, offering insights into performance under different demand levels. The results reveal *iContinuum*'s consistent response time variation and inversely correlated throughput, confirming system reliability.

| **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 test *iContinuum*'s ability to measure computational parameters such as CPU and memory utilization. Figure 6 shows utilization diagrams for each node and pod during a 33-minute experiment, collecting data every 15 seconds. Annotations on the top of the plots mark the occurrence of various injected events into the system. 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 user per 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 reveal higher resource usage on the *Master* node, responsible for task scheduling, and on *Worker3*, hosting two microservices. *Worker1* processes object detection, while *Worker2* handles lower-demand tasks like alarm generation.

**Networking Level:** In a 360-second simulation using Locust, we tested HTTP traffic with 50 users and a spawn rate of 1 user per second to analyze the effects of bandwidth and delay settings on application performance. Figures 7(a) and 7(b) showcase how varying bandwidths, from 50Mbps down to 10Mbps, impact response times and throughput, respectively. Meanwhile, Figure 7(c) explores the influence of delays ranging from 5ms to 20ms on response times, maintaining a fixed bandwidth of 50Mbps. Corresponding throughput results are depicted in Figure 7(d). These experiments underscore *iContinuum*'s adaptability to diverse networking conditions, demonstrating its capability to dynamically adjust parameters for realistic emulation of real-world scenarios.

## *C. A Use Case: Intent-based Scheduling*

In this section, we showcase an intent-based scheduling method using *iContinuum*, aiming to maintain application response time (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 $e1$, 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. Subsequently, at $e2$, 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 via flow scheduling. Then, at $e3$, we increase the concurrent users in Locust from 10 to 20 deliberately triggering another violation of the intent. This is responded 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. This experiment clearly illustrates how intent-based scheduling algorithms can be effectively emulated by *iContinuum*.

## VI. Related Work

The compute continuum features diverse capabilities, locations, programming models, and constraints, encompassing computing power, storage, networking, and specialized components like GPUs and AI. Developers must skillfully use these resources to ensure smooth application migration across networks and service providers. [11] and [12] 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. [11] 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* [4], specializing in fog node placement algorithms within the IoT domain, is an adaptation of *CloudSim* [5]. *EdgeCloudSim* [6], another adaptation of CloudSim, offers modeling capabilities such as network configuration, mobility, and traffic patterns. *While Yet Another Fog Simulator (YAFS)* [13] does support edge topology simulation, it primarily targets the IoT domain, and its execution time is notably high, consequently leading to increased response times [14]. However, these simulation tools have limited capabilities to accurately and authentically replicate real-world scenarios.

*EmuFog* [15] 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. While EmuFog

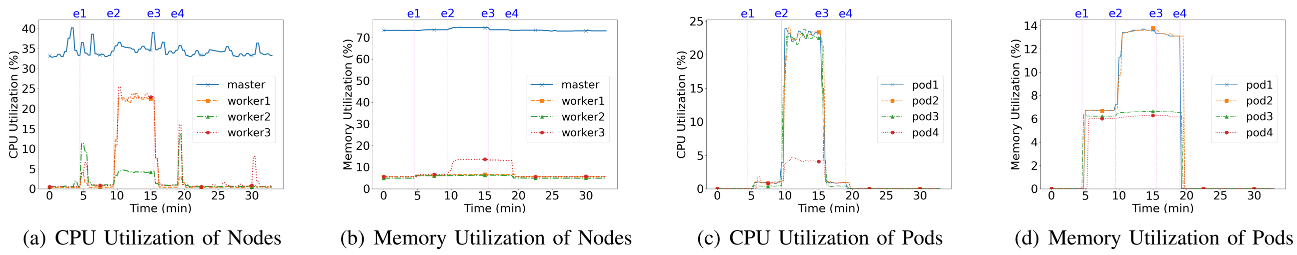

(a) CPU Utilization of Nodes (b) Memory Utilization of Nodes (c) CPU Utilization of Pods (d) Memory Utilization of Pods

Fig. 6: Nodes & Pods CPU & Mem Utilization. e1: App. Deployment e2: Traffic Gen. e3: Traffic Stop e4: App. Termination

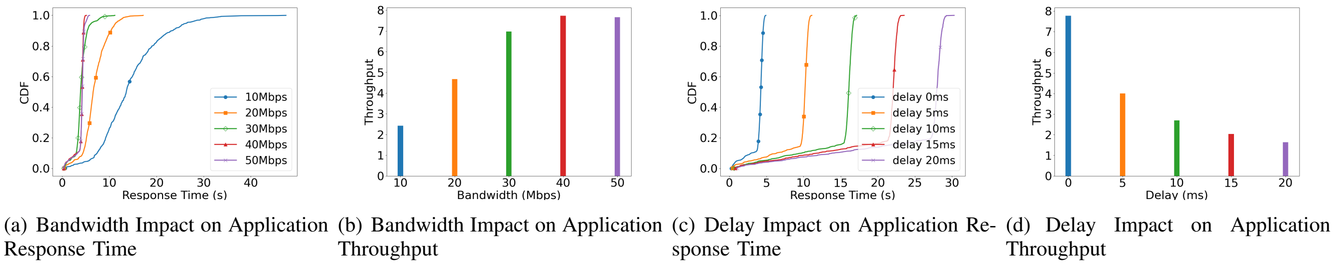

(a) Bandwidth Impact on Application Response Time (b) Bandwidth Impact on Application Throughput (c) Delay Impact on Application Response Time (d) Delay Impact on Application Throughput

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

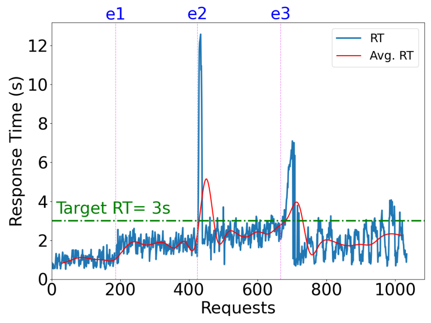


Fig. 8: Application Response Time over the time when different events are induced. Response Time (RT)- Average Response Time (Avg. RT)

utilizes MaxiNet [16] to track local node events such as CPU and memory consumption, it lacks a universal interface for monitoring global metrics such as response time [17]. *Fogify* [12] 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 [11].

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 *Containerization*. 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) [18] has emerged as a promising solution for addressing the aforementioned gap by capturing business intent and subsequently activating and ensuring it throughout the network [19]. Recent studies are focusing on extending IBN to the computing domain, including edge and edge-to-cloud domains [20], [21], [22], [23]. *iContinnum* provides seamless support for intents, enabling system administrators and developers to articulate their desired outcomes without necessitating explicit instructions on how to achieve them.

## VII. Conclusions and Future Works

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 software-defined networking and containerization, *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. Additionally, we presented a use case for intent-based scheduling using *iContinuum*, showcasing the comprehensive capabilities of our proposed toolkit. In our future work, we aim to enhance our emulation tool by integrating mobility, wireless connectivity, and pod-to-pod network management support. Furthermore, we plan to propose novel intent-based scheduling methods to optimize resource allocation based on high-level objectives using *iContinuum*.

## Acknowledgment

This project is partially supported by ARC LP210200213 and ARC DP230100081 grants. Grundy is supported by an ARC Laureate Fellowship. The authors would like to thank Dr. Tianzhang He for his valuable assistance with this work.

## References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.