An Automated Edge Computing Approach for IoT Device Registration and Application Deployment

Vitumbiko Mafeni and Younghan Kim, *Member, IEEE*

***Abstract*—As the Internet of Things (IoT) evolves rapidly across various industries, the number of IoT protocols and applications is growing with vast number of heterogeneous components and entities. In setups with thousands of IoT devices, manual deployment of applications and device registration become impractical due to their time-consuming and costly nature, as well as the requirement for background knowledge of IoT devices and protocols. Furthermore, IoT devices often have resource constraints that prevent them from running complex software. Therefore, there is a significant need to enhance and optimize edge computing systems for IoT, making them suitable and dynamic for automated IoT device registration and heterogeneous application deployment. In this article, we present an edge-based framework designed to facilitate the automated registration of diverse wireless IoT devices and the deployment of IoT applications. To validate our approach, we use a smart irrigation system enhanced with a containerized machine learning model as a proof of concept. Our evaluation of the implemented prototype demonstrates that our system is scalable and feasible.**

***Index Terms*—Automation, edge computing, Internet of Things (IoT), service orchestration, wireless sensors.**

I. Introduction

THE expanding landscape of the Internet of Things (IoT) has led to the widespread deployment of devices and sensors, dispersed across various locations to perform diverse tasks in our daily lives. However, the integration of these physical devices with cloud-based deployments has introduced complexities in configuring and deploying IoT applications, presenting a significant challenge that requires prior technical expertise [1].

Traditionally, the conventional approach involves transmitting sensor data to the cloud for processing and subsequently dispatching control signals to relevant actuators within IoT applications. This method, while widely used, has inherent drawbacks, including increased latency, heightened network load, and compromised privacy [2]. The emergence of edge computing has arisen as a solution to alleviate the strain on cloud infrastructure caused by massive data volumes, intricate network configurations, and scalability concerns. However, deploying applications at the edge for IoT remain a formidable challenge [3], exacerbated by the heterogeneous nature of protocols, device data formats, communication capabilities, technologies, and hardware [4], [5], [6].

Moreover, the sheer diversity of IoT devices, numbering in the thousands and millions, necessitates interoperability at various levels [7]. Consequently, there is an imperative need for a reliable and automated management system capable of supporting the rapid growth and complexity of the IoT environment in a scalable manner. Manual deployment of IoT applications on numerous devices within a heterogeneous setup is not only tedious and error-prone, but also time-consuming. In dense IoT deployments, novel business models like sensing as a service demand increased automation for the provisioning and maintenance of IoT installations [8].

One pertinent application of IoT technology lies in agriculture, specifically in the realm of smart farming practices. This includes intelligent irrigation and livestock management, essential components for enhancing agricultural efficiency. Notably, farming predominantly occurs in rural regions, where a significant portion of farmers may lack awareness and understanding of how to effectively utilize or configure IoT technologies to optimize their operations [9].

Consider a scenario where human intervention is required for intricate steps in the registration and configuration process of IoT devices. When an IoT device is poised to join a network, transmit data, or access services, it needs to undergo registration [10]. This process includes the manual entry of its unique identifiers (such as device name or serial number, device type, and device model and manufacturer information), along with other configuration settings if applicable. This manual involvement is prone to errors due to a deficiency in awareness and knowledge of the underlying procedures. Consequently, a substantial number of IoT devices seeking network entry or configuration can lead to a laborious, error-prone, and time-consuming process when carried out manually [11].

To address these challenges, we propose an edge computing approach. For efficient management and processing of data originating from IoT devices, an edge service must exhibit autonomy, statelessness, and portability. These attributes are crucial for enabling quick migrations and deployment of edge services across the edge-cloud-computing environment while maintaining a high level of service availability. A powerful solution for deploying services and applications with ease is

Manuscript received 22 November 2023; revised 27 February 2024; accepted 17 April 2024. Date of publication 3 May 2024; date of current version 20 June 2024. This work was supported in part by Institute of Information and communications Technology Planning and Evaluation (IITP) grants funded by the Korea government (MSIT) under Grant 2020-0-00946, and in part by the Development of Fast and Automatic Service recovery and Transition software in Hybrid Cloud Environment) and the MSIT, Korea, under the ITRC (Information Technology Research Center) support program under Grant IITP-2024-RS-2023-00258649 supervised by the IITP. *(Corresponding author: Younghan Kim.)*

The authors are with the School of Electronic Engineering, Soongsil University, Seoul 06978, South Korea (e-mail: vitumafeni@dcn.ssu.ac.kr; younghak@ssu.ac.kr).

Digital Object Identifier 10.1109/JSYST.2024.3392290

through container virtualization [12]. We leverage this approach to orchestrate services and utilize portable containerized components in IoT services. Kubernetes [13] and Docker Swarm [14], well-known solutions for cloud orchestration, are employed in various popular orchestration solutions for edge computing in IoT, such as Akri, Kubeedge, Flotta, and K3s [15].

In this article, we develop and implement an edge computing platform designed to automate application deployment and manage the configuration of IoT devices without human intervention. To validate our approach, we apply it to a smart irrigation farming system, incorporating a machine learning model. The agricultural sector, with its unique challenges and predominantly rural settings, exemplifies the need for automated solutions due to potential limited awareness and technical understanding among farmers [16]. In summary, our contributions encompass the following.

1) Present an innovative orchestration framework based on Kubernetes designed to enable the automated deployment and management of IoT applications within heterogeneous computational environments.
2) A mechanism to remove the need for human involvement in the configuration and registration of IoT devices. We introduce an automated solution to seamlessly integrate resource-constrained IoT devices into a centralized cluster through a Kubernetes-based abstraction layer.

To demonstrate feasibility and scalability, we implement a containerized smart irrigation farming system, supported by a recommendation machine learning model as a workload deployed

The rest of this article is organized as follows: Section II provides a review of the research background. Section III outlines our proposed solution. Section IV delves into the experiment setup and evaluation. Finally, Section V concludes this article.

II. Background

This section provides a foundation, summary, and overview of the fundamental concepts and technologies that form the basis of the proposed system.

A. Edge Computing

Focusing on edge computing, the article in [12] explores the potential of edge computing as a means to enhance service delivery and performance by extending cloud resources closer to service environments. This article conducts an analysis of existing implementations of processing environments with edge resources, considering quality of service (QoS) parameters and orchestration platforms. It evaluates popular edge orchestration platforms based on their ability to incorporate remote devices into processing environments and adapt scheduling algorithms to enhance QoS attributes. Experimental results compare platform performance, indicating that Kubernetes and its distributions have promise for effective scheduling at the network's edge. However, challenges remain in adapting these tools for the dynamic and distributed nature of edge computing environments as it does not address wireless device heterogeneity on the edge.

Ullah et al. [15] presented MiCADO-Edge, an extension of the MiCADO cloud orchestration framework tailored for multilayered cloud-to-edge environments. It automates microservices deployment across cloud and edge nodes, emphasizing monitoring and enforcing user-defined runtime management policies. Real-world case studies in video processing and secure healthcare data analysis validate MiCADO-Edge's practicality.

A framework, FogDEFT, introduced in [17] investigates the limitations of conventional cloud-centric IoT applications in meeting latency requirements. This fog computing federation framework leverages topology and orchestration specification for cloud applications (TOSCA) for service deployment within fog environments. FogDEFT standardizes distributed application design using TOSCA service templates, ensuring interoperability across heterogeneous fog devices. This framework overlooks automated IoT wireless devices registration.

In the article [18], an edge computing IoT-based solution for strawberry farming addresses challenges by collecting, analyzing, and predicting data related to cultivation. The platform integrates monitoring services, manages IoT devices for data collection, and employs computer vision for real-time disease detection. However, it lacks details on automating IoT application deployment and device registration.

Furthermore, we also came across [19], which investigates on resource orchestration within the device-edge-cloud continuum, with a specific emphasis on edge artificial intelligence (AI). The article argues that to meet the growing demands of intelligent applications in this continuum, resource orchestration should incorporate edge AI, emphasizing local autonomy and intelligence. However, we find it lacking in terms of edge automation.

Resende [20] presented a solution that is designed to automate the provisioning and life cycle management of IoT end nodes. The study mentions that current methods of provisioning and managing of IoT end nodes are manual involving static software deemed to be insufficient for expansion.

While our study focuses on edge computing, highlighting software defined wireless sensor networks (SDWSN) is crucial due to their tight relation to IoT. Wireless Sensor Networks (WSN) face challenges like limited computational capability and energy constraints. software defined networking (SDN), with an external controller managing network intelligence, aims to address these challenges. SDWSN merges these concepts, enhancing efficiency, and sustainability [21], [21], [23].

Comparing edge computing and SDWSN highlights their differences. SDWSN focuses on networking while edge computing emphasizes local processing. In addition, SDWSN may rely on centralized servers for scalability which may face latency issues while edge computing distributes computation for independent local processing.

B. Containerization and Container Orchestration

Several articles are dedicated to containerization and according to the official documentation [25], [24], a container can be described as a standardized software unit that bundles both code and all its dependencies. Containerization is a technology that enables developers to package their applications and dependencies into lightweight, portable containers that can run anywhere [25], [26].

Containerization guarantees application portability, simplifying deployment, testing, and migration. Containerized applications can scale up or down seamlessly in response to changing environmental conditions, such as varying traffic demands. Furthermore, containers ensure reproducibility, maintaining a consistent working environment regardless of the number of executions [27]. Docker, a leading containerization platform, offers a robust ecosystem of tools and services for building, deploying, and managing containers [28].

Container orchestration, as defined in [29], automates the deployment, supervision, scaling, and networking of containers. Various tools, including Kubernetes, Docker Swarm, and Apache Mesos, manage container life cycles effectively. Kubernetes, originating from Google engineers, enables users to build application services spanning multiple containers, schedule container instances across clusters, dynamically adjust counts, and oversee long-term operational status. Kubernetes streamlines manual procedures associated with deploying and scaling containerized applications.

C. Frameworks and Tools

While numerous studies have explored novel architectures that enhance edge computing in various ways, many of these frameworks lack certain components necessary for automation.

TOSCA is employed in [2] to introduce the FogDEFT framework, designed to address interoperability and heterogeneity in IoT devices. This framework presents a user-friendly paradigm for modeling and dynamically deploying fog services remotely, on-demand, and on the fly. However, in contrast to our methodology, this research overlooks the management of a larger quantity of IoT devices.

EdgeX is a framework to manage profile information for various IoT devices and support AI inference model operation presented in [30]. The article highlights the need for edge computing and weaknesses of cloud computing, such as delayed response time for processing service requests and security concerns regarding data confidentiality. However, we find the framework proposed to lack specific details on how the ML applications and services can be deployed and scaled efficiently to support increasing traffic demands in case many devices are connected. In addition to that, we also find it lacking a procedure on automating IoT device registration hence requiring human intervention to handle configurations.

In our study we simplify IoT device registration process, which involves carefully considering security measures to confirm the legitimacy of connecting devices before granting them access to services. In this context, a relevant study [31], introduces a fog computing-based system model for IoT architecture, focusing on remote authentication to ensure secure device-to-device communication. While authentication is vital, it is just one aspect of the broader framework. Automating decision-making for communication requests between devices, eliminating human intervention, can reduce errors, enhance system availability, and better support time-sensitive applications. Containerized and automated authentication approaches offer effective automation solutions. In such approach, it can enable seamless scaling using orchestration tools to efficiently verify the presence of unique IDs among IoT devices within a database of known devices. However, the methodology presented in this article does not incorporate containerization and orchestration. The absence of these elements limits the utilization of declarative methods for automation.

Several papers [32], [30], [34] present approaches and architectures based on TOSCA to automate deployments of applications in the cloud or extending the cloud to the edge. However, TOSCA has limitations and not flexible on the edge due to its inability to keep track of run time deployments and offer reconfiguration decisions on the fly. Although TOSCA aims to be vendor-agnostic, the manuscripts above do not specify how to handle scalability issues in TOSCA. When managing very large and complex applications or when dealing with rapid scaling requirements. Orchestrating and managing numerous components can become resource-intensive.

Ferry et al. [35] introduced another framework called SERVERLEss4I0T in their publication. This framework functions as a platform for creating, deploying, and managing applications across the Cloud-Edge-IoT spectrum. Specifically, they present the platform as a tool for specifying and deploying serverless functions on both Cloud and Edge resources, along with deploying necessary supporting services and software stacks throughout the entire Cloud-Edge-IoT continuum. To validate their solution, tests were conducted on a smart building equipped with multiple IoT devices. While the manuscript provides valuable insights, it does not explicitly clarify the scalability and feasibility of their system in large-scale IoT environments.

The authors introduced the smart environmental monitoring and analytical in real-time (SEMAR) framework in their publication [36]. The article's primary objective is to enhance the development of smart cities by presenting SEMAR as a solution to the challenges associated with integrating various IoT application systems. SEMAR is designed to offer flexibility, interoperability, and efficient data handling. Significantly, this framework lacks the integration of orchestration or containerization technologies for streamlined deployments, and it also overlooks auto-scaling, a crucial aspect for accommodating increased traffic from IoT devices.

SEnviro Connect framework is proposed in [37] focusing on the role of IoT platforms as a transversal middleware in addressing challenges like device heterogeneity, support for a large number of connected devices, system reliability, safety, energy efficiency, and third-party usability. While employing containerization via Docker, this method neglects the aspect of scaling up containers to handle additional requests. Furthermore, the framework fails to capitalize on orchestration tools for harnessing declarative or internet-based configurations, thus missing out on opportunities for automation.

Numerous cutting-edge open-source orchestration frameworks are specifically designed to tackle edge computing challenges within the realm of IoT devices.

KubeEdge is an open-source system designed to expand the native orchestration capabilities of containerized applications to edge hosts, as detailed on their official website [38]. By leveraging Kubernetes, this platform specifically focuses on orchestrating containerized IoT application services within IoT edge computing environments. In a recent article authored by Kim [39], an in-depth evaluation of the framework is presented, encompassing aspects such as the distribution of computational resources and the latency experienced between various edge nodes. However, configuring KubeEdge can be challenging, requiring complex knowledge about networking and operating system to deploy it [40].

The Akri [41] framework is another noteworthy player in the IoT-edge computing domain, functioning as a Kubernetes Resource Interface. It simplifies the integration of diverse leaf devices, such as IP cameras and USB peripherals, into a Kubernetes cluster. Akri also supports embedded hardware resources like GPUs and FPGAs. Notably, it efficiently identifies nodes with access to these devices and manages workload scheduling based on their availability. However, it is important to highlight that Akri does not provide out-of-the-box support for IoT wireless device discovery.

Last, Flotta is another framework applicable in the domain of IoT-edge computing. As described in the framework's documentation [42], it introduces an efficient approach to centralize the management of edge devices. This is achieved through the use of Kubernetes Custom Resource Definitions (CRDs), which excel at overseeing diverse device configurations. However, Flotta does not support the an automated registration, management and automation of IoT wireless devices. Conversely, it does not offer support for the automatic scaling of pods in instances where there is a sudden increase in traffic from IoT devices. Energy monitoring, a crucial aspect of IoT, is also not covered as an out of the box feature in Flotta.

Upon reviewing the existing literature on related works, frameworks, and tools for facilitating edge computing, several common factors have emerged that contribute to the realization of automation, scalability, and interoperability at the edge. The accumulated research affirms the substantial potential of edge computing, particularly in the context of IoT. Among the most noteworthy and prevalent elements that facilitate automated deployment of IoT applications and ensure interoperability are as follows.

1) Containerization: Bundles application libraries and dependencies into a single component, simplifying application migration and scaling. This approach enables easy redeployment without extensive reconfiguration, as all dependencies are packaged together.
2) Orchestration: Allows for efficient and repetitive deployment of the same application, enhancing adaptability in dynamic computing environments.
3) Wireless Device Discovery and Registration: Streamlines onboarding and management, ensuring scalability, interoperability, security, and effective device monitoring in dynamic environments. The framework should seamlessly add new IoT devices, automatically configuring them based on their attributes to minimize manual setup.
4) Uniformity and Interoperability: Ensures that devices with diverse protocols and data formats can connect and communicate seamlessly, fostering a cohesive and efficient IoT ecosystem.
5) Heterogeneous Wireless Device Support: An effective IoT edge computing framework should accommodate diverse wireless devices, managing various protocols and connectivity standards. This flexibility enables the framework to adapt to evolving IoT deployments, beyond specific protocols or wired devices.

Table I presents a summary of comparison of the discussed frameworks concerning their alignment with the primary focus of our manuscript. We illustrate the key components necessary for an automated edge computing framework. In the table, "✓" signifies the presence of a particular factor or component within the framework, while an "x" indicates not supported.

III. Proposed Framework for IoT Device Registration and Application Deployment

This article presents an automated framework specifically crafted for deploying IoT applications and facilitating the registration or onboarding of IoT devices. The framework is tailored to address the intricacies of IoT environments marked by the diversity of wireless communication technologies and protocols, including Zigbee, Sigfox, Bluetooth Low Energy (BLE), and Wi-Fi.

The proposed framework acknowledges that users may lack expertise in establishing IoT systems, especially in deploying IoT applications and conducting device registration, as discussed in [9]. Automation becomes crucial in addressing these challenges, streamlining IoT processes, reducing errors, optimizing costs, and meeting stringent QoS requirements, particularly for time-sensitive applications. In conventional farming, farmers spend a significant amount of time monitoring and assessing crops, as highlighted in [43] and [44], making smart agriculture essential to optimize farming practices and minimize manual labor.

Fig. 1 depicts the comprehensive architecture of our proposed framework, seamlessly built on top of an existing edge computing framework, namely Flotta. The proposed framework is strategically divided into three principal components: the cloud components, edge components, and wireless end-node devices. Here, we describe the roles and functions of each component within the framework.

A. Cloud Components

The cloud component acts as the central server, managing edge, and end nodes comprehensively. It deploys workloads and monitors edge node status continuously. During setup, critical information about wireless IoT end nodes, like unique identifiers and device names, is stored in the cloud data store. This ensures discoverability and enables automatic registration for preregistered end nodes, eliminating the need for human intervention.

TABLE I
COMPARING THE MENTIONED FRAMEWORKS FOR ENABLING AUTOMATED IoT EDGE COMPUTING

Framework	Containerization	Wireless IoT Devices Auto-Discovery & Registration	Heterogeneous Wireless IoT Device Support
FogDEFT [2], [17]	✓	x	✓
EdgeX [30]	x	x	✓
RemoteAuth. BlockChain [31]	x	x	✓
SERVERLEss4I0T [35]	✓	x	✓
KubeEdge [38]	✓	x	✓
Akri [41]	✓	✓	x
Project-Flotta [42]	✓	x	x
SEMAR [36]	x	x	✓
SEnviro Connect [37]	✓	x	✓
Our Framework	✓	✓	✓

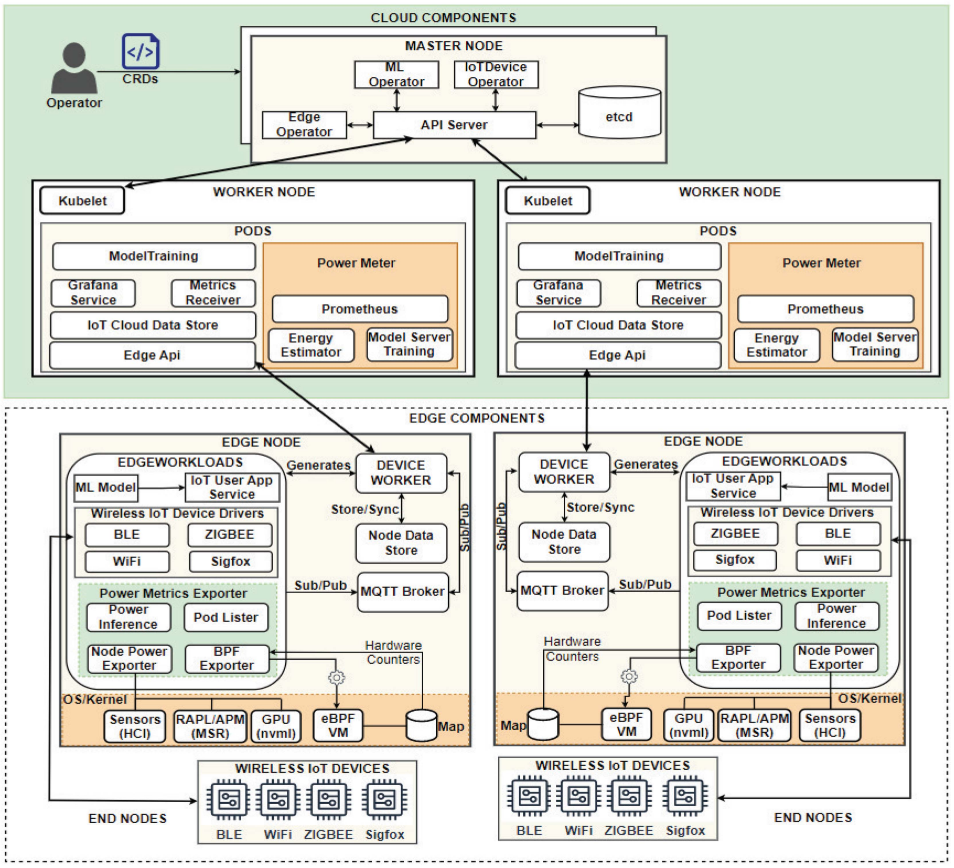

Fig. 1. Proposed architecture for automating IoT devices registration and applications deployment.

Our framework, built on Project-Flotta, harnesses Kubernetes CRDs for enhanced capabilities. An operator creates a configuration CRD, typically an EndNodeAutoConfig YAML file, defining criteria for identifying end node devices as they connect to the platform on the edge. Criteria include wireless communication, protocol, workload to deploy upon device discovery, and selection of the edge node for configuration application. If no specific edge node is designated, the configuration applies uniformly to all edge nodes in the cluster.

Applying the CRD configuration deploys the specified device driver to the preferred Edge Node, initiating the discovery process and listening for specified wireless end devices. Upon detecting a designated end node device, the edge operator orchestrates automatic deployment of the specified edge workload, often a machine learning model. An ML controller facilitates model training within the cloud environment. For reference, an example of the EndNodeAutoConfig Custom Resource (CR) is provided in Listing 1.

Integrating with the kubernetes efficient power level exporter (Kepler) open-source project [45], we have enabled energy monitoring within our framework. The power meter pod operates on the cloud worker nodes, serving to display metrics collected from the edge regarding power usage for each edge node. In addition, it incorporates a machine learning component utilized for predicting the energy consumption of each edge node. The power estimator leverages this machine learning model to offer future analytics.

Listing 1: Kubernetes CR YAML for EndNodeAutoConfig.

```
apiVersion: mgmt.project-flotta.io/v1alpha1
kind: EndNodeAutoConfig
metadata:
  name: endnodeautoconfig-sample
  namespace: default
spec:
  device: 53974c0582c543c28
  configuration:
    connection: ble
    devicePlugin:
      containers:
      - image: vitu1/plugin-ble:1.1
        name: device-plugin
    protocol: ble
    workloadSpec:
      containers:
      - image: vitu1/ml-app:1.20
        name: machine-learning-iot
    recoveryAction:
      type: alert
      alertConfig:
        recipients:
        - email: email1@example.com
          phone: "+1234567890"
        - email: email2@example.com
          phone: "+9876543210"
      alertTrigger:
        sensorMalfunction:
          condition: value < 0.0
```

The algorithm described in Algorithm 1 illustrates the automated deployment of a designated workload. In addition, during subsequent heartbeats, if the end node is identified as being in an unhealthy state, predefined actions are executed. For instance, in the event of an IoT application crash or unresponsiveness, the recovery procedure may include an automatic application restart. Alternatively, alerts or notifications may be dispatched to inform administrators or operators of the problem, enabling them to intervene manually if required.

B. Edge Components

At the edge, the edge node establishes a secure connection with the cloud node through the exposed Edge application programming interface (API), employing mutual authentication. To sustain this connection, the edge node sends a heartbeat signal

Algorithm 1: Automated IoT Application Deployment.

```
1:  while true do
2:    $EdgeManager$ receive a heartbeats from $EdgeNode$ every $HeartbeatTimer$
3:    $EdgeManager$ verify request and maintains mutual authentication with $EdgeNode$.
4:    if $isEndNodeRegistered(EndNodeID) = false$ then
5:      $Manager \leftarrow EdgeManager$.Save to $Observer$ from $EdgeNode$.
6:      if $writeDeviceInfoToStatus(EdgeNode) = true$ then
7:        $Manager \leftarrow Configuration$.Automated deploy application
8:        return Success
9:      else
10:       return Error
11:     end if
12:   else
13:     if $isDeviceHealthy(EndNodeID)) = false$ then
14:       $Manager \leftarrow Configuration$.RecoveryAction
15:     end if
16:   end if
17: end while
```

every 5 min to the Edge API as a Service on port 9001. During each heartbeat exchange, the edge node, centrally managed by the device worker, undertakes several crucial tasks. It retrieves new configurations from the cloud, updating its settings accordingly. In the presence of a valid internet connection, it syncs with the cloud, pushing and pulling relevant updated data and configurations. As part of this process, the edge node conducts a data pull operation, retrieving and storing all pre-existing end node data in a Node Data Store, typically utilizing SQLite for this purpose. The configuration also includes specifications for device drivers, outlining the image and repository required for the image to function as a workload through Podman. Once the image is retrieved and initiated, it actively scans for compatible end node devices within its range. Upon activation of an end node device by a user (e.g., a farmer), the device driver extracts the device's unique identifier (serial number) and device name.

The wireless IoT device drivers employ event-driven communication through the message queuing telemetry transport (MQTT) broker. Here, the IoT wireless device driver publishes the gathered information to the MQTT broker, while the device worker subscribes to all relevant topics. The device worker, is responsible to converting to a standard data format for all types of IoT end nodes and push to the server. Likewise, it converts and maps the instructions from the cloud to specific data format for an IoT end node through the MQTT broker then the device driver. The following are the topics;

- `$device/edge/upstream/ble`
- `$device/edge/upstream/+`
- `$device/edge/upstream/availability/ble`

- `$device/edge/upstream/availability/+`
- `$cloud/driver/downstream/ble`
- `$cloud/driver/downstream/+`

These MQTT topics are categorized into three distinct groups: upstream, downstream, and availability.

Within the upstream topics, such as `$device/edge/upstream/ble`, signifies that data from BLE devices is transmitted. If there were another wireless protocol in use, such as Wi–Fi, the corresponding topic would be formatted as `$device/edge/upstream/wifi`. Similarly, for downstream topics, the structure follows suit. The availability topic serves the purpose of allowing IoT devices to report unexpected disconnections.

The device worker then cross-references the incoming IoT device information with the data stored in the Node Data Store. If a match is found, the IoT wireless device is granted automatic registration without the need for further user configurations. During subsequent edge node heartbeats, the edge node updates the cloud regarding the status and information of connected wireless IoT devices. It does this by pushing relevant information to the cloud, contingent on an available internet connection. The communication between the device worker and various device drivers occurs via MQTT topics, facilitating data exchange and coordination. Moreover, our framework is extensible by adding more device drivers to support more IoT wireless communication technologies and protocols.

Furthermore, our framework categorizes end node devices into two types: input and output devices. Input devices can accept commands, such as switches or actuators, enabling them to execute boolean functions like toggling ON/OFF. Conversely, output devices, like sensors, solely observe and collect environmental data without accepting commands. The configurations retrieved from the cloud contain specific commands to be transmitted to end node devices capable of receiving such instructions. The device driver plays a critical role in mapping data from diverse formats into a universal format understood by the device worker. Upon successful device registration and the transmission of status updates to the cloud via the Edge API, the specified workload, as configured in the EndNodeAutoConfig CR, is automatically deployed. In our specific case, we deploy a Machine Learning model that serves as a preprocessing tool and functions as a recommendation system for optimizing the smart farm irrigation system. The algorithm 2 shows the process of an automated device registration as explained.

Ultimately, our utilization of Kepler, leveraging eBPF [46], facilitates dynamic kernel energy monitoring aspect. Kepler harnesses eBPF to obtain performance counters and various system statistics to gauge workload energy consumption based on these metrics, subsequently exporting them as Prometheus metrics. These exported metrics, through device workers on the edge to the cloud, are then accessed by the Power Meter pod located within the cloud worker nodes during each heartbeat transmission.

Algorithm 2: Automated IoT Device Registration.

```
1:  while true do
2:    $EdgeNode$ sends a heartbeat signal to $CloudNode$ every $HeartbeatTimer$.
3:    Maintain mutual authentication with $CloudNode$.
4:    if $HeartbeatTimer > 0$ then
5:      $Config \leftarrow EdgeNode$ GET $Configuration$ from $CloudNode$.
6:      if $!empty(Configuration)$ then
7:        $NodeDataStore \leftarrow EdgeNode$ sync $CloudNode$.
8:        if $pluginDiscoverWirelessEndNode() = true$ then
9:          if $deviceLocalDB(EndNodeID) = true$ then
10: if $registerDevice(EndNodeID) = success$ then
11: return Success
12: else
13: return Error
14: end if
15:         end if
16:       end if
17:     end if
18:   end if
19: end while
```

IV. Experiment Setup and Evaluation

Our experimental system configuration encompasses a cloud cluster, an edge node, and a network of IoT wireless devices. The cloud cluster consists of one master node and two worker nodes, all of which are Nucs equipped with Intel Core i5-5250 U CPUs running at 1.60 GHz. Each node boasts 8 GB of RAM, 500 GB of ROM, and is outfitted with 2 CPU cores. Our setup relies on Kubernetes version 1.23 and Flotta version 0.2.0. To achieve automation, we have developed an operator using Kubebuilder [47], and the corresponding GitHub repository for our operator can be accessed online.[1]

On the edge side, our test-bed configuration leverages commonly used IoT hardware platforms. We use the Raspberry Pi Model 4B with a 1.5 GHz 64-bit quad core ARM Cortex-A72 processor and 4 GB of RAM. In our setup, we employ an Ethernet connection on the Raspberry Pis as the backbone network to connect to the cloud. Moreover, each Raspberry Pi is equipped with a Zigbee USB Gateway stick (ConBee II) and a manufacturer builtin Bluetooth module to capture Zigbee signals and BLE signals respectively not forgetting Wi–Fi communication.

For our end nodes, we utilize NodeMCU devices, including both ESP32, which offers BLE and Wi-Fi capabilities, and ESP8266, which provides Wi-Fi functionality. These nodes are equipped with DHT11 temperature and humidity sensors. We test Zigbee devices by using Aqara temperature and humidity sensors. Table II provides details about the devices we utilized for testing our framework. We created 118 configurations for the IoT devices, deploying them with various settings, attributes, and capabilities. Virtual IoT devices are created to easily achieve hundreds of IoT devices connected to our framework. Our experiment primarily focused on comparing with the manual

[1]https://github.com/vitu1234/flotta-operator

TABLE II
END NODE DEVICES SIMULATED IN OUR EXPERIMENT

Device Name	Device Type	Data Units	Protocol
Aqara Hub	Sensor	Degree (°C) & Humidity (%)	Zigbee
NodeMCU Amica	Sensor	Degree (°C)	BLE
ESP32 Devkitv1	Sensor	Degree (°C) & Humidity (%)	Wi-Fi
Arduino Uno	LED/Actuator	Boolean	BLE
Arduino Yun	Sensor	Level (%)	Wi-Fi

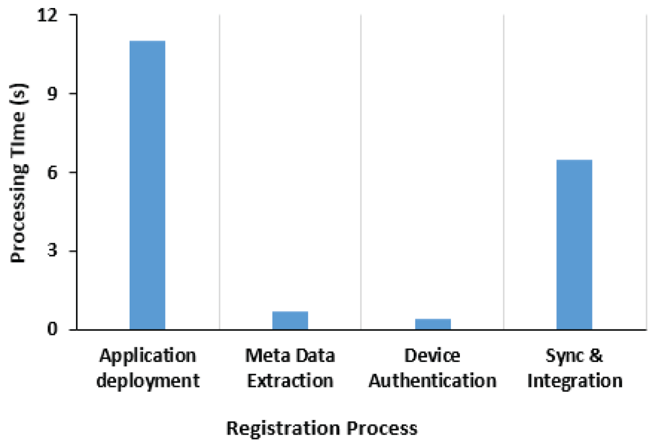

Fig. 2. Processing time for end node registration to IoT applications deployment.

configuration method and comparison with another framework, SEMAR.

The manual device registration process involves entering information for each device into the system, either by manual input or scanning a manufacturer's QR code. Users input details for device identity, security credentials, and communication parameters. This includes a unique device name, type, authentication setup, communication protocols, and security protocols like TLS/SSL. Additional details such as manufacturer info, firmware version, and a device description may also be provided but not required. Upon connecting to the network, the device may get manual approval before being provisioned.

Unlike the manual registration process, to successfully register end-node devices and deploy IoT applications automatically, a four-step process is followed. It begins with metadata extraction, collecting essential information about the device. Then, device authentication verifies the device's identity by cross-referencing its unique identifiers with data from the cloud. Next, IoT applications are deployed, with the initial deployment taking longer due to image retrieval. Last, synchronization and integration ensure the device stays updated with the latest settings. Metadata extraction is vital for authentication, and the time for each step is represented in Fig. 2.

In our experiment, we aimed to showcase the practicality of our framework by evaluating its performance in automatically registering IoT end nodes and deploying predefined applications

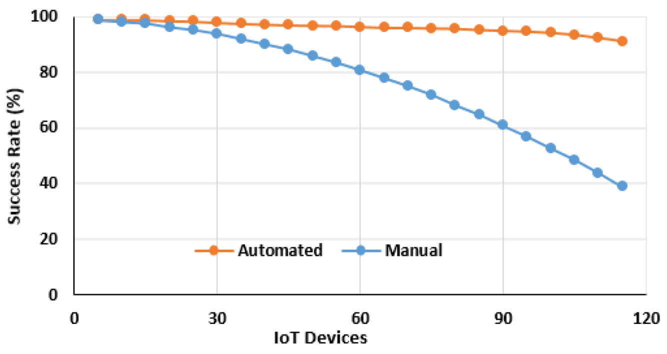

Fig. 3. Success rate, manual versus our framework.

compared to a manual approach. The registered IoT end nodes appear in the web dashboard and Kubernetes control plane on the EdgeDevice CR.

We begin our evaluation by comparing the success rates of manual and automated IoT device registrations. The success rate measures the percentage of registrations completed without errors, assessing complexities like managing duplicate attributes, addressing network errors, and enforcing device name/ID limitations. It also considers accuracy of information, error-handling, validation checks, and system reliability, providing insights into the effectiveness and reliability of our registration process.

$$\mathrm{SRT}(\%) = \frac{\mathrm{SD}}{D} \times 100 \tag{1}$$

where

SRT is the Success Rate expressed in percentage,

SD represent the number of successful deployments and

device registration,

D represent the total number of deployments, failed

and successful.

In contrast to the manual approach, our framework incorporates validation mechanisms during device registration to detect and address duplicate attribute values, significantly reducing the likelihood of data inconsistencies and conflicts within the IoT network. We use the mathematical equation in 1 to calculate the success rate. While deployment success varied, our framework consistently achieved a success rate above 90%, surpassing the manual system, especially as the number of IoT devices increased, resulting in a decreased success rate, as depicted in Fig. 3.

We further compared the efficiency of our automated framework with a manual approach for IoT device registration and application deployment. In the manual process, tasks are sequential and labor-intensive, resulting in linear growth patterns. Our automated framework, on the other hand, registers devices and deploys applications automatically, exhibiting constant growth rates and significant efficiency and scalability advantages. For 118 IoT devices, the manual approach takes up to 22 min, while our automated method completes the process in just 1.5 min,

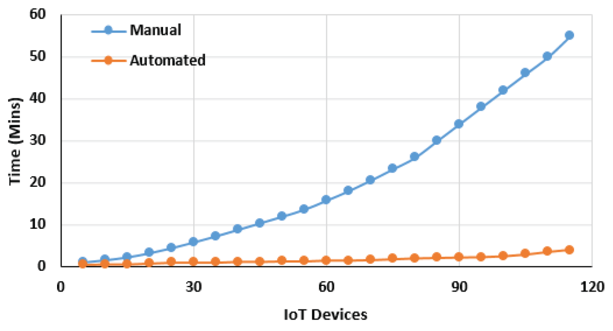

Fig. 4. IoT End Nodes registration and application deployment time, Manual vs Our Framework.

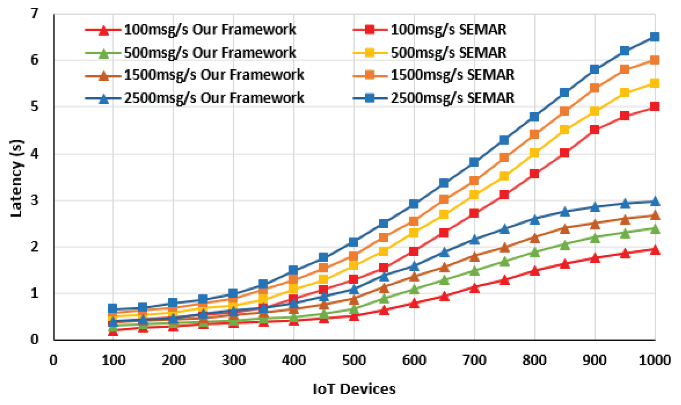

Fig. 5. Latency at various rates.

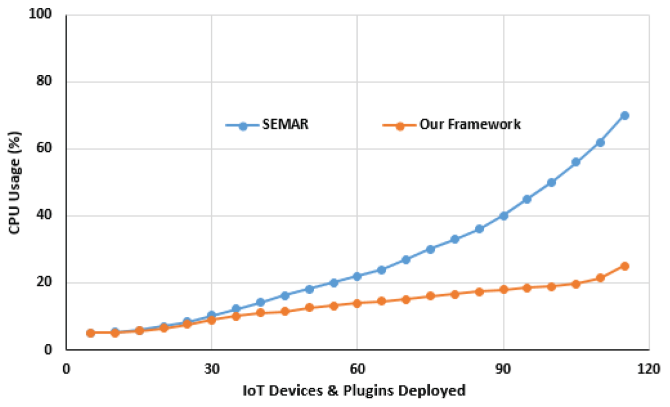

Fig. 6. CPU utilization.

highlighting the clear benefits of automation in saving time and enhancing scalability in IoT deployments. Fig. 4 reveals significant results from this comparison.

We continue evaluating our approach through an assessment of our framework's performance as we scaled up the number of IoT devices by comparing it to another framework (SEMAR) in [36]. To start the comparison, we conducted a thorough examination of system performance by conducting a latency experiment. This experiment focused on communication between edge nodes and IoT end nodes, comparing it to SEMAR. Four distinct test scenarios were involved with varying message rates—100, 500, 1500, and 2500 messages per second—while concurrently increasing the number of IoT devices.

Messages used for ingestion, generated via simulation, represent distinct phenomena. They average 55 bytes for up-link transmission and 30 B down-link from the device plugin. These messages consist of six primary components: header, metadata, device properties, security, control, and additional information. The header contains device specifics like ID and connection details. Metadata provides information on device properties. Security enables secure transmission with validation tokens. Control includes device acknowledgment and operational status. Additional information includes longitude and latitude based on the main board's geographical position.

Notably, in our framework, the maximum latency recorded was 2.98 s during the most demanding scenario: transmitting 2500 messages per second from 1000 IoT end nodes. In contrast, SEMAR experienced a latency of 6.6 s under identical conditions and with the same number of IoT nodes. At lower message rates and increased device numbers, latency exhibited a slight upward trend but remained within acceptable limits. The marginal differences in latency across diverse test cases suggest that our system performs consistently well, even under extreme conditions. This underscores the system's efficiency and robustness, indicating positive scalability and a capacity to handle substantial workloads. Latency persists at higher value across all test cases within SEMAR. The evaluation of this explanation is illustrated in Fig. 5.

Our automation algorithms are designed for efficiency, ideal for resource-constrained IoT environments. Each algorithm has constant time complexity (O(1)), with perpetual loops involving operations like conditional checks, function calls, and assignments, all considered constant-time factors. Space complexity is also low (O(1)), as the algorithms primarily use variables for references and data, without variable-sized data structures. The computational load remains consistent over time due to perpetual loops involving operations, such as heartbeat verification, mutual authentication, configuration retrieval, and device registration.

After experimenting with our algorithms, the outcomes indicate that our framework effectively manages a substantial number of devices without depleting available resources. Notably, with 125 IoT devices connected to both frameworks, the highest CPU usage reached 56% in our framework, while SEMAR framework reached up to 75% CPU usage for the same number of devices. However, performance may be affected by factors, such as the efficiency of underlying functions, external dependencies, and network conditions. It must also be noted that we only tested up to 120 IoT devices for CPU usage as the workloads used were resource intensive and we could not exceed connecting beyond 120 IoT devices. These findings are visually depicted in Fig. 6.

We conducted further comparison between our framework and SEMAR with a focus on response time. As elucidated in [36], measuring response time, determining the time difference between transmitting data to the device plugin and receiving the corresponding message. The process for obtaining this information is straightforward in the case of HTTP POST. When the IoT device transmits data to the server using the REST API service, the response message is promptly returned. However,

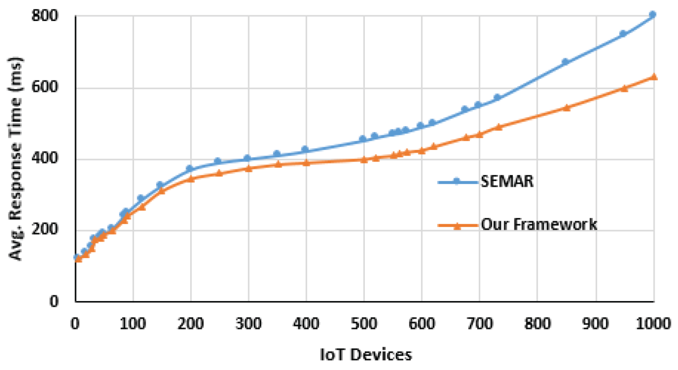

Fig. 7. Average response time.

for MQTT, the program is tailored to measure the response time. This adaptation involves sending the MQTT message to the device when data is stored in the storage. The response time for HTTP and MQTT plugins is illustrated in (2) and (3), respectively.

$$\bar{RT}_{\text{HTTP}} = \frac{1}{N}\sum_{i=1}^{N}(t_{\text{receive}_i} - t_{\text{transmit}_i}) \tag{2}$$

where

t_{transmit_i} is the time of data transmission,

t_{receive_i} is the time of response reception,

N is the total number of instances.

$$\bar{RT}_{\text{MQTT}} = \frac{1}{M}\sum_{j=1}^{M}(t_{\text{store}_j} - t_{\text{transmit}_j}) \tag{3}$$

where

t_{transmit_j} is the MQTT message transmission time,

t_{store_j} is the data storage &amp; MQTT message reception time,

M is the total number of instances.

For 1000 devices, the average response time in our framework is 630 ms, outperforming SEMAR which recorded a average response time of 800 ms. This performance difference is illustrated in Fig. 7. As we increase the number of IoT nodes in both frameworks, our framework outperforms the SEMAR framework, which registered a longer response rate. The trend indicates that as the number of IoT nodes increases, the response rate also rises in both frameworks, with our framework consistently exhibiting shorter response times. The difference in response rate between the frameworks remains negligible for up to 400 IoT devices, but becomes noticeable once the number of IoT devices reaches 500 and beyond.

We continued our experiment and evaluation through a comprehensive comparison between our framework and SEMAR, focusing on average throughput. We ingested our device plugins with requests per second, simulating 1 to 1000 virtual IoT devices or publishers distributed in 13 incremental steps (1, 2, 5, 7, 10, 25, 50, 75, 100, 250, 500, 750, 1000) using JMeter [48] and the MQTT plugin as a load generator [49].

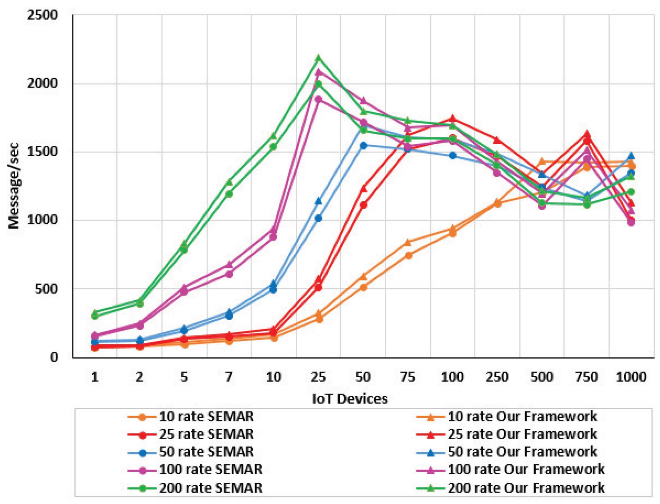

Fig. 8. Throughput ingestion.

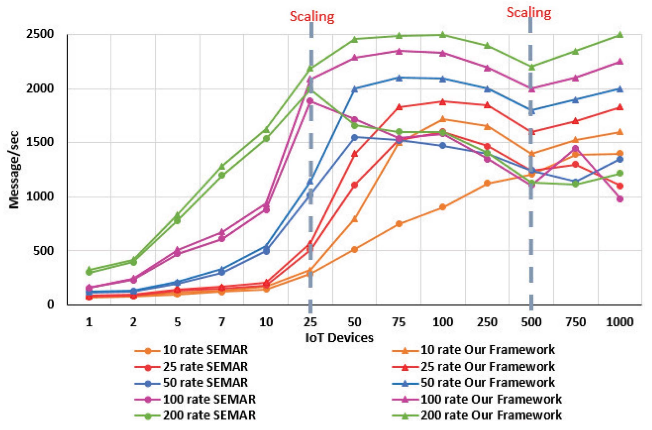

Fig. 9. Throughput ingestion after scaling.

In SEMAR, a maximum average of 1994 messages per second is reached with 25 IoT devices and a rate of 200 messages per second sent from each IoT device. Once the number of IoT devices exceeds 25, there is a decline in the number of messages, with other rates showing a similar trend. This is attributed to the saturation of requests that the framework can handle, competing with the ingestion rate of the server.

We provide the relationship between messages per second, rates, and the number of devices, along with the formula to calculate the throughput, as shown in (4) and (5) to arrive at the figures depicted in Figs. 8 and 9

$$\bar{T} = f(x, y, r) \tag{4}$$

where $T$is the throughput $f(x, y, r)$ is a function that maps the number of devices$>x$messages per second $y$and rates $r$to the throughput.

This relationship can further be expressed mathematically as follows:

$$\bar{T}(x, y, r) = g(x) \cdot y \cdot r \tag{5}$$

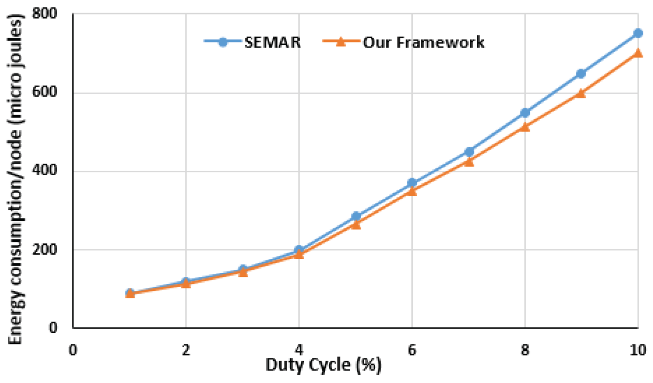

Fig. 10. Energy consumption and duty cycles.

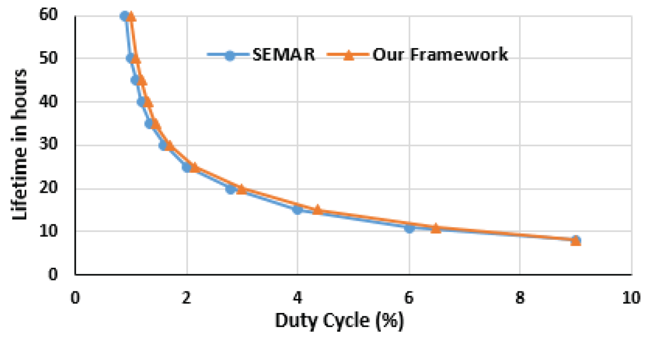

Fig. 11. Edge node life time and duty cycles.

where $T(x, y, r)$is the throughput as a function of the number of devices x messages per second y and rates $rg(x)$ represents a function that describes how throughput varies with the number of devices. For our framework, a maximum average of 2189 messages per second is reached with 25 IoT devices and a rate of 200 messages per second sent from each IoT device. Similar to SEMAR, our device plugins in the containers fail to contain all the requests beyond 25 IoT devices. To address this, we scale up the pods, since we are utilizing an orchestration and containerization approach in Kubernetes. This analogy is illustrated in Fig. 9.

We utilize Kubernetes operations to enhance our system's capacity by leveraging its scalability feature. When surpassing 25 IoT devices, we scale up device plugin pods to manage increasing requests, supporting up to 500 IoT devices and 2500 messages per second at a rate of 200. Scaling continues as needed to handle incoming requests effectively. Our framework demonstrates remarkable scalability, managing a substantial volume of messages and a larger number of devices, surpassing SEMAR. Refer to Fig. 9 for results.

We finally conducted a comparative analysis of our proposed framework with SEMAR, focusing on energy efficiency. Our investigation centered on the correlation between energy consumption and duty cycles, as well as the impact of duty cycles on edge node lifespan. Throughout the energy consumption and duty cycles study, we maintained a consistent number of end-node devices (1000 end nodes) and utilized identical workloads for testing each edge node across both frameworks. We leveraged Kepler, which offers an energy monitoring dashboard which helped us to derive our results.

Considering that a 100% duty cycle implies zero sleep time and consequently high energy consumption, we limited the duty cycle to between 1% and 10% to see the actual impact. Our findings revealed a direct relationship between duty cycle increments and energy consumption in both frameworks. However, our framework exhibited superior energy efficiency compared to SEMAR. This superiority stems from our adoption of a containerized approach to workloads, enabling the starting and stopping of containerized applications without the overhead associated with managing the entire operating system. Consequently, this approach facilitates more efficient resource utilization and enhanced energy efficiency [50], [51]. The results of the explanation are depicted in Fig. 10.

Conversely, the life time of an edge node is contingent upon its overall energy consumption, with higher lifespans correlating inversely with duty cycles. This relationship is visually represented in Fig. 11. Although our framework demonstrates a slight advantage over SEMAR, it remains imperative to minimize duty cycles in both frameworks to achieve optimal power management.

V. Conclusion

This article focuses on extending Kubernetes to the edge for IoT device management, with a focus on automating IoT device registration and application deployment. The goal is to reduce human involvement, enhance interoperability, and streamline management within heterogeneous configurations. Our testing demonstrates the framework's superior performance, efficiency, and scalability compared to manual approaches and other frameworks, even with hundreds of IoT end nodes. Future work should explore implementing predictive machine learning for edge node autoscaling, maintenance, and developing node-based load balancing mechanisms for dispersed IoT setups.

References

[1] U. Breitenbücher, K. Képes, F. Leymann, and M. Wurster, "Declarative vs. imperative: How to model the automated deployment of iot applications?," in *Proc. 11th Adv. summer Sch. Serv. Oriented Comput.*, 2017, pp. 18–27.
[2] S.N. Srirama and S. Basak, "Fog computing out of the box with FOGDEFT framework: A case study," in *Proc. IEEE 15th Int. Conf. Cloud Comput.*, 2022, pp. 342–350.
[3] A. C. Beltrão, B. B. N. de França, and G. H. Travassos, "Performance evaluation of kubernetes as deployment platform for IoT devices," in *Proc. Ibero- Amer. Conf. Softw. Eng.*, 2020, pp. 96–110.
[4] D. C. Y. Vargas and C. E. P. Salvador, "Smart IoT gateway for heterogeneous devices interoperability," *IEEE Latin Amer. Trans.*, vol. 14, no. 8, pp. 3900–3906, Aug. 2016.
[5] N. Pazos, M. Müller, M. Aeberli, and N. Ouerhani, "Connectopen-automatic integration of IoT devices," in *Proc. IEEE 2nd World Forum Internet Things*, 2015, pp. 640–644.
[6] S. N. Srirama, "A decade of research in fog computing: Relevance, challenges, and future directions," *Softw.: Pract. Experience*, vol. 54, no. 1, pp. 3–23, 2024.
[7] K. Kotis and A. Katasonov, "An ontology for the automated deployment of applications in heterogeneous IoT environments," *Semantic Web J.*, 2012.
[8] L. Kiefer, "Concept and implementation of a tosca orchestration engine for edge and IoT infrastructures," Master's thesis, Univ. Stuttgart, Stuttgart, Germany, 2021.
[9] M. R. M. Kassim, "IoT applications in smart agriculture: Issues and challenges," in *Proc. IEEE Conf. Open Syst.*, 2020, pp. 19–24.

[10] B.-C. Chifor, I. Bica, V.-V. Patriciu, and F. Pop, "A security authorization scheme for smart home Internet of Things devices," *Future Gener. Comput. Syst.*, vol. 86, pp. 740–749, 2018.
[11] M. Weqar, S. Mehfuz, and D. Gupta, "Autonomous device discovery for IoT: Challenges and future research directions," in *Internet of Things*, 2023, 1st Ed. Chapman and Hall/CRC, 2023, pp. 257–276.
[12] I. Čilić, P. Krivić, I. Podnar Žarko, and M. Kušek, "Performance evaluation of container orchestration tools in edge computing environments," *Sensors*, vol. 23, no. 8, 2023, Art. no. 4008.
[13] Kubernetes, "Kubernetes: Production-grade container orchestration," Accessed: Sep., 13, 2023. [Online]. Available: https://kubernetes.io/
[14] Docker, "Docker swarm," Accessed: Sep., 13, 2023. [Online]. Available: https://docs.docker.com/engine/swarm/
[15] A. Ullah, H. Dagdeviren, R. C. Ariyattu, J. DesLauriers, T. Kiss, and J. Bowden, "Micado-edge: Towards an application-level orchestrator for the cloud-to-edge computing continuum," *J. Grid Comput.*, vol. 19, pp. 1–28, 2021.
[16] A. R. Madushanki, M. N. Halgamuge, W. S. Wirasagoda, and A. Syed, "Adoption of the Internet of Things (IoT) in agriculture and smart farming towards urban greening: A review," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 4, pp. 11–28, 2019.
[17] S. Basak and S. N. Srirama, "Fog computing out of the box: Dynamic deployment of fog service containers with Tosca," *Int. J. Netw. Manage.*, 2023, Art. no. e2246. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.2246
[18] M. Cruz, S. Mafra, E. Teixeira, and F. Figueiredo, "Smart strawberry farming using edge computing and IoT," *Sensors*, vol. 22, no. 15, 2022, Art. no. 5866.
[19] H. Kokkonen et al., "Autonomy and intelligence in the computing continuum: Challenges, enablers, and future directions for orchestration," 2022, *arXiv:2205.01423*.
[20] C. Resende, "Dot-digital orchestration of things," in *Proc. IEEE Int. Conf. Smart Comput.*, 2023, pp. 247–248.
[21] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad Hoc Netw.*, vol. 7, no. 3, pp. 537–568, 2009.
[22] Y. Maleh, Y. Qasmaoui, K. El Gholami, Y. Sadqi, and S. Mounir, "A comprehensive survey on SDN security: Threats, mitigations, and future directions," *J. Reliable Intell. Environ.*, vol. 9, no. 2, pp. 201–239, 2023.
[23] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "A survey on software-defined wireless sensor networks: Challenges and design requirements," *IEEE Access*, vol. 5, pp. 1872–1899, 2017.
[24] Docker, "Use containers to build, share and run your applications: Package software into standardized units for development, shipment and deployment," Accessed: Sep., 17, 2023. [Online]. Available: https://www.docker.com/resources/what-container/
[25] M. Narasimhulu, D. V. Mounika, P. Varshini, K. Amarendra, and T. R. K. Rao, "Investigating the impact of containerization on the deployment process in devops," in *Proc. IEEE 2nd Int. Conf. Edge Comput. Appl.*, 2023, pp. 679–685.
[26] M. Khan, T. Becker, P. Kuppuudaiyar, and A. C. Elster, "Container-based virtualization for heterogeneous HPC clouds: Insights from the eu H2020 cloudlightning project," in *Proc. IEEE Int. Conf. Cloud Eng.*, 2018, pp. 392–397.
[27] N. Zhou, H. Zhou, and D. Hoppe, "Containerization for high performance computing systems: Survey and prospects," *IEEE Trans. Softw. Eng.*, vol. 49, no. 4, pp. 2722–2740, Apr. 2023.
[28] L. Urblik, E. Kajati, P. Papcun, and I. Zolotova, "A modular framework for data processing at the edge: Design and implementation," *Sensors*, vol. 23, no. 17, 2023, Art. no. 7662.
[29] Redhat, "What is container orchestration," Accessed: Sep., 17, 2023. [Online]. Available: https://www.redhat.com/en/topics/containers/what-is-container-orchestration
[30] T.-G. Kwon and K. Ro, "A study on edge computing-based microservices architecture supporting IoT device management and artificial intelligence inference," in *Proc. IEEE Int. Conf. Electron., Inf., Commun.*, 2023, pp. 1–2.
[31] M. Pradhan and S. Mohanty, "Remote authentication of IoT devices based upon fog computing," in *Proc. Mach. Intell. Techn. Data Anal. Signal Process.: Proc. 4th Int. Conf. MISP*, 2023, vol. 1, pp. 287–299.
[32] A. Tsagkaropoulos, Y. Verginadis, M. Compastié, D. Apostolou, and G. Mentzas, "Extending Tosca for edge and fog deployment support," *Electronics*, vol. 10, no. 6, 2021, Art. no. 737.
[33] M. Wurster, U. Breitenbücher, K. Képes, F. Leymann, and V. Yussupov, "Modeling and automated deployment of serverless applications using tosca," in *Proc. IEEE 11th Conf. Serv.-Oriented Comput. Appl.*, 2018, pp. 73–80.
[34] S. A. Noghabi, J. Kolb, P. Bodik, and E. Cuervo, "Steel: Simplified development and deployment of {Edge-Cloud} applications," in *Proc. 10th USENIX Workshop Hot Topics Cloud Comput.*, 2018, p. 6.
[35] N. Ferry, R. Dautov, and H. Song, "Towards a model-based serverless platform for the cloud-edge-IoT continuum," in *Proc. 22nd IEEE Int. Symp. Cluster, Cloud Internet Comput. (CCGrid)*, 2022, pp. 851–858.
[36] Y. Y. F. Panduman, N. Funabiki, P. Puspitaningayu, M. Kuribayashi, S. Sukaridhoto, and W.-C. Kao, "Design and implementation of semar IoT server platform with applications," *Sensors*, vol. 22, no. 17, 2022, Art. no. 6436.
[37] S. Trilles, A. González-Pérez, and J. Huerta, "An IoT platform based on microservices and serverless paradigms for smart farming purposes," *Sensors*, vol. 20, no. 8, 2020, Art. no. 2418.
[38] KubeEdge, "Kubeedge documentation," Accessed: Sep., 18, 2023. [Online]. Available: https://kubeedge.io/
[39] S.-H. Kim and T. Kim, "Local scheduling in kubeedge-based edge computing environment," *Sensors*, vol. 23, no. 3, 2023, Art. no. 1522.
[40] V.-C. Le and M. Yoo, "Lightweight kubeedge tool for edge computing environments," *J. Korean Soc. Commun. Stud.*, vol. 46, no. 9, pp. 1507–1514, 2021.
[41] Akri, "Akri documentation," Accessed: Sep., 18, 2023. [Online]. Available: https://docs.akri.sh/
[42] Flotta, "Project-flotta framework documentation," Accessed: Sep., 18, 2023. [Online]. Available: https://project-flotta.io/documentation/v0_2_0/intro/overview.html
[43] M. Ayaz, M. Ammad-Uddin, Z. Sharif, A. Mansour, and E.-H. M. Aggoune, "Internet-of-Things (IoT)-based smart agriculture: Toward making the fields talk," *IEEE Access*, vol. 7, pp. 129551–129583, 2019.
[44] S. Navulur et al., "Agricultural management through wireless sensors and Internet of Things," *Int. J. Elect. Comput. Eng.*, vol. 7, no. 6, 2017, Art. no. 3492.
[45] Kepler, "Kepler documentation," Accessed: Jan., 24, 2024. [Online]. Available: https://sustainable-computing.io/
[46] ebpf, "Ebpf documentation," Accessed: Jan., 24, 2024. [Online]. Available: https://ebpf.io/
[47] Kubebuilder, "Kubebuilder documentation," Accessed: Sep. 25, 2023. [Online]. Available: https://book.kubebuilder.io/
[48] JMeter, "Jmeter documentation," Accessed: Sep. 25, 2023. [Online]. Available: https://jmeter.apache.org/
[49] M. Guerra, "Jmeter documentation," Accessed: Sep. 25, 2023. [Online]. Available: https://gist.github.com/marianoguerra/be216a581ef7bc23673f501fdea0e15a
[50] Ö. E. Demirkol and A. Demírkol, "Energy efficiency with an application container," *Turkish J. Elect. Eng. Comput. Sci.*, vol. 26, no. 2, pp. 1129–1139, 2018.
[51] E. Christian, "How containerized applications increase speed & efficiency," Accessed: Sep., 20, 2020. [Online]. Available: https://scoutapm.com/blog/how-containerized-applications-work

Vitumbiko Mafeni received the B.S. degree in information & communication technology from Daeyang University, Lilongwe, Malawi, in 2020. He is currently working toward an integrated M.Sc. and Ph.D. degrees in information telecommunications engineering, with a focus on automation, networking, and computers with Soongsil University, School of Electronic Engineering, Seoul, South Korea. His current focus is on automating the deployment and management of 5G network functions across a multi-cloud, multi-cluster setup.

He is also a member of Distributed Cloud & Network(DCN) Laboratory at Internet Infra System Research Center - SSU IISTRC in Seoul, South Korea. His primary research interests include intelligent communication System, IoT, and edge-cloud computing.

Younghan Kim (Member) received the B.S. degree in electronics engineering from Seoul National University, Seoul, South Korea, in 1984, and the M.S. and Ph.D. degrees in electrical engineering from Korea Advanced Institute of Science and Technology, Seoul, South Korea, in 1986 and 1990, respectively.

Since 1994, he has been with Soongsil University, where he is currently a Professor with the school of electronic engineering.