Received 25 December 2020; accepted 28 January 2021. Date of publication 1 February 2021;
date of current version 19 February 2021. The review of this paper was arranged by Associate Editor Stamatis Karnouskos.

Digital Object Identifier 10.1109/OJIES.2021.3055901

Performance Evaluation of Containerization in Edge-Cloud Computing Stacks for Industrial Applications: A Client Perspective

YU LIU [1], DAPENG LAN [2], ZHIBO PANG [3] (Senior Member, IEEE), MAGNUS KARLSSON [1] (Member, IEEE), AND SHAOFANG GONG [1] (Member, IEEE)

[1]Department of Science and Technology, Linköping University, 60221 Norrköping, Sweden
[2]Department of Informatics, University of Oslo, NO-0316 Oslo, Norway
[3]ABB AB, Corporate Research, Forskargränd 7, 72178 Västerås, Västmanland, Sweden

CORRESPONDING AUTHOR: YU LIU (e-mail: yu.a.liu@liu.se).

This work was supported by the financial grant by Swedish Innovation Agency, Vinnova.

ABSTRACT Today, the edge-cloud computing paradigm starts to gain increasing popularity, aiming to enable short latency, fast decision-making and intelligence at the network edge, especially for industrial applications. The container-based virtualization technology has been put on the roadmap by the industry to implement edge-cloud computing infrastructures. Has the performance of the container-based edge-cloud computing stacks reached industry requirement? In this paper, from the industrial client perspective, we provide a performance evaluation methodology and apply it to the state-of-the-art containerization-based edge-cloud computing infrastructures. The influences of the message sending interval, payload, network bandwidth and concurrent devices on full stack latency are measured, and the processing capability of executing machine learning tasks are benchmarked. The results show that containerization on the edge does not introduce noticeable performance degradation in terms of communication, computing and intelligence capabilities, making it a promising technology for the edge-cloud computing paradigm. However, there is a large room for performance improvement between current implementation of the edge-cloud infrastructure and the demanding requirements anticipated by time-critical industrial applications. We also emphasize and showcase that partitioning of an industrial application into microservices throughout the whole stack can be considered during solution design. The proposed evaluation methodology can be a reference to users of edge-cloud computing as well as developers to get a client perspective overview of system performance.

INDEX TERMS Performance evaluation, edge-cloud computing, containerization, partitioning.

I. INTRODUCTION

Traditional centralized cloud-enabled Internet of Things (IoT) architecture have encountered huge challenges after many years rapid expansion, e.g., the heterogeneity of hardware, software, communication protocols and data format [1], and especially the demanding low latency and high reliability that are required by time-critical applications in cyber-physical systems (CPS) in industrial IoT (IIoT). In recent years, the edge-cloud computing paradigm, as a result of the technological evolution in cloud computing, has been proposed to resolve the challenges and facilitate the adoption of IoT innovations into various domains, as shown in Fig. 1.

With distributed edge computing units, a part of the functionalities of the cloud platform is relocated to the edge of field networks, leading to boost of performance in communication, computing and intelligence. First, edge computing units enable low latency and fast response while eliminate the heterogeneity challenge by providing rich communication interfaces. Second, tasks used to be computed in the cloud are extended to the edge units so as to accelerate decision-making at the edge. Third, deployment of machine learning/deep learning models to the edge moves artificial intelligence closer to user scenarios and fosters novel machine learning techniques such as federated learning [2].

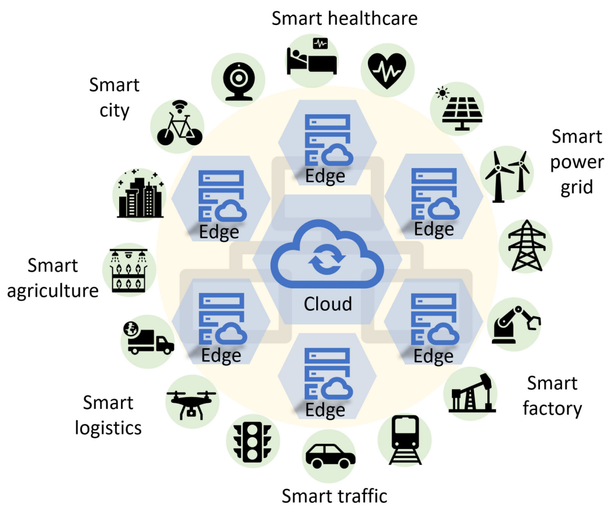

FIGURE 1. An overview of the edge-cloud computing paradigm used in different scenarios, e.g, smart city, agriculture, logistics, traffics, factory, power grid, and healthcare. Edge devices are connected to the cloud platform via Ethernet while the communications between field and edge devices are through heterogeneous wired or wireless protocols. (A part of the icons are created by Adrien Coquet, Misbahul Munir, Chameleon Design, SBTS and Royyan Razka from the Noun Project. [3]).

Furthermore, edge-cloud computing reduces the traffic towards the cloud and therefore features enhanced security and privacy.

The widespread deployment nature of edge computing units calls for a flexible framework that enables continuous integration/continuous deployment (CI/CD) at scale, regardless of the heterogeneity in hardware and software. In this regard, microservice and Docker container-based virtualization become an indispensable approach [4]–[7]. Microservices split a traditional monolithic application into smaller task modules with little dependency on each other while the lightweight and small footprint characteristics of the container technology make it an optimal carrier for microservices. Therefore, container-based virtualization is a key enabler to the success of edge computing, especially in the approaching 5G era, which has been a consensus of hyper-scale cloud (HSC) and network infrastructure suppliers [8] that are keen to bring containerization into edge computing.

However, can the state-of-the-art edge-cloud computing stacks fulfill the industry requirement? How much room of the performance shall be improved to reach the anticipation? These questions are still unclear. The heterogeneous industrial sectors and end applications of IIoT, i.e., process automation, industrial control, power system automation, building automation, and automobile, etc., have resulted in rich invention of industrial connectivity protocols [9]. The diverse connectivity between industrial field devices and edge infrastructure, depending on the communication medium and protocols from the media access control (MAC) layer all the way toward the application layer as defined in the open systems interconnection (OSI) model, leads to widely divergent performance. Many researchers have provided thorough study on the performance of existing industrial connectivity protocols from different perspectives, such as IEEE 802.11 [10], [11], IEEE 802.15.4 [12], [13], ISA100.11a [14], [15], WIA-PA [16], [17], wirelessHART [18], [19], Thread [20], [21], and LoRaWAN [22]–[24], whereas the performance of the backbone, i.e., the edge-cloud computing stack, is still unknown.

Therefore, in this study, we limit our scope to the performance of containerization-based edge-cloud computing stacks in IIoT applications. Specifically, we focus on the procedure that starts from a field device message arriving the edge unit and ends with the same message being received after it is serviced by either edge or cloud. From a client perspective, communication, computing, and intelligence capabilities at the edge and between the edge and the cloud platforms should be considered to provide a comprehensive understanding of the holistic performance of the edge-cloud computing stacks.

The main contributions of this study are as follows.

- Performed a client perspective evaluation of the holistic system latency performance of containerization, namely full stack round trip time with regard to message sending interval, payload, network bandwidth and concurrent devices, on the state-of-the-art edge-cloud computing platforms.
- Evaluated the processing capability of containerization in edge-cloud computing stacks, in terms of executing machine learning and neural network tasks for real industrial applications.
- Profiled the performance limits and highlighted the necessity of partitioning in currently available edge-cloud computing infrastructures in industrial practice.
- Showcased the methodology of applying partitioning to an industrial application, i.e, a vertical plant wall system, based on the performance evaluation results, to verify the feasibility and to promote the methodology.

The remainder of the paper is organized as follows. In Section II, the background of edge-cloud computing stacks are provided. In Section III, the virtualization-based system performance evaluations of edge-cloud stacks in the literature are reviewed. Sections IV and V present the evaluation methodology and detailed experiment setup, respectively. In Section VI, the evaluation results are analyzed and the necessity for partitioning is discussed. In Section VII, a case study about partitioning in a real-world industrial application using the container-based edge-cloud computing paradigm is presented. Section VIII discusses the meaning of the proposed evaluation methodology, results, and future work. The last section concludes the paper.

II. EDGE-CLOUD COMPUTING STACK

In this section, the architecture and typical deployment models of the state-of-the-art edge-cloud computing stack are introduced as background information.

A. ARCHITECTURE

Fig. 2 illustrates a containerization-based edge-cloud computing architecture. In the edge unit, industry applications are modularized as independent containers that are managed by

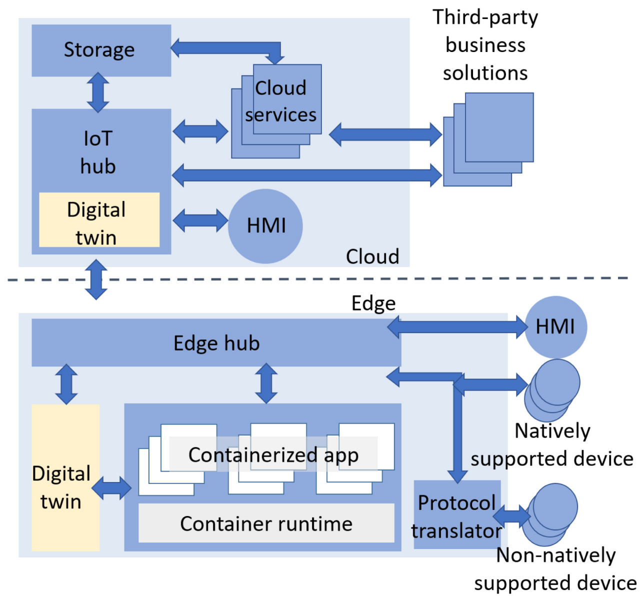

FIGURE 2. Architecture breakdown of the edge-cloud computing paradigm for containerization-based industrial IoT applications.

a container runtime. These containerized applications can be remotely created, upgraded and destroyed in an elastic manner with low overhead while the procedure can be fully managed with container orchestration tools such as Kubernetes [25]. An edge hub infrastructure takes responsibility to maintain traffic flows taking place in the edge, including bidirectional communication between field devices and the cloud platform, and message routing among containerized applications, which is similar to the role that an IoT hub plays in the cloud. Field IoT devices can get served by edge applications by establishing communications to the edge hub, either using natively supported protocols such as the message queuing telemetry transport (MQTT) protocol or through a protocol translator. This design has become the de facto edge-cloud computing architecture agreed and put into practice by the industry, e.g., Microsoft Azure IoT Edge [4], Amazon Greengrass [5], IBM Edge Application Manager [6], and Huawei KubeEdge [7], and is envisioned to be leveraged by industry artificial intelligence and IoT (AIoT) applications [26].

B. EDGE-CLOUD DEPLOYMENT MODELS

With the edge-cloud computing stack, three typical models that describe the connectivity between the field devices and the edge and cloud platforms can be utilized to deploy applications.

1) DEVICE-CLOUD MODEL (*D-C*)

The device-cloud model enables a field device to establish a direct connection to the cloud without getting traffic passing through the edge infrastructure. The field device only takes advantage of the networking capability of the edge device where Internet is accessed. Communication between the field device and the cloud services is straightforward within an established session. This model originates from the cloud computing paradigm in which the edge device only functions as a router.

2) DEVICE-EDGE-CLOUD MODEL (*D-E-C*)

The device-edge-cloud model inherits from the device-cloud model but provides extra reliability to the applications. Traffics from the field devices are firstly aggregated to the edge hub and then forwarded to the cloud with a multiplexed communication channel so as to save bandwidth. In case of intermittent loss of connection to the cloud, all the field device traffics can temporarily be held at the edge unit till the recovery of the connection to the cloud.

3) DEVICE-EDGE MODEL (*D-E*)

The device-edge model shifts more responsibilities to the edge device, i.e., both communication and computing take place at the edge device. The model ensures field devices can directly establish communication channels to and get feedback from edge-native services, which aims at fast response and low latency for time-critical applications or scenarios where the cloud is not reachable.

III. RELATED WORK

Many researches have been conducted to explore the performance of containerization in the edge-cloud computing stacks, from networking layer, application layer or resource utilization perspective, by utilizing one or more of the device-cloud, device-edge-cloud, and device-edge deployment models. For instance, some studies have investigated the networking performance of the container technology. In [27], the authors measured the performance of three network solutions specifically for managing container communication, namely Flannel, Swarm Overlay, and Calico. The latency of network layer and the throughput of two transportation layer protocols, i.e., TCP and UDP, were studied. In [28], the authors investigated the latency performance of Docker container that functioned as a HTTP proxy and highlighted its advantage to the virtual machine (VM) based solutions. The authors of [29] presented a edge-cloud computing architecture utilizing the Docker container technology. The publish-subscribe pattern based latency was measured to evaluate the performance of containerized microservices migrated between cloud and the edge devices. The results show that containerized microservice can satisfy ultra-low latency required by novel applications. The authors in [30] researched the capability of container-based virtualization used in time sensitive applications in industrial automation. Three criteria, namely round trip time, CPU time delay, and time distribution were considered to assess the system performance of containerization. In [31], the authors looked into the latency performance of multi-access edge computing platform running a container-based user mobility analysis service and concluded that the low latency promise in the edge could be guaranteed with a containerized implementation. The authors in [32] provided a thorough performance benchmarking of stream data latency, scalability and resource utilization on a container-based edge computing system. Three scenarios, edge-only, cloud-only and combined edge-cloud, were benchmarked on a self-implemented testbed using open source frameworks running as containers.

Plenty of studies narrow their view to application layer system performance in microservice-based cloud and edge computing. In [33], the authors introduced a container-based function as a service (FaaS) model for the edge environment. The model was benchmarked on both resource constrained single board computer and high performance machine. The system performance was reflected by CoAP and HTTPS based latency metric. The authors in [34] presented an assessment of a containerized MQTT broker cluster. Throughput, end-to-end latency and system resource consumption of runtime were benchmarked, and an average latency of less than 10 ms was achieved. In [35], the authors made a comparison of full-cloud and edge-cloud architectures in respect to time delay and bandwidth usage. An anomaly detection task was utilized to mimic IIoT use case. A containerized edge gateway and a virtual machine based cloud were selected to perform data processing, whereas MQTT was selected as the protocol to deliver messages.

A lot of work focus on the processing capability of container running on edge platforms. In [36] the authors evaluated the container performance from a hardware perspective including CPU, memory, disk and network utilization, and drew a conclusion that the performance of containerization was comparable to the host OS. The authors in [37] applied the same metrics to a comparative study between the container and the hypervisor virtualization deployed at the network edge device. In [38] the author performed a comprehensive study on the performance of containers running on five single board computers that were popularly used as edge gateway platforms, with consideration of metrics such as CPU execution time and load, memory speed, disk I/O and network throughput, as well as power consumption. Similarly, in [39] the authors utilized containers to run microservices on an IoT edge gateway with WiFi, radio and satellite interfaces, and evaluated the container performance with aforementioned metrics as well as system boot-up time. The authors in [40] focused on evaluating container hardware utilization and processing capabilities based on different runtime technologies. In [41] the authors inspected the performance of containerized deep learning applications running on an edge platform in a smart home environment. Specifically, CPU load, execution time and network traffic were studied and compared to native applications.

There are also studies considering the interference of concurrently executed container instances that are running in an edge-cloud environment. In [42], the authors evaluated interference of microservices that were carried out within a single container and separate containers, and the CPU load and the throughput of memory and network were used for metrics. The authors in [43] utilized handling rate and connection time to compare the concurrent performance between edge container and cloud container-based server, whereas in [32] the author performed similar tests but used latency as the rating criterion.

Table I gives a summary of the review studies, including deployment models, measured performance, adopted metrics, and experiment environment that are used in the test. It can be observed that existing studies have evaluated the performance of containerization-based edge-cloud computing stacks with a focus on a specific aspect or layer, e.g., latency introduced by virtualization, interference between containers, and hardware utilization. These approaches are valuable to identify bottlenecks existed in a specific layer. However, from an industrial client perspective, it is equivalently significant to have a holistic evaluation that penetrates the full stack of the edge-cloud computing infrastructures, so as to provide a complete picture of the system performance. Besides, benchmarking with a real industrial application and platform is essential to reproduce real-world edge-cloud computing scenarios, which can greatly validate the results and bring meaningful reference to industrial production development. Therefore, in this study, we aim to probe the holistic performance of containerization-based edge-cloud computing paradigm, utilizing the cutting-edge infrastructures available to the industry.

IV. EVALUATION METHODOLOGY

This section introduces the evaluation methodology that is applied throughout the study.

A. EVALUATION CRITERIA

A mature IIoT solution shall comprehensively consider all aspects of communication, computing and intelligence capabilities, as the performance improvement in one layer can be mitigated by performance loss in other layers, which can lead to system performance degradation. In order to provide a holistic vision of the edge-cloud computing paradigm, the following criteria are used in this study.

1) FULL STACK ROUND TRIP TIME (RTT)

The time latency metric measures how fast a field application can send update to and get a result from service applications in an edge-cloud architecture. In an IIoT application, the time latency can be introduced by various aspects such as the application layer implementation, network stack, propagation delay, and processing time. Therefore, we use full stack RTT to characterize the latency feature for a typical edge-cloud architecture. The evaluation of a specific layer can help identify the bottleneck of the system, however, from a client perspective, the full stack RTT is more meaningful to depict the holistic performance of the adopted edge-cloud architecture, thus provides insightful knowledge to the solution design. Fig. 3 illustrates the measurement of the full stack RTT in the aforementioned three deployment models in Section II.B. It counts from an application layer message getting transmitted till the corresponding response message

TABLE 1 A Summary of the Related Work, Including the Deployment Models, Measured Performance, Tested Metrics, and Experiment Environment Used in the Studies

Work	Deployment models	Measured performance	Metrics	Experiment environment
[27]	D-E or D-C	container network throughput	Ping delay, TCP/UDP throughput	local testbed
[28]	D-E-C	latency in network virtualization	response time	local testbed
[29]	D-C, D-E	end-to-end service latency	message transmission time	local testbed
[30]	D-E	latency of Dockerized software	UDP round trip time	local testbed
[31]	D-E	service request handling latency	service response time	local testbed with industrial application
[32]	D-C, D-E, D-E-C	latency, scalability, resource utilization	round trip time, CPU/memory/network I/O utilization	local testbed
[33]	D-E	latency, concurrency	CoAP/HTTPS based service execution time	local testbed
[34]	D-E-C	throughput, latency, resource consumption	end-to-end transmission time, MQTT message throughput, CPU/memory utilization	local testbed
[35]	D-E-C, D-C	latency, bandwidth usage	end-to-end delay, network load	industrial platform
[36]–[41]	D-E	resource utilization	CPU/memory/disk/network utilization, power consumption, boot-up time	local testbed
[42]	D-E or D-C	concurrency	CPU/memory/disk/network utilization	local testbed
[43]	D-E-C	concurrency	handling rate, connection time	industrial platform

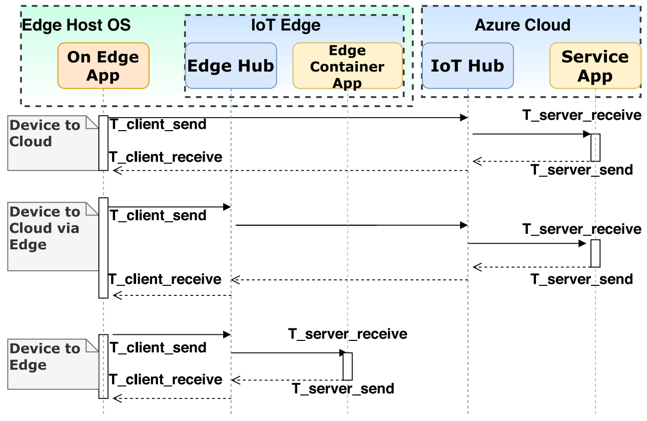

FIGURE 3. The data flow of three benchmarking cases. (1, 2) Device-cloud/device-edge-cloud: an application directly connecting to the cloud / connecting to the cloud via the Edge hub infrastructure, and getting a response from a service application hosted in the cloud. (3) An application connecting to the Edge hub and getting a response from a containerized application hosted at the edge.

arriving the application layer, which reflects the accumulated latency throughout the full stack of an edge-cloud architecture. With full stack RTT as the metric, several parameters that can influence the system performance are benchmarked, i.e., the **message sending interval**, the **payload size**, the **network bandwidth** and the **amount of concurrent devices**. These parameters are tweaked to evaluate the performance boundary of different deployment models.

The full stack RTT values are evaluated according to two factors, namely time sensitivity and determinism [44]. Time sensitivity requires the latency of critical industrial applications to meet their deadlines while determinism guarantees the latency is predictable, i.e., the jitter shall be considered and constricted.

2) PROCESSING CAPABILITY

In an industrial AIoT solution, computing and intelligence services such as data analytics, machine learning based predictive maintenance and so on, are more related to creating business values. This criterion is proposed to measure computing and intelligence capabilities of the edge-cloud computing architecture in order to explore the system limitation and to serve as a guideline in AIoT solution design. In the benchmarking, training and prediction tasks for machine learning and neural networks models are adopted for stress test while the **execution time**, **CPU load** and **memory utilization** are recorded for comparison. In this way, the computing performance in terms of providing intelligence tasks is measured.

V. EXPERIMENT SETUP

A. APPLICATION LAYER PROTOCOL

With the proliferation of next generation telecommunication technologies like 5G and WiFi 6, the application layer communication protocol becomes the bottleneck in IoT practice. An appropriate selection of the application layer protocol can be significant to the benchmarking. Standards such as OPC UA over Time sensitive network (TSN) [45] and data distribution service (DDS) [46] have been promoted by the industry to push edge computing into time-critical domain. But they are still not full-fledged and lack of seamless integration to existing cloud infrastructures. The study [47] surveyed a series of popular IoT messaging protocols, e.g., MQTT, the hypertext transfer protocol (HTTP), the advanced message queuing protocol (AMQP), the constrained application protocol (CoAP) and the extensible messaging and presence protocol (XMPP). The results show that the MQTT protocol is the only protocol that has been natively integrated to a vast majority of edge-cloud computing platforms such as Microsoft, Amazon, Google, Alibaba and IBM. Furthermore, the work done in [48], [49] show that MQTT ourperforms HTTP, AMQP and XMPP in terms of latency performance. Therefore, in this experiment, the MQTT protocol is selected as the application layer protocol.

B. CONTAINERIZED EDGE INFRASTRUCTURE

In this study, Microsoft Azure IoT Edge is utilized as the edge infrastructure. Azure IoT Edge offers a container-based edge computing framework that enables customized services to be deployed to the edge as microservices. An edge agent module is responsible for instantiating containerized applications while an edge hub module manages the message communication and routing. The Raspberry Pi 3B+ board is chosen as the hardware platform which features a Broadcom BCM2837B0 SoC running at 1.4 GHz, a 1 GB SDRAM and an Ethernet port with the maximum throughput of 300 Mbps. The edge platform is directly connected to the Internet through the Ethernet port at Linköping University, Sweden. According to measurement, the bandwidth of downlink and uplink are 236.82 and 275.76 Mb/s at the edge Ethernet port. The edge platform runs the Raspbian 9.11 (Stretch) operating system which is based on Linux kernel 4.19. In all the experiments, the applications are run with the default Linux time-sharing scheduling policy namely SCHED_OTHER to keep consistency.

C. CLOUD INFRASTRUCTURE

An IoT Hub service and a virtual machine (VM) are deployed to the Azure cloud to provide message ingestion and to run service applications, respectively. IoT Hub has built-in MQTT broker capability and is able to route messages between IoT or edge devices and service applications. An S1-standard tier of the IoT Hub service is used, which enables 400 K messages per day and a theoretical throughput of 100 messages per second while the message size can be up to 256 KB. The VM features four Intel Xeon E5-2673 CPUs running at 2.40 GHz and a 16 GB RAM. The services are deployed to the same data center, i.e., the Azure North Europe data centre, which is close to where the edge platform deployed.

D. EXPERIMENT IMPLEMENTATION

1) EXPERIMENT 1 - FULL STACK ROUND TRIP TIME TEST

In this experiment, the full stack RTT is tested under three edge-cloud models (*D-C, D-E-C* and *D-E*). A client application written in Python is deployed to the edge device, running on the host operating system, as shown in Fig. 3. With *D-C* model, the client application initializes a connection to IoT Hub. It periodically sends messages with sequence numbers to IoT Hub using the MQTT protocol with QoS set to 1. A containerized service application running in the VM listens to the incoming messages through IoT Hub. The messages are processed by the service application and a direct method is triggered and sent back to the client via IoT Hub. The *D-E-C* model setup is similar to the *D-C* model, except that the client application is configured to connect to IoT Hub through the Edge Hub module running as a container in IoT Edge. In *D-E* model, a containerized application is deployed to IoT Edge as a service module. Messages from the client application are routed to the service module where a direct method towards the client is triggered. In all three models, four timestamps, i.e., $T_{ClientSend}$, $T_{ServerReceive}$, $T_{ServerSend}$, and $T_{ClientReceive}$, are recorded, as marked in Fig. 3. The full stack RTT is calculated as follows.

$$\begin{aligned} full\,stack\,RTT &= T_{ClientReceive} - T_{ClientSend} - T_{Server} \\ T_{Server} &= T_{ServerSend} - T_{ServerReceive} \end{aligned} \tag{1}$$

Four parameters, i.e., the message sending interval, message payload, network bandwidth and number of concurrent devices, are adjusted to evaluate their influences on the full stack RTT metric with regard to aforementioned three edge-cloud models. The detailed parameter configuration is shown in Table II.

TABLE 2 **Parameter Configuration for Experiment 1**

Scenario	**Message sending interval**	**Payload**	**Bandwidth**	**Node number**
1	**variable**	message number + device ID	unlimited	1
2	200 ms	message number + device ID + **variable** bytes payload	unlimited	1
3	200 ms	message number + device ID	**variable**	1
4	200 ms	message number + device ID	unlimited	**variable**

2) EXPERIMENT 2 - PROCESSING CAPABILITY TEST

In this experiment, three machine learning models extracted from this study [50], which are used to detect anomalies in an indoor environment so as to realize predictive maintenance with an active plant wall system, are utilized for the benchmarking. Specifically, a regression model is implemented using the Scikit-learn library, an autoencoder and a long short-term memory-encoder decoder (LSTM-ED) neural network model are implemented with Tensorflow. For model implementation details refer to [50].

These models are deployed to three environments, the edge host OS, a container on the edge, and the VM in the cloud. Models execution time, CPU load and memory usage of both training and prediction procedures are measured with the Linux Sysstat toolbox.

Two experiments are conducted separately to evaluate the communication and processing capabilities so as to guarantee the evaluation results are not interfered by each other. The separation also helps to identify the performance limitation for each aspect. The benchmarked containerization environment is consistent in both experiments, as the container runtime adopted by Azure IoT Edge is based on Moby Docker, which is the upstream project of Docker.

VI. EXPERIMENTAL RESULTS

In this section, the results of the benchmarking are presented and discussed.

A. RESULTS OF EXPERIMENT 1

1) MESSAGE SENDING INTERVAL

Fig. 4 shows the results of full stack RTT with regard to different message sending intervals ranging from 1 ms to 5 seconds. Fig. 4(a)–4(c) are the full stack RTT distributions in the *D-C*, *D-E-C* and *D-E* models. As a baseline, the full stack RTT benchmarked with the native MQTT protocol (implemented with Eclipse Paho) and with the raw socket are shown in Fig. 4(d) and 4(e). Fig. 4(f)–4(j) show the full stack RTT variations with regard to the incremental message sequence number in the situations of Fig. 4(a)–4(e).

In Fig. 4(a)–4(c), it can be observed that the full stack RTTs are deterministic when message sending intervals are above 200 ms, i.e., the mean, median, and value ranges do not vary a lot, which guarantees determinism or predictability to applications.

The full stack RTTs tend to increase drastically as the sending interval goes smaller. In the *D-C* model, more outliers can be seen when the sending intervals are set to 50 and 100 ms while the majority RTT values are still bounded within 180 ms. When the sending interval goes down to 10 ms, the RTT experiences a dramatic growth. This can be verified in Fig. 4(f) that the full stack RTT slightly grows for each message as time goes on. In the *D-E-C* and *D-E* models, the RTT performances already go worse when the sending intervals are smaller than 100 ms, which can also be verified in Fig. 4(a) and 4(h) that the RTTs linearly increase when the sending intervals are below 100 ms. Therefore, the drastic increase in the three models are in accordance with the phenomenon of incremental full stack RTTs shown in Fig. 4(f)–4(h), which results from the buffering mechanism implemented by the Azure edge-cloud infrastructure to guarantee that the messages are delivered instead of being dropped. However, the increasing phenomenon cannot continue forever since the buffer size is limited. It also suggests that with current edge-cloud infrastructure, a message sending interval of 200 ms has reached the performance limit to promise a deterministic performance in all three edge-cloud models.

Comparing the deterministic performance zones of Fig. 4(a)–4(c), the average RTT for *D-C* and *D-E-C* models are around 114 ms and 136 ms, suggesting that sending traffic through the edge hub results in additional 22 ms delay. For the *D-E* model, the average RTT increases from 123 ms to 179 ms as the sending interval increases from 200 ms to 5 s. This phenomenon can result from the operating system context switch, considering both the container and the host OS are not enabled with real-time features. It also suggests in our benchmarking platform, a containerized service application deployed to the edge device cannot offer better latency performance than deployed to the cloud. However, the edge infrastructure brings stability to the network communication by exploiting multiplexing traffics within a single communication channel. For instance, Fig. 4(g) and 4(h) show more linearity and smoothness than Fig. 4(f).

Fig. 4(d) shows the results of benchmarking with the Paho MQTT [51] implementation and the Mosquitto broker [52]. Both the broker and a service application are containerized and deployed to the same edge platform. In the figure, a full stack RTT of 5.7 ms is achieved when the sending interval is set to 100 ms. The performance decreases when the interval is shortened from 50 ms to 1 ms, but in the worst case the RTT is still bounded below 80 ms, which can be clearly seen from Fig. 4(i). The results suggest that a containerized MQTT server is able to provide demanding latency performance with message sending interval below 100 ms.

Benchmarking results in Fig. 4(e) and 4(j) show the full stack RTT of raw socket implementation. It exhibits that a containerized TCP/IP server deployed to the edge device is able to provide superior latency performance, i.e., 543 us average RTT is achieved with 1 ms sending interval. The fluctuation in and variations between different curves observed in Fig. 4(j) are believed to be attributed to the TCP congestion-avoidance algorithm, but need further investigation with dedicated experiments in the future.

In summary of Fig. 4, with Azure IoT Edge and IoT Hub (S1-standard tier) infrastructure, a message sending interval above 200 ms can bring deterministic full stack RTT performance, which is much under the promised performance (100 messages/second). Also, the latency performance cannot benefit from the container-based IoT Edge infrastructure. It is the additional considerations such as authentication, encryption, routing mechanism and management components that must be added to the industrial edge implementation, that greatly worsen the latency performance.

2) MESSAGE PAYLOAD

The influences of the message payload to the full stack RTT in the *D-C*, *D-E-C*, and *D-E* models are exhibited in Fig. 5(a)–5(c). Specifically, a baseline implemented using Paho MQTT is benchmarked and shown in Fig. 5(d). It can be observed in the four graphs that messages with a payload size up to 10 kilobytes do not show clear difference in terms of latency while the RTT starts to increase when the payload size is set to 100 and 250 kilobytes, which indicates that the MQTT protocol can guarantee an optimal performance with the payload size under 10 kilobytes.

Although the full stack RTT experiences a growth when the message payload is above 100 kilobytes in the aforementioned four conditions, the growth tendencies vary a lot. In the case of payload size set to 250 kilobytes, the average RTTs increase by 1.67, 4.26, 1.96 and 2.24 times, respectively. This is due to that the cloud platform is equipped with more powerful

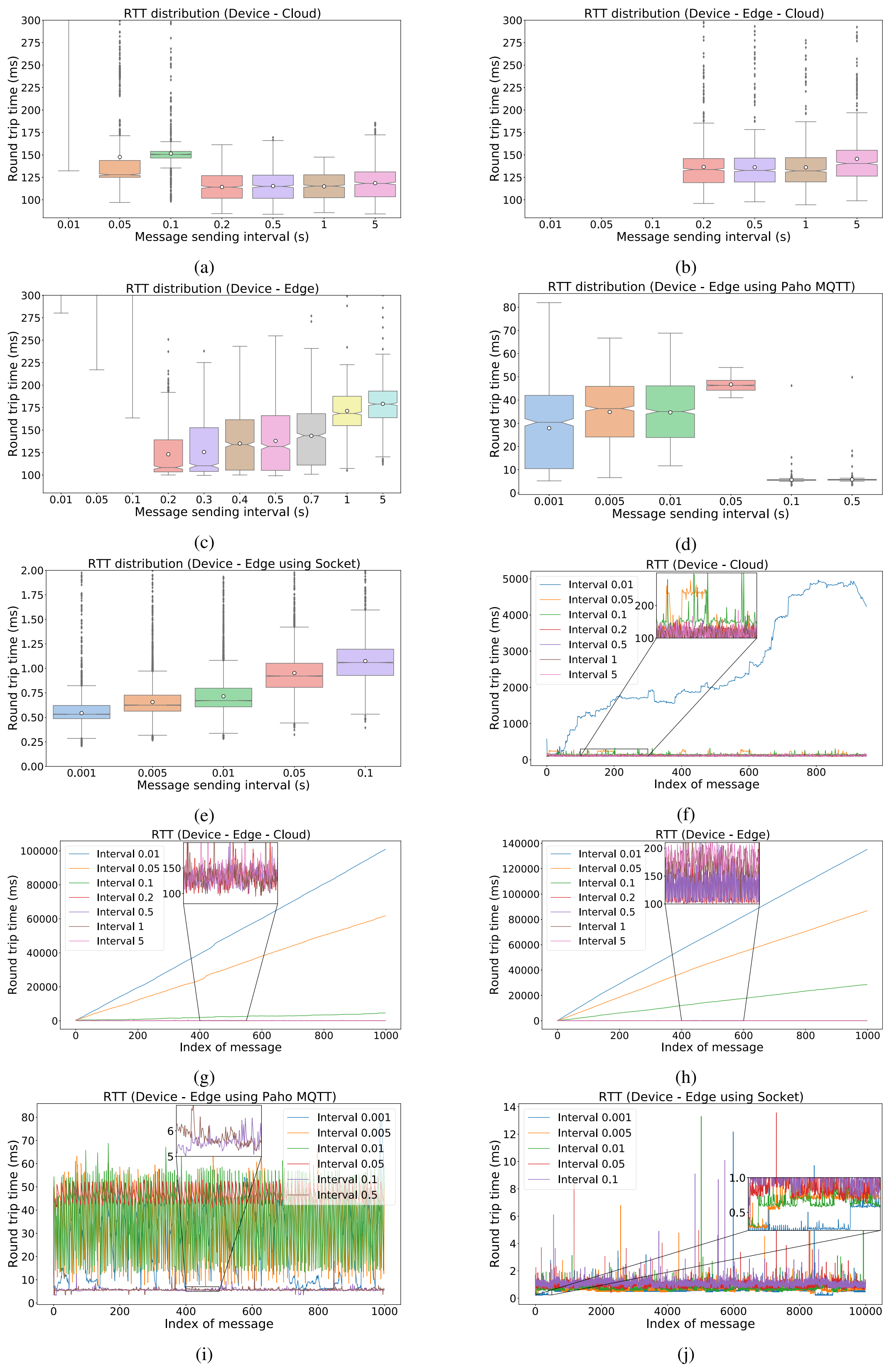

FIGURE 4. **Sub-figure (a)–(c): Full stack round trip time with regard to message sending interval in *D-C/D-E-C/D-E* models. Sub-figure (d)–(e): Full stack round trip time with regard to message sending interval using the Paho MQTT/raw Socket implementations in *D-E* model. In the boxplots, the white circle marks the average, the notch line marks the median, and the black dots are outliers. Sub-figure (f)–(j): Full stack round trip time changes with regard to message index, corresponding to the benchmarkings of subfigure (a)–(e), respectively.**

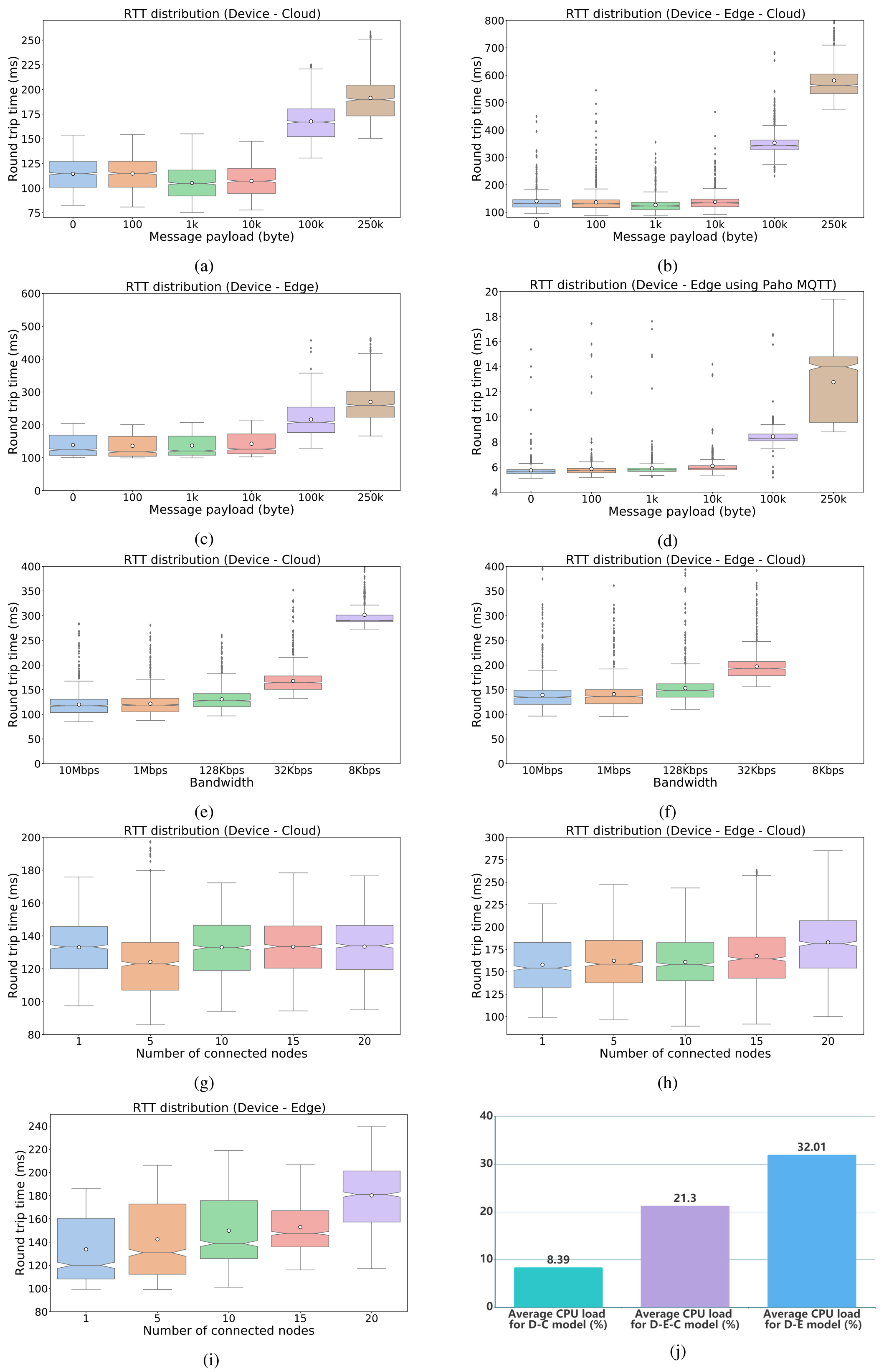

FIGURE 5. **Sub-figure (a)–(c): Full stack round trip time with regard to message payload in *D-C/D-E-C/D-E* models. Sub-figure (d): Full stack round trip time with regard to message payload using the Paho MQTT implementation in *D-E* model. In the boxplots, the white circle marks the average, the notch line marks the median, and the black dots are outliers. Sub-figure (e)–(f): Full stack round trip time with regard to bandwidth. Sub-figure (g)–(i): Full stack round trip time with regard to the number of concurrent nodes. Sub-figure (j): CPU utilization between the D-C, D-E-C, and D-E models in the concurrency test.**

machines that can handle larger messages while the resource constrained edge platform has reached the bottleneck. Besides, the performance gap between the containerized Paho MQTT server deployed to the edge and the Azure IoT Hub/IoT Edge infrastructures is still obvious. Under heavy message payload, the RTT of the Paho MQTT test case is bounded under 20 ms while the worst cases of Azure IoT Hub/IoT Edge infrastructures have been above 250, 700 and 400 ms, respectively.

3) NETWORK BANDWIDTH

Fig. 5(e) and 5(e) show the influence of network bandwidth on the full stack RTT metric. The RTT gradually gets increased as the bandwidth is narrowed but not in a proportional manner. For instance, when the bandwidth is declined by 78 times from 10 Mbps to 128 Kbps, the RTT performance only downgrade by 13% and 10% in the *D-C* and *D-E-C* models. An obvious performance loss appears when the bandwidth is limited to 32 Kbps and goes extremely worse when the bandwidth is constrained to 8 Kbps. In the 8 Kbps case, the average RTT for *D-C* model is slowed down to 301 ms while the boxplot for the *D-E-C* model cannot be seen in Fig. 5(f) due to the large latency.

In general, considering the fact that nowadays Internet infrastructure can easily reach 100 Mbps or above, the network bandwidth should not be a factor to impact the performance of edge-cloud computing.

4) NUMBER OF CONCURRENT DEVICES

The impact of concurrently connected devices are measured in Fig. 5(g)–5(i), corresponding to the *D-C*, *D-E-C*, and *D-C* models. Apart from the client application, the number of other concurrent nodes is increased from 1 to 20. In the case of the *D-C* model, concurrent connections to the cloud do not result in any performance degradation to the system and the full stack RTT distributions are quite close. In the *D-E-C* model, the average RTTs are relatively close when the number of concurrent nodes is below 15, but show a clear jump when the number of nodes increases to 20. Additionally, the up bounds of the RTT tend to gradually increase from 225 to 280 ms, showing a performance decrease. In the *D-E* model, the quality of service degrades quickly as the concurrent nodes are added from 1 to 20, leading to a 35% loss of RTT performance from 133 to 180 ms.

The downgrade of performance in the *D-E-C* and *D-E* models is due to the limited computing resource in an edge device. The CPU usage comparison is shown in Fig. 5(j), which clearly shows that *D-E-C* and *D-E* models demand much more compute resource than the *D-C* model. In the *D-E-C* model, the load is mostly put on the edge hub module that manages message routing while heavy computations take place in the cloud. When it comes to the *D-E* model, both message routing and computations take place in the edge, which is more resource-consuming.

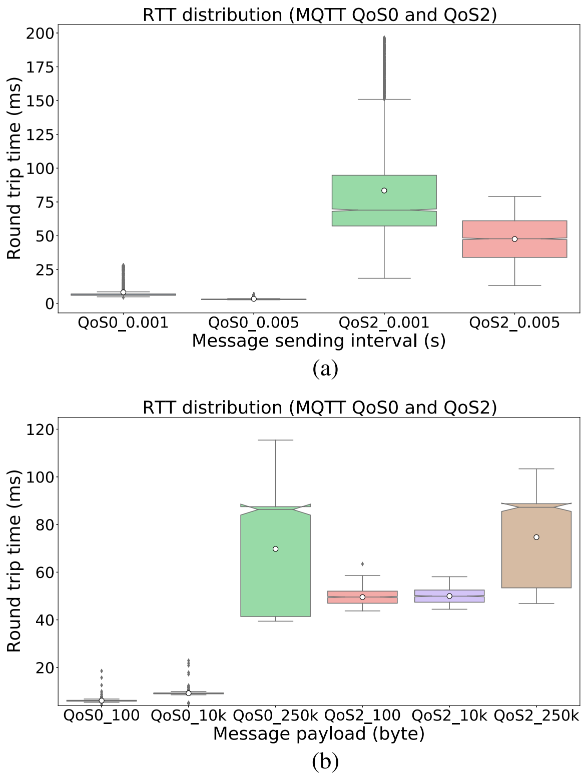

FIGURE 6. Sub-figure (a): Full stack round trip time with regard to message sending interval when MQTT QoS is set to 0 and 2. Sub-figure (b): Full stack round trip time with regard to payload when MQTT QoS is set to 0 and 2. These two experiments are conducted with the Paho MQTT library.

5) THE IMPACT OF MQTT QOS

Different MQTT QoS values can have different impacts on the communication quality as well as latency. In experiment 1, the MQTT QoS is set to 1 for all the four test scenarios, which is hard coded to the Azure edge-cloud infrastructure. Based on the Paho MQTT library, the full stack RTT performance with regard to message sending interval and payload using QoS 0 and 2 are investigated and the results are shown in Fig. 6. It can be observed that the conclusions we draw from QoS 1 still hold for QoS 0 and 2 test cases. In Fig. 6(a), for both QoS 0 and 2 cases, as message sending interval decreases from 5 ms to 1 ms, the RTT performance goes worse, i.e., the average RTT is increased and the values become less deterministic. In Fig. 6(b), a similar performance degradation is witnessed when the payload is increased from 100 bytes to 250 kilobytes for both QoS 0 and 2. The maximum payload size that can guarantee the optimal performance is 10 kilobytes, the same as in the case of QoS 1. Meanwhile, it can be noticed that QoS 0 can always deliver the optimal full stack RTT performance among all QoS values when other parameter settings are consistent, which benefits from the nature that QoS 0 does not require acknowledgement for messages.

B. RESULTS OF EXPERIMENT 2

The results of experiment 2 are recorded in Table III, which shows the processing capabilities of three platforms

TABLE 3 A Processing Capability Comparison on the Edge Host Operating System, Container on Edge, and the Cloud Virtual Machine. a Regression Machine Learning Model, an Auto-Encoder Neural Network Model, and a Long Short-Term Memory-Encoder Decoder Ensemble Model are Utilized to Benchmark the Execution Time, CPU Load and Memory Utilization. the Peak and Average Percentage Values are Based on the Resource Utilization of a Single Process, Which Can Be Above 100%, Depending on the Number of CPU Cores, E.g., 400% is the Maximum for a 4-Core CPU. the System Percentage Values are Based on the Resource Utilization of the Whole System, I.e., Maximum 100%. Comparisons of the Cost and the Scalability are Also Provided. (More * Means Better Scalability.)

Platform			Raspberry Pi 3B+ Raspbian Stretch			Raspberry Pi 3B+ Debian Stretch Container			Cloud VM Ubuntu 18.04.5 LTS		
Model			Regression	Auto-encoder	LSTM-ED	Regression	Auto-encoder	LSTM-ED	Regression	Auto-encoder	LSTM-ED
Time	Training		320 ms	39134 ms	1304 s	341 ms	40915 ms	1339 s	26 ms	4293 ms	102.7 s
	Prediction		0.317 ms	199.11 ms	314.55 ms	0.323 ms	203.56 ms	716 ms	0.075 ms	0.988 ms	7.26 ms
CPU Load (%)	Training	Peak	101.0	125.0	134.0	212.0	127.0	136.0	291.0	176.0	280.0
		Average	99.71	117.71	127.85	176.43	119.9	130.34	274.05	175.5	239
		System	26.16	35.36	37.14	70.72	37.13	38.48	76.49	40.35	77.67
	Prediction	Peak	100.0	124.0	132.0	209.0	125.0	136.0	285.0	166.0	294.0
		Average	99.5	115.91	130.36	188.66	121.6	133.68	236.4	154.73	282.73
		System	25.79	26.21	27.03	25.86	26.26	27.69	25.16	26.4	67.18
Memory Usage (%)	Training	Peak	8.5	21.8	31.4	8.1	22.1	32.0	0.6	1.4	2.3
		Average	8.15	21.76	30.58	7.68	21.74	30.94	0.55	1.4	2.3
		System	76.55	88.65	95.2	84.14	93.56	95.68	7.03	8.34	6.93
	Prediction	Peak	6.8	30.9	33.6	6.4	25.0	33.0	0.5	1.6	2.3
		Average	6.8	26.21	32.2	6.4	23.53	31.51	0.5	1.52	2.3
		System	75.64	92.32	93.84	84.20	95.57	94.63	6.98	8.43	6.94
Cost	Training		35 USD (One time cost)			35 USD (One time cost)			0.15 USD/hour (pay as you go)		
	Prediction										
Scalability	Hardware		*			*			***		
	Software		*			**			***		

concerning executing machine learning applications in the edge-cloud architecture.

From execution time perspective, the performance of the container-based edge environment is rather close to the host OS in both training and prediction tasks. For the regression and the auto-encoder models, the difference in execution time can be neglected. As for the complicated LSTM-ED model, running in the containerized environment takes 35 seconds more in training, which is merely slowed down by 2.6%. While the prediction time is doubled, it is still bounded to a few hundred milliseconds that is acceptable to an edge platform. The performance of the cloud VM shows great advantages. Compared to the container environment, the training time is accelerated by 13, 9.5 and 13 times and the prediction time 4.3205 and 98.6 times, for the regression, auto-encoder and LSTM-ED models respectively.

In respect to the CPU and memory utilization, the peak and average values reflect a single application's consumption of the hardware resource while the system percentage value is the main focus. The system level utilization metric is also more significant because running a containerized application needs the support from container runtime. Comparing the CPU load and the memory usage, the performance of running the auto-encoder and the LSTM-ED models in the edge container and on the edge host OS is close. For the regression model, in the container, the training task is scheduled to execute with the multithreading technique to speed it up, leading to 70% system CPU load and 84% memory usage, which are much higher than executing on the host OS (26% system CPU load and 76% memory usage). The CPU load in the cloud VM is generally higher than in the edge for both training and prediction tasks of the three models while the memory utilization is limited to 8.43% in the worst case, benefiting from the equipped 16 GB memory. High CPU load and large memory of the cloud VM also explain the superior performance in terms of the execution time metric.

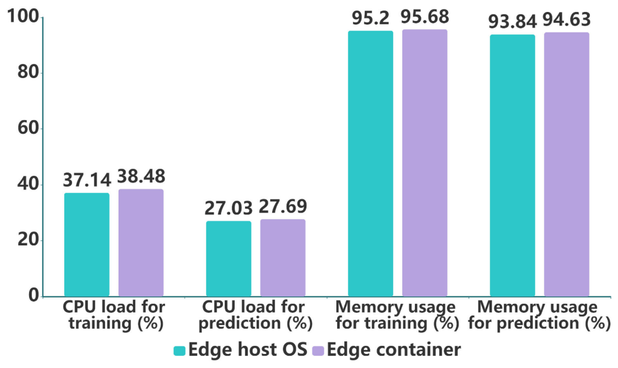

FIGURE 7. A comparison of resource utilization between the edge host OS and the edge container with regard to executing the LSTM-ED model.

Fig. 7 gives a visual comparison between the edge host OS and the edge container with regard to resource utilization of executing the LSTM-ED model. The measured metrics are sufficiently close. The result suggests that, compared to running machine learning on the host OS, the container-based virtualization does not introduce a considerable performance downgrade but offers additional flexibility and scalability to the deployment of applications, which is a complement to the computing and intelligence features in the edge-cloud computing paradigm. The cloud-based virtual machine can be used for processing computation-demanding tasks such as neural networks model training, with which the execution time can be greatly accelerated while maintaining a minimum cost.

C. HIGHLIGHTS OF EXPERIMENTAL RESULTS

IoT applications are categorized into the massive IoT, broadband IoT, critical IoT and industrial automation IoT [53]. The industry poses distinct performance requirements on IoT applications. For instance, the latency requirement in industrial automation is much demanding, e.g., < 2 ms cycle time for

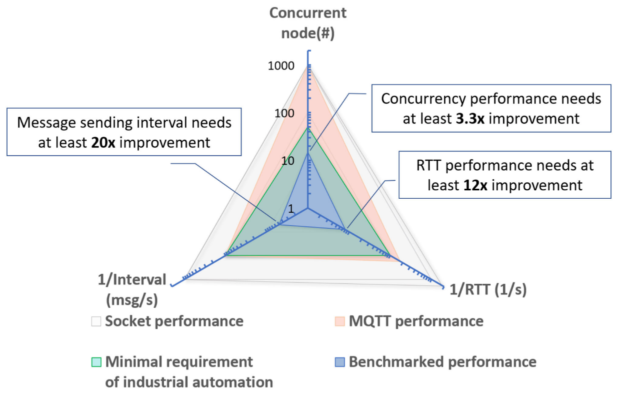

FIGURE 8. Performance gaps between the benchmarked edge-cloud infrastructure, the minimal requirement for industrial automation, and the performance of raw MQTT/Socket protocols are rather large. (Note logarithmic scale is used.) Current edge-cloud infrastructure needs to improve at least 3.3, 12 and 20 times in terms of concurrency, RTT and message sending interval to fulfill industry automation needs.

motion control, 2-10 ms for factory automation, $\sim$ 50 ms for process monitoring and 10-100 ms for video-operated remote control [54] while latency is relatively tolerable in the massive IoT category, e.g., 40-500 ms for traffic management, $<$ 1 s for audio and video transfer in smart grid [55], etc. The payload in industrial communication can range from a few bytes for process automation up to 250 bytes for machine control [54] and 1.5 kilobytes for smart grid [56]. Considering the number of concurrent nodes, typically $>$ 20 are deployed for high-mobility nodes and 10000 are required for low-mobility nodes [54].

The quantitative results of the two experiments provide a direct view of the performance of containerized edge-cloud computing stacks from a client perspective. Therefore, it's more meaningful to evaluate the results according to aforementioned hard values from industrial requirements. The key findings highlighted by the experiment results are reflected by Fig. 8 and listed as follows.

- With current implementation of the containerization based edge-cloud computing infrastructure provided by the public cloud industry, an average full stack RTT above 100 ms is observed, regardless of locating the service to the cloud or the edge.
- The time sensitivity and determinism can only be guaranteed when message sending interval is limited to 200 ms in the benchmarking, which is far from the performance promised by the cloud supplier, indicating that the cloud industry has not been prepared to enter the industrial automation sector but remain in the massive IoT domain.
- Twenty concurrent nodes connecting to the edge already shows non-negligible performance degradation to the system, which cannot fulfill the industry automation requirement in which up to 10000 nodes can be deployed.
- The system performance is insignificantly impacted by the bandwidth variation whereas the quality of service is guaranteed when the payload is below 10 kilobytes, which is sufficient for many industry automation use cases.
- The MQTT protocol and the TCP/IP stack are able to satisfy the demanding latency needs for time-critical tasks in IIoT, with 5.7 ms and less than 1 ms full stack RTT achieved, respectively. Therefore, the latency performance of current edge-cloud computing infrastructure imposes the bottleneck, which shall be largely optimized.
- To reach the minimal requirement of industry automation, the performance of current edge-cloud infrastructure shall be improved by at least 3.3, 12, and 20 times in concurrency, RTT and message sending interval, respectively.
- Container-based virtualization does not bring noticeable performance loss in terms of communication, computing and intelligence compared to the host OS, thus can play an essential role in the edge-cloud computing paradigm for industrial applications. It is able to execute machine learning tasks at the edge. However, the execution efficiency and resource utilization suggest the cloud shall be prioritized for heavy task load.

From client perspective, the performance evaluation results can be taken as a reference at the beginning of solution design for industrial applications that consider using the state-of-the-art edge-cloud computing infrastructures. With current performance, an appropriate task partitioning between the edge and the cloud is inevitable for industrial applications. Taking the performance limitation into account, a static partitioning strategy can be utilized, in which the tasks are broken down into microservices and distributed to the edge and the cloud. More complicated dynamic partitioning strategies that enable online offloading of tasks between edge devices and the cloud shall also be investigated.

VII. PARTITIONING FOR EDGE-CLOUD COMPUTING: A CASE STUDY

In this section, a real application, i.e., the vertical plant wall system (VPS) [57], is introduced as a case study to showcase utilization of the container-based edge-cloud computing paradigm with proper partitioning in industrial applications.

Fig. 9(a) shows a VPS that is installed in a workshop. A VPS consists of vertically grown greenery and is used to purify indoor climate. As shown in Fig. 9(b), with IoT technologies, environmental sensors that measure temperature, humidity, CO_2, particulate matter (PM), multiple gas concentrations, illumination intensity and the water level in the tank, are installed to a VPS. Controllers used for manipulating the LED lighting, ventilation and irrigation systems are also deployed. In such a system, sensor readings shall be periodically sent to the cloud while the controller can be invoked on-site and from a remote site. In general, the sensor update and remote operation of controllers are latency tolerable, except that when a water leakage is detected, the water pump shall be shut down immediately. In light of this, a static partitioning strategy is leveraged in the solution design to break down the system

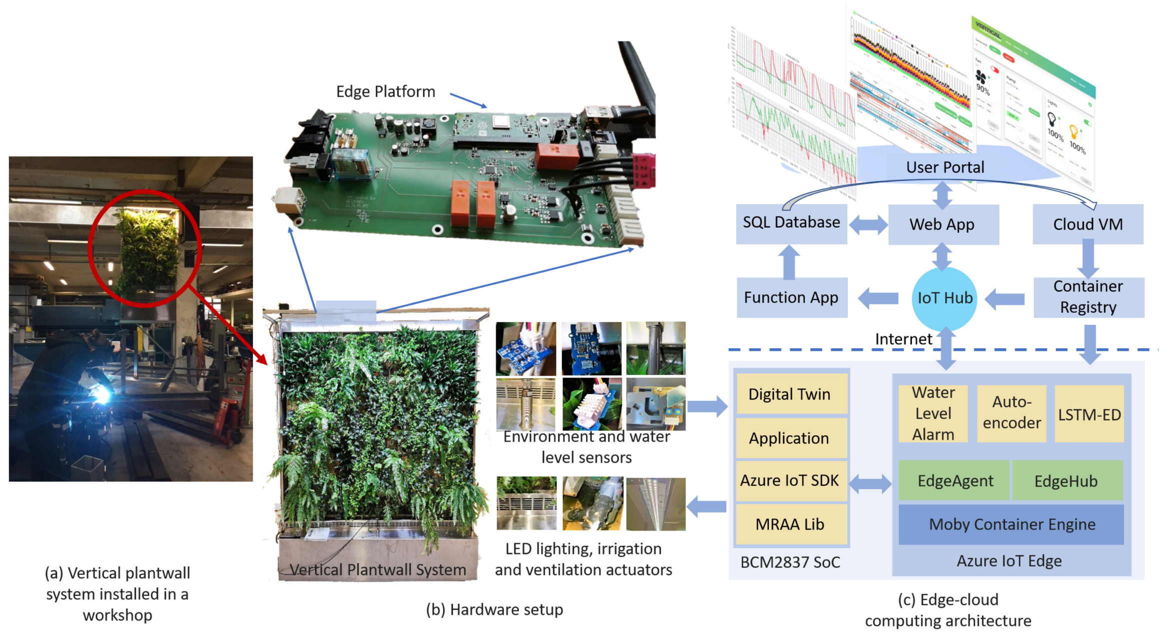

FIGURE 9. **Applying container-based edge-cloud computing paradigm to the vertical plantwall system. (a) A vertical plantwall system is installed in a workshop. (b) The hardware setup of a vertical plantwall system equipped with sensors, actuators and a BCM2837 SoC-based edge platform. (c)The block diagram of the edge-cloud computing architecture for the vertical plantwall system. A partitioning strategy is leveraged to break down the system tasks into microservices that can be deployed to the edge host OS, edge containers and the cloud.**

TABLE 4 **Static Partitioning of a Vertical Plantwall System. the System Functions are Partitioned Into Microservices That are Distributed Across the Edge-Cloud Computing Infrastructures According to Their Performance Requirements**

Function	Microservice	Performance Requirement	Platform
Sensor update	Local application for sensor reading and actuator control	Latency tolerable, offline capability, hardware accessibility	Host OS
Actuator control			
Water level alarm	Containerized water level alarm and control service	Latency <200 ms, offline capability	Edge container engine
Anomaly detection	Containerized auto-encoder model	Online learning capability	
	Containerized LSTM-ED model		
Data storage	SQL database	High scalability, massive storage capacity	Cloud platform
Model training	VM and Container registry	Powerful computing capability	
Human-machine interface	Web application for visualization and interaction	Remote accessibility	

functions into microservices that can be deployed throughout the whole stack. The partitioning details are described in Table IV while the block diagram of the used edge-cloud architecture with service partitioning is shown in Fig. 9(c).

The edge platform is built on a BCM2837 SoC. Sensor update and actuator control functions are latency tolerable but shall have offline capability and hardware accessibility. Therefore, an application natively running on the edge host OS is developed with Azure IoT SDK and the MRAA library to enable interfacing with hardware and communication with the edge-cloud infrastructure. The control parameters are stored in the digital twin to ensure offline capability, which can also be updated from the cloud.

Taking advantage of the Azure IoT Edge infrastructure, three functions are deployed as containers to the edge, i.e., a water level alarm module and two anomaly detection modules using the auto-encoder and LSTM-ED models that are used in Experiment 2. The periodically updated sensor data will be routed to all three modules by the Edge Hub module. With a containerized water level alarm module in the edge, the water tank is monitored and the pump can be stopped in real-time (< 200 ms) in case of an intermittent offline. Containerization also enables the anomaly detection models to realize online learning as they can be flexibly upgraded to the latest models.

As for the cloud, IoT Hub is deployed to enable bidirectional communication between the services and the device. A serverless function application that can be triggered by IoT Hub messages is used to save sensor data to an SQL database that features scalable and massive storage capacity, whereas a web application-based human-machine interface is developed to visualize sensor data and manipulate control functions, which shall accessible from anywhere. Besides, benefiting from the performance advantage, a cloud VM is exploited for training neuron network models, which are then pushed to a container registry service and fetched by the IoT Edge so as to realize online training. In this way, a VPS can be fully digitized by partitioning the tasks throughout the edge-cloud infrastructure.

VIII. DISCUSSION

Catering to the edge-cloud computing paradigm, industrial applications are migrating from monolithic towards cloud native applications, which highly relies on the containerization technology. The performance evaluation experiments conducted in this study intend to provide a client perspective understanding of the capability of containerization in edge-cloud computing stacks on the state-of-the-art infrastructures.

Considering the nature that majority in the industrial society are clients of edge-cloud computing stacks, the proposed holistic evaluation methodology can offer a client perspective view of the system performance. The results can be a reference to applications using a similar edge-cloud implementation while the methodology can be referenced by developers to understand the holistic system performance and determine whether a fine-grained probe on each stack layer shall be conducted. In IIoT practice, for performance-relaxed applications, after aforementioned evaluation, appropriate task partitioning strategies can be applied at the beginning of solution design. For performance-demanding applications, more fine-grained experiments shall be conducted to investigate bottlenecks of performance. The evaluation methodology used in this study can also be referenced to perform a joint benchmarking of communication and computation for a specific industrial application.

Limited by the platform used in this evaluation, i.e., the Azure IoT Edge infrastructure, many other parameters that can potentially impact the performance such as the QoS value and the scheduling policy cannot be thoroughly studied. As for the future, evaluation and development based on open source implementation of edge-cloud computing stacks shall be considered, as it offers more flexibility and transparency than commercial implementations. Meanwhile, evaluation on real-time operating systems, different physical platforms, and with other quantitative metrics such as power consumption can also be explored.

IX. CONCLUSION

In this study, we conducted a comprehensive performance evaluation of the popular containerized edge-cloud architecture to investigate the performance gap between available edge-cloud stacks and industry requirement. Three edge-cloud connectivity models are benchmarked using the state-of-the-art edge-cloud computing infrastructures, i.e., the Azure IoT Hub/IoT Edge platform. The holistic capabilities in terms of communication, computing and intelligence are measured with full stack round trip time and system resource utilization. We find out that current implementation of the edge-cloud infrastructure by the public cloud industry has not been full-fledged for time-critical industrial applications. The performance in respect to concurrency, RTT and message sending interval must be improved by at least 3.3, 12, and 20 times, respectively to satisfy industry requirement. We also confirm that the native MQTT and TCP/IP protocols are able to offer demanding latency performance, showing a large room for optimization in today's edge-cloud computing infrastructure. The results also show that container-based virtualization does not introduce noticeable performance loss in communication, computing and intelligence, which indicates that containerization has a promising future in the edge-cloud computing paradigm. We also call for partitioning to be leveraged for industrial applications and verified the feasibility with a case study, i.e., the digitalization of a vertical plant-wall system using the edge-cloud computing paradigm.The results and evaluation methodology of the study give a client perspective overview of the system performance, providing a reference to both users in the industry and developers as a start point before fine-grained probe of a given edge-cloud computing stack.

ACKNOWLEDGMENT

The authors thank Ola Weister at Vertical Plant Systems AB for providing vertical plant walls as research platform, and thank Associate Professor Evangelos Angelakis for his valuable inputs.

REFERENCES

[1] Y. Liu, K. Akram Hassan, M. Karlsson, Z. Pang, and S. Gong, "A data-centric Internet of Things framework based on azure cloud," *IEEE Access*, vol. 7, pp. 53839–53858, 2019.

[2] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Netw.*, vol. 33, no. 5, pp. 156–165, Sept./Oct. 2019.

[3] "Noun Project: Free icons & stock photos for everything," Accessed: Dec. 25, 2020. [Online]. Available: https://thenounproject.com/

[4] kgremban, "Azure IoT Edge documentation," Accessed: Dec. 25, 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/

[5] "Greengrass," Accessed: Dec. 25, 2020. [Online]. Available: https://aws.amazon.com/greengrass/

[6] "IBM edge application manager - IBM developer," Accessed: Dec. 25, 2020. [Online]. Available: https://developer.ibm.com/components/ibm-edge-application-manager/

[7] "Kubeedge," Accessed: Dec. 25, 2020. [Online]. Available: https://kubeedge.io/en/

[8] C. Bravo and H. Bäckströ, "Edge computing and deployment strategies for communication service providers," White Paper, Ericsson, 2020.

[9] A. A. Kumar S., K. Ovsthus, and L. M. Kristensen, "An industrial perspective on wireless sensor networks–A survey of requirements, protocols, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1391–1412, Jul.–Sep. 2014.

[10] X. Ma and X. Chen, "Performance analysis of IEEE 802.11 broadcast scheme in ad hoc wireless LANs," *IEEE Trans. Veh. Technol.*, vol. 57, no. 6, pp. 3757–3768, Nov. 2008.

[11] S. Hu, Y. Zhu, and X. Xiao, "Performance research of IEEE 802.11 WLAN," in *Proc. Int. Conf. Comput. Inf. Sci.*, 2013, pp. 1323–1326.

[12] C. Wang, C. Chou, P. Lin, and M. Guizani, "Performance evaluation of IEEE 802.15.4 nonbeacon-enabled mode for Internet of vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 6, pp. 3150–3159, Dec. 2015.

[13] M. S. Akbar, H. Yu, and S. Cang, "Performance optimization of the IEEE 802.15.4-based link quality protocols for WBASNS/IOTS in a hospital environment using fuzzy logic," *IEEE Sensors J.*, vol. 19, no. 14, pp. 5865–5877, 2019.

[14] N. Q. Dinh, S.-W. Kim, and D.-S. Kim, "Performance evaluation of priority CSMA-CA mechanism on ISA100.11A wireless network," in *Proc. 5th Int. Conf. Comput. Sci. Convergence Inf. Technol.*, 2010, pp. 991–996.

[15] F. P. Rezha and S. Y. Shin, "Performance analysis of ISA100.11a under interference from an IEEE 802.11b wireless network," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 919–927, May 2014.
[16] T. Zhong, C. Mengjin, Z. Peng, and W. Hong, "Real-time communication in WIA-PA industrial wireless networks," in *Proc. 3rd Int. Conf. Comput. Sci. Inf. Technol.*, vol. 2, 2010, pp. 600–605.
[17] M. Zheng, W. Liang, H. Yu, and Y. Xiao, "Performance analysis of the industrial wireless networks standard: WIA-PA," *Mob. Netw. Appl.*, vol. 22, no. 1, p. 139–150, Feb. 2017.
[18] Q. Huang, A. Sikora, V. F. Groza, and P. Zand, "Simulation analysis of wirelessHART nodes for real-time actuator application," in *Proc. IEEE Int. Instrum. Meas. Technol. Conf. Proc.*, 2014, pp. 1590–1594.
[19] J. Benoit, A. Yao, L. Saladis, and Y. Zheng, "Performance evaluations of multi-hop wirelessHART network and 6LoWPAN using different topologies," in *Proc. Global Smart Ind. Conf.*, 2018, pp. 1–5.
[20] E. Azoidou, Z. Pang, Y. Liu, D. Lan, G. Bag, and S. Gong, "Battery lifetime modeling and validation of wireless building automation devices in thread," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 2869–2880, Jul. 2018.
[21] D. Lan, Z. Pang, C. Fischione, Y. Liu, A. Taherkordi, and F. Eliassen, "Latency analysis of wireless networks for proximity services in smart home and building automation: The case of thread," *IEEE Access*, vol. 7, pp. 4856–4867, 2019.
[22] H. Lee and K. Ke, "Monitoring of large-area IoT sensors using a lora wireless mesh network system: Design and evaluation," *IEEE Trans. Instrum. Meas.*, vol. 67, no. 9, pp. 2177–2187, Sep. 2018.
[23] D. Croce, M. Gucciardo, S. Mangione, G. Santaromita, and I. Tinnirello, "Lora technology demystified: From link behavior to cell-level performance," *IEEE Trans. Wireless Commun.*, vol. 19, no. 2, pp. 822–834, Feb. 2020.
[24] A. Furtado, J. Pacheco, and R. Oliveira, "PHY/MAC uplink performance of lora class a networks," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6528–6538, Jul. 2020.
[25] "Production-grade container orchestration," Accessed: Dec. 25, 2020. [Online]. Available: https://kubernetes.io/
[26] J. S. Katz, "AIoT: Thoughts on artificial intelligence and the Internet of Things," 2019. [Online]. Available: https://iot.ieee.org/conferences-events/wf-iot-2014-videos/56-newsletter/july-2019.html
[27] H. Zeng, B. Wang, W. Deng, and W. Zhang, "Measurement and evaluation for docker container networking," in *Proc. Int. Conf. Cyber-Enabled Distribution Comput. Knowl. Discov.*, 2017, pp. 105–108.
[28] R. S. V. Eiras, R. S. Couto, and M. G. Rubinstein, "Performance evaluation of a virtualized http proxy in KVM and docker," in *Proc. 7th Int. Conf. Netw. Future*, 2016, pp. 1–5.
[29] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for IoT using docker and edge computing," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 118–123, Sep. 2018.
[30] M. Sollfrank, F. Loch, S. Denteneer, and B. Vogel-Heuser, "Evaluating docker for lightweight virtualization of distributed and time-sensitive applications in industrial automation," *IEEE Trans. Ind. Informat.*, to be published, doi: 10.1109/TII.2020.3022843.
[31] T. Leppanen *et al.*, "Edge-based microservices architecture for Internet of Things: Mobility analysis case study," in *Proc. IEEE Global Commun. Conf.*, 2019, pp. 1–7.
[32] F. Carpio, M. Delgado, and A. Jukan, "Engineering and experimentally benchmarking a container-based edge computing system," in *Proc. IEEE Int. Conf. Commun.*, 2020, pp. 1–6.
[33] T. Pfandzelter and D. Bermbach, "tinyFAAS: A lightweight faas platform for edge environments," in *Proc. IEEE Int. Conf. Fog Comput.*, 2020, pp. 17–24.
[34] Z. Y. Thean, V. Voon Yap, and P. C. Teh, "Container-based MQTT broker cluster for edge computing," in *Proc. 4th Int. Conf. Workshops Recent Adv. Innov. Eng.*, 2019, pp. 1–6.
[35] P. Ferrari *et al.*, "Performance evaluation of full-cloud and edge-cloud architectures for Industrial IoT anomaly detection based on deep learning," in *Proc. II Workshop Metrology Ind. 4.0 and IoT (MetroInd4.0 IoT)*, 2019, pp. 420–425.
[36] E. Preeth N, F. J. P. Mulerickal, B. Paul, and Y. Sastri, "Evaluation of docker containers based on hardware utilization," in *Proc. Int. Conf. Control Commun. Comput. India*, 2015, pp. 697–700.
[37] F. Ramalho and A. Neto, "Virtualization at the network edge: A performance comparison," in *Proc. IEEE 17th Int. Symp. A. World Wireless, Mobile Multimedia Netw.*, 2016, pp. 1–6.
[38] R. Morabito, "Virtualization on Internet of Things edge devices with container technologies: A performance evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.
[39] A. S. Gaur, J. Budakoti, and C. Lung, "Design and performance evaluation of containerized microservices on edge gateway in mobile IoT," in *Proc. 2018 IEEE Int. Conf. Internet Things (iThings). IEEE Green Comput. Commun. (GreenCom). IEEE Cyber, Physical. Social Comput. (CPSCom). IEEE Smart Data (SmartData)*, Halifax, NS, Canada, 2018, pp. 138–145, doi: 10.1109/Cybermatics_2018.2018.00055.
[40] R. Kumar and B. Thangaraju, "Performance analysis between RunC and kata container runtime," in *Proc. IEEE Int. Conf. Electron., Comput. Commun. Technol.*, 2020, pp. 1–4.
[41] N. Gupta, K. Anantharaj, and K. Subramani, "Containerized architecture for edge computing in smart home: A consistent architecture for model deployment," in *Proc. Int. Conf. Comput. Commun. Inform.*, 2020, pp. 1–8.
[42] D. N. Jha, S. Garg, P. P. Jayaraman, R. Buyya, Z. Li, and R. Ranjan, "A holistic evaluation of docker containers for interfering microservices," in *Proc. IEEE Int. Conf. Serv. Comput.*, 2018, pp. 33–40.
[43] S. Liu and Y. Zu, "Design and research of edge layer service platform based on flexible service architecture," in *Proc. IEEE 10th Int. Conf. Softw. Eng. Serv. Sci.*, 2019, pp. 555–560.
[44] T. Lennvall, M. Gidlund, and J. Åkerberg, "Challenges when bringing IoT into industrial automation," in *Proc. IEEE AFRICON*, 2017, pp. 905–910.
[45] J. Pfrommer, A. Ebner, S. Ravikumar, and B. Karunakaran, "Open source OPC UA PubSub over TSN for realtime industrial communication," in *Proc. IEEE 23rd Int. Conf. Emerg. Technol. Factory Automat.*, vol. 1, 2018, pp. 1087–1090.
[46] J. F. Inglés-Romero, A. Romero-Garcés, C. Vicente-Chicote, and J. Martínez, "A model-driven approach to enable adaptive QoS in DDS-based middleware," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 1, no. 3, pp. 176–187, Jun. 2017.
[47] E. Al-Masri *et al.*, "Investigating messaging protocols for the Internet of Things (IoT)," *IEEE Access*, vol. 8, pp. 94 880–94 911, 2020.
[48] T. Sultana and K. A. Wahid, "Choice of application layer protocols for next generation video surveillance using internet of video things," *IEEE Access*, vol. 7, pp. 41 607–41 624, 2019.
[49] L. Šikić *et al.*, "A comparison of application layer communication protocols in IoT-enabled smart grid," in *Proc. Int. Symp. ELMAR*, 2020, pp. 83–86.
[50] Y. Liu, Z. Pang, M. Karlsson, and S. Gong, "Anomaly detection based on machine learning in IoT-based vertical plant wall for indoor climate control," *Building Environ.*, vol. 183, 2020, Art. no. 107212. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360132320305837
[51] I. Craggs, "Eclipse Paho," Accessed: Dec. 25, 2020. [Online]. Available: https://www.eclipse.org/paho/
[52] "Eclipse Mosquitto," Accessed: Dec. 25, 2020. [Online]. Available: https://mosquitto.org/
[53] "Cellular IoT evolution for industry digitalization," White Paper, Ericsson, Jan. 2019.
[54] S. Gangakhedkar, H. Cao, A. R. Ali, K. Ganesan, M. Gharba, and J. Eichinger, "Use cases, requirements and challenges of 5G communication for industrial automation," in *Proc. IEEE Int. Conf. Commun. Workshops.*, 2018, pp. 1–6.
[55] F. Holik, "Meeting smart city latency demands with SDN," in *Proc. Asian Conf. Intell. Inf. Database Syst.*, 2019, pp. 43–54. [Online]. Available: https://doi.org/10.1007/978-3-030-14132-5_4
[56] J. Hiller, M. Henze, M. Serror, E. Wagner, J. N. Richter, and K. Wehrle, "Secure low latency communication for constrained industrial IoT scenarios," in *Proc. IEEE 43rd Conf. Local Comput. Netw.*, 2018, pp. 614–622.
[57] Y. Liu, K. Akram Hassan, M. Karlsson, O. Weister, and S. Gong, "Active plant wall for green indoor climate based on cloud and Internet of Things," *IEEE Access*, vol. 6, pp. 33 631–33 644, 2018.

YU LIU received the B.Eng. degree in electronics science and technology from the Harbin Institute of Technology, Harbin, China, in 2014, the M.Sc. degree in computer science from the University of Trento, Trento, Italy, in 2016, the M.Sc. degree in innovation in information and communication technology from the Technical University of Berlin, Berlin, Germany, in 2017, and the Licentiate of Philosophy in 2019 from the Department of Science and Technology, Linköping University, Linköping, Sweden, where he is currently working toward the Ph.D. degree. His research interests include cloud-based Internet of Things solution, ML or AI in IoT applications, embedded systems, and wireless sensor networks. He was one of the recipients of the Swedish Embedded Award in 2018 and an Invited Speaker in the Embedded Conference Scandinavia, in 2019.

DAPENG LAN received the B.Eng. degree in microelectronics from Sun Yat-sen University, Guangzhou, China, in 2014, the M.Sc. degree in ICT innovation from the KTH Royal Institute of Technology, Stockholm, Sweden, in 2016, and the M.Sc. degree in innovation in information and communication technology from the Technical University of Berlin, Berlin, Germany, in 2017. He is currently a Ph.D. Research Fellow on fog computing with the Department of Informatics, University of Oslo, Oslo, Norway. In 2017, he was with InnoEnergy, Stockholm, Sweden, about smart building energy management. From January to September 2016, he was a Thesis Student with ABB Corporate Research Center, Västerås, Sweden. His research interests include fog computing, Internet of Things, and Distributed systems.

ZHIBO PANG (Senior Member, IEEE) received the M.B.A. degree in innovation and growth from the University of Turku, Turku, Finland, in 2012 and the Ph.D. degree in electronic and computer systems from the KTH Royal Institute of Technology, Stockholm, Sweden, in 2013. He is currently a Senior Principal Scientist with ABB Corporate Research, Västerås, Sweden, an Adjunct Professor with the University of Sydney, Sydney, NSW, Australia, and an Affiliated Faculty and the Ph.D. Supervisor with the KTH Royal Institute of Technology. He is the Co-Chair of the Technical Committee on Industrial Informatics. He is currently an Associate Editor for the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, the IEEE JOURNAL OF BIOMEDICAL AND HEALTH INFORMATICS, and the IEEE JOURNAL OF EMERGING AND SELECTED TOPICS IN INDUSTRIAL ELECTRONICS. He is currently the Guest Editor of the PROCEEDINGS OF THE IEEE, the IEEE INTERNET OF THINGS JOURNAL, and the IEEE REVIEWS IN BIOMEDICAL ENGINEERING. He was an Invited Speaker at the Gordon Research Conference on Advanced Health Informatics (AHI2018), the General Chair of the IEEE ES2017, and the General Co-Chair of the IEEE WFCS2021. He was the recipient of the 2016 and 2018 Inventor of the Year Award by ABB Corporate Research.

MAGNUS KARLSSON (Member, IEEE) was born in Västervik, Sweden, in 1977. He received the M.Sc., Licentiate of Engineering and Ph.D. degrees from Linköping University, Linköping, Sweden, in 2002, 2005, and 2008, respectively. In 2003, he joined the Communication Electronics Research Group, Linköping University, where he is currently an Associate Professor. Apart from his broad interest in electronics system design and microwave technology in particular, his main work involves wideband transceiver and antenna techniques, and wireless communication. The later includes high speed data transmission and sensor networks and its associated applications.

SHAOFANG GONG (Member, IEEE) received the B.Sc. degree in microelectronics from Fudan University, Shanghai, China, in 1982, and the Licentiate of Engineering and Ph.D. degrees from Linköping University, Linköping, Sweden, in 1988 and 1990, respectively. Between 1991 and 1999, he was a Senior Researcher with the research institute RISE Acreo, Gothenburg, Sweden. From 2000 to 2001, he was the Chief Technology Officer with a spin-off company from the research institute. He was an Adjunct Professor with Linköping University. Since 2002, he has been the Chair Professor of communication electronics with Linköping University. His main research interests include communication electronics including radio frequency, microwave system design, high speed data transmissions, and wireless sensor networks towards the Internet of Things.