SEGA: Secured Edge Gateway Microservices Architecture for IIoT-based Machine Monitoring

Atonu Ghosh, *Graduate Student Member, IEEE,* Anandarup Mukherjee, *Graduate Student Member, IEEE,* and Sudip Misra, *Senior Member, IEEE*

***Abstract*—In this work, we propose SEGA, a secured edge gateway microservices architecture for Industrial IoT-based monitoring of machines in industries. SEGA allows the secured collection, transmission, and temporary storage of data within the edge network. A KNN-based analytics module hosted on the edge gateway processes time-sensitive machine monitoring data on the gateway itself and identifies machines' operational status. The system predicts the machine state and displays the monitored parameters such as current consumed, power factor, power consumption, and vibrational state of machinery. SEGA also enables secured offloading of data and advanced analytical functions from the edge gateway to the cloud. SEGA's deployment results show negligible changes in the edge gateway's performance due to the inclusion of various security and encryption mechanisms. However, the resource-constrained edge sensor nodes show an increase in wireless packet transmission latencies between them and the gateway by approximately 84.12 ms.**

***Index Terms*—Industrial IoT, Machine monitoring, Edge networks, Edge Intelligence, Microservices, IIoT Gateway**

I. Introduction

IIOT has found its application in the industries for various use cases such as equipment monitoring, worker safety, task automation, and more. The data generated in the IIoT environment are of various types. Some are time-sensitive, and some need higher security levels to keep the business processes and trade secrets intact. This makes the management and security of the IIoT data among the significant challenges posed to the development of IIoT systems and architectures [1]. Smart edge-based systems help handle time-sensitive data without needing to be sent immediately to the cloud for processing. However, edge devices are limited by their ability to run computationally intensive tasks, perform long-term data storage, and offer dynamically scalable services.

Interestingly, preprocessing of the data on the edge devices helps achieve two primary goals of IIoT – 1) overcoming unnecessary network transfer lags and 2) avoiding network congestion and enhanced bandwidth savings [2]. Further, a combination of edge-computing, backed by cloud-based processing, extends a plethora of benefits to an IIoT network, the most crucial being the ability to secure data much nearer to the site of data gathering/sensing. The traditional monolithic structure of the IoT solutions, especially at and nearer to the edge of the architecture, makes it harder for the solutions to be tailored for specific industrial requirements that vary broadly among businesses [3]. Hence, the lack of these primary requirements of IIoT solutions becomes a bottleneck in the broader adoption of IoT-based solutions in the industries. Typically, an IIoT solution for an industrial application, especially machine monitoring tasks, should meet the following expectations:

- Low latency: industrial monitoring and control applications strongly emphasize on latency requirements of a solution. Typically, latency below 30 ms is a desirable metric in industrial monitoring applications.
- Scalability: industrial applications require systems to be highly scalable to accommodate the changing requirements of the industrial environments and business processes.
- Security: robust and layered security aspects of IIoT solutions are always desirable in industrial applications.

A. Motivation

Owing to the fairly recent developments in the softwarization of solutions and the rise of modular hardware, it has become much simpler to handle massive data volumes at the edge. These new features, along with strategies such as parameter-based offloading of functional processes (i.e., machine learning, big-data operations, and others) to a much robust computing infrastructure (such as a cloud), motivate the use of edge computing in various IIoT architectures. A significant portion of these changes has been made possible by using microservice-based architectures at the cloud and the edge of the IIoT network. As an active research area, these developments in edge-based microservices have not focused largely on security aspects of the data and enhancing the integrity of the information flowing from the site of data collection to the remote server or cloud.

In this work, we address this lacuna in existing IIoT solutions by proposing "SEGA", which is an end-to-end **S**ecured **E**dge **Ga**teway microservice architecture for IIoT-based machinery monitoring in industries. The salient security-rich features of our proposed architecture are as follows:

1) **Sensors:** SEGA implements the first level of security at the data sensing site itself. A "Secure Boot" of the microcontroller in the sensor nodes ensures that the sensor nodes' bootloader is uncompromised. Secondly, implementing "Flash Encryption" of the microcontroller keeps the data, application code, and encryption keys secure. Each sensor node uses a different key for data encryption so that the compromise of one sensor node

A. Ghosh, A. Mukherjee, and S. Misra are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India.

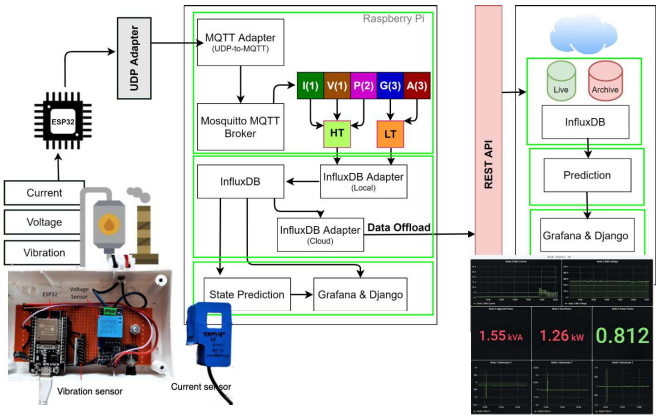

Figure 1: The microservice architecture for SEGA

does not lead to the compromise of the whole system. Lastly, using the one-time programmable hardware-based keys of the microcontroller further prevents any undue manipulations to the hardware.

2) **Edge Gateway:** The microservices communicate securely among themselves using strict authentication-based Application Programming Interfaces (APIs), which acts as the primary layer of security. The API secret keys are stored securely in an access restricted system environment. The internal communication and networking among the edge-based microservices and the cloud-based ones are done through an entirely different private network. These features, along with highly differentiated and well-defined user-access abstraction levels, acts as another security barrier to unauthorized access to edge-based data.
3) **Edge to Cloud:** The data in transit from the edge to the cloud are secured through encryption using public-key cryptography, and the transmission channel uses a Secure Shell (SSH) connection over TCP/IP. Additionally, the SSH connections are secured by enforcing authentication key-based validation of the communicating parties.

B. Contributions

The proposed "SEGA" system demonstrates the design and implementation of an end-to-end secured system for IIoT. The proposed architecture is highly fault-tolerant, scalable, and reliable due to the microservice-based modular architecture along with its contemplative configuration that has been implemented. The specific contributions of this work are as follows,

- A detailed end-to-end architecture of a microservice-based edge computing system for IIoT has been proposed. In addition, each system component and its integrations have also been discussed elaborately.
- The security measures employed in the proposed microservice-based system have been discussed and demonstrated on a real system.
- The time-sensitivity of IIoT applications has been prioritized in the proposed system architecture. In addition, the effect of security measures in resource-constrained devices has been experimented with and evaluated.

II. Related Work

IIoT environment comprises a massive number of sensor nodes that collectively generate a mammoth volume of data [4]. Moreover, in specialized and precise IIoT applications such as condition monitoring, the data generation rates are quite high, and so is the generated data load on the network. These factors lead to exceedingly high bandwidth usage, strains the server in memory, and processing power usage. As a possible solution to such issues, data reduction techniques have been successfully explored for industrial motor monitoring systems [5]. Additionally, edge-based systems have also been used for "Big Data" preprocessing before forwarding the data to the cloud [6]. As most industrial applications are sensitive to time delay, processing data on site using edge computing comes off as a viable solution. Non-time-sensitive tasks are offloaded to remote cloud servers for processing. Alternatively, optimized QoS-aware offloading strategies [7] and AI-based offloading solutions [8] have proved promising. Optimizing network using SDN-based solutions to control the data flow intelligently is yet another promising approach [9].

Towards securing IIoT networks and the data flowing through it, blockchain-based solutions [10] have been developed that exploit its hashing mechanism. Edge-cloud-based Remaining Useful Life (RUL) prediction methods have come up using AI technology to predict the RUL of the machinery. Such practices make use of the "lightweight temporal convolutional networks" [11]. There is not just the need for optimization on the hardware level, but the software aspects must be considered too for an efficient end-to-end IIoT system. As a result, service management leveraging containerization technologies such as Docker is being popularly adopted for IIoT. Studies have shown that such container-based virtualization has resulted in only 16% of the total power consumption with such container-based architecture [12]. The resource-constrained nature of the IoT devices makes it difficult to implement the existing security mechanisms. Despite the challenges, approaches have explored modified RSA algorithms implementing four prime numbers of 512 bits and hash-based signatures for device authentication purposes [13]. Similar developments can be found in the literature where the private-public key-based encryption mechanisms have been implemented [14]. However, most of the existing works in the literature provide security for a portion of the network architecture. Due to limitations of device power, processing, and resources, the security within the edge network gets ignored or is not addressed robustly. This results in mostly unsecured edge layer devices, which are prone to manipulation from adversarial entities. The SEGA system proposed in this work addresses this issue and provides complete security – from the source of data collection to the cloud.

III. System Description

The SEGA system makes it easier and seamless to add or remove sensor nodes from the edge network while taking

precautions that the regular network operation is not hindered. The microservice-based architecture of the edge gateway device requires no additional configuration to accommodate the newly added nodes. The only manual intervention required is to register the AES key of the new sensor node in the edge gateway device. This AES key addition is done through the system management dashboard. The newly added sensor nodes connect to the edge gateway device by using the pre-registered access credentials in its firmware which is encrypted by using "Flash Encryption" as discussed in section IV. Moreover, to handle computation and storage loads beyond the capability of the edge gateway hardware, by virtue of the microservice-based architecture, it requires minimal effort to replicate the edge gateway device to balance the load/traffic in the network.

A. SEGA Architecture

The developed system hardware – sensor nodes, and edge gateway – monitors electrical parameters such as *RMS Voltage*, *RMS Current*, *Real Power*, *Apparent Power*, *Power Factor*, and *Vibration Parameters*. The edge system consists of wirelessly communicating sensor nodes equipped with sensors that send the sensor data to the central edge gateway. The edge gateway connects to the cloud servers over the Internet. The edge gateway device preprocesses the sensor nodes' data. Time-sensitive control instructions such as turning off malfunctioning machinery are generated by the edge gateway device based on the data it receives from the sensor nodes. The non-time-sensitive data are transformed and offloaded to the cloud servers for complex analytics and archival storage. This selective offloading capability of the edge device helps to improve the Quality of Service (QoS) of the SEGA system by reducing unnecessary traffic over the wireless network. Fig. 1 graphically represents the SEGA microservices architecture and the components of the sensor node in our arrangement.

SEGA incorporates the microservice-based architecture, making it highly scalable and flexible to accommodate emerging business requirements of industry and enabling rapid solution deployments. Security and privacy requirements for a reliable IIoT-based data transfer are ensured through security at different levels, including appropriate data encryption on resource-constrained sensor nodes. The edge-based gateway enables the availability of essential services such as real-time monitoring even with intermittent Internet connectivity. The data stored in the cloud enforces robust encryption mechanisms in addition to the authentication mechanisms that make the data breach even more difficult, if not impossible.

B. Network Architecture

Typical IIoT applications often involve a combination of sensor nodes and sensors. If each sensor nodes contain the same number and types of sensors, we refer to such nodes as homogeneous nodes. In this work, we make use of such homogeneous sensor nodes for monitoring industrial machinery. A SEGA gateway G_j may connect to n sensor nodes, and each of these sensor nodes can further contain p sensors. The sensors s with a sensor node n in the proposed architecture can be represented as an array such that,

$$n_i = [s_1, s_2, s_3, \cdots, s_p], \quad p > 0, p \in \mathbb{I}^+ \tag{1}$$

It is to be noted that the set n_i represents both single-dimensional sensors (temperature, power) and multi-dimensional sensor values (vibration values along x, y, and z axes). Further, the amount of data generated in a SEGA system, considering j edge gateways G, can be represented as:

$$G_j = \{[s_1, s_2, \cdots, s_p]_1, [s_1, s_2, \cdots, s_p]_2, [s_1, s_2, \cdots, s_p]_n\} \quad n, j \geq 1, \in \mathbb{I}^+ \tag{2}$$

This deployment of edge gateways and their constituent sensor nodes can be dispersed within the industrial premise or even multiple geographically distributed premises of the same industrial entity. The sensor node in the proposed SEGA monitors a machine/device's electrical parameters such as Current (I_{rms}), Voltage (V_{rms}), Real Power (P) measured in Watts (W), Apparent Power (S) measured in Volt-Amps (VA), Power Factor (Φ) of the machinery and its vibrational states (a_x, a_y, a_z). For each machinery that needs monitoring, a sensor node is allotted for the machinery to sense vibrations and power-related factors of the machinery. Except for the clamp-on current sensor, requiring minimal interference with a monitored machine's power supply, the other parameters collected by the sensor node are mostly non-intrusive. Each live and registered sensor node then transmits the sensed data directly to the edge gateway device. The sensor node derives the values of P, S, and Φ by using the sensed values of I_{rms} and V_{rms}. Considering a resistance r and impedance z, the value of $P = I^2 \times r$, $S = I^2 \times z$, and $\Phi = P \times S^{-1}$. The parameters captured by each sensor node represented by equation 1 is now represented as:

$$n_i = [V_{rms}, I_{rms}, P, S, \Phi, a_x, a_y, a_z] \tag{3}$$

The network topology implementation of the proposed IIoT system is subjective to the specific application requirements such as the area to be covered $(A \times B)$ m^2 and the specified requirements for an acceptable time latency (t). Our implementation of the SEGA system uses the star network topology. Interestingly, our choice of a star topology arises from the fact that its alternative, a mesh-based architecture, will induce massive traffic redundancies in the network and subsequently will increase network latencies over time. A star network avoids such network degradations. Also, nodes far from the sink node experience a longer data transmission delay in a mesh network. This delay increases as the nodes move away from the sink node. Further, a star topology also enhances the traceability of faults and significantly reduces fault detection in the network. Finally, the practicality of using a star topology is also advocated by the long list of wireless communication protocols that support the star topology but not the mesh topology.

The edge gateway device receives data from the sensor nodes, preprocesses the data, stores the data in the local databases for rendering services without Internet connectivity,

and forwards the data to the cloud server whenever Internet connectivity is available. All the services rendered by the edge gateway device are implemented as microservices. The edge gateway device connects to the cloud server using TCP over the Internet. The cloud server hosts a superset of the services that are rendered by the edge gateway device. The cloud-based server performs complex data analytics and large data storage functions.

Definition 1. ***Time-critical data:*** *rapidly changing temporal data that has an approximate update interval of more than* $10Hz$ *(*10 *changes/ data-points per second) is a time-critical data in the context of this work.*

C. Micro-Service Architecture

The services involved with receiving sensor data, decrypting the received data, segregating time-critical data (Definition 1) from others, lightweight analytics on the preprocessed data, data storage, data offloading to the cloud server, and rendering real-time visualization dashboard to the user, all have their dedicated microservices in the SEGA architecture. Each of these microservices is containerized using Docker, which provides several benefits over traditional monolithic IoT service architectures. With microservices, the deployment is rapid and highly flexible, with the added benefit of robustly accommodating emerging business requirements without re-designing the previous system or taking it down. The Docker-based containers being lightweight than Virtual Machines (VMs), provide better performance on resource-constrained devices [15], and hence is selected for the SEGA system. Fig. 1 outlines the microservice architecture at the edge gateway (primarily).

Data Handler Module: this module consists of the UDP adapter, MQTT adapter, MQTT broker, and the local InfluxDB adapter, which is an MQTT subscriber. This module is responsible for receiving data from the sensor nodes, decrypting the data, and data preprocessing. The UDP adapter receives the data from the sensor nodes, and then it sends the data to the MQTT adapter. The MQTT adapter publishes the received data on various MQTT topics of the broker after appending sensor node-specific information to it. The MQTT broker receives topics for both time-critical and non-time-critical data. The topic-wise segregated data are then fed to the local InfluxDB adapter. The Influxdb database records are periodically offloaded to the cloud server by the cloud InfluxDB adapter. This collected data of the current consumption is used for incremental training of the systems classifier that is used for analyzing the machine state.

MQTT Communication: SEGA predominantly communicates using MQTT due to it being an open-source solution [16], its requirements for a significantly lesser network overhead, and especially, its “event-triggered” nature. The ”publish-subscribe” model of MQTT makes the bi-directional and many-to-many device data flow extremely simple in addition to decoupling the sender and the receiver devices, unlike in client-server based communication protocols such as HTTP and CoAP. One of the exclusive features of MQTT is the ability to ”retain” messages. This ”retain” functionality in MQTT ensures that the newly added clients can get the published messages published before they join the network. This becomes useful in scenarios where a particular topic’s messages are infrequent. Thus the newly added clients get the data immediately after entering the network [17]. Other than MQTT, none of the most used communication protocols such as CoAP, AMQP, and HTTP has three QoS provision levels [18]. Owing to MQTT’s method of transmitting messages using named topics, the introduction of malicious data in the network gets obliterated along with unauthorized access to messages. Its diaphanous nature makes it highly suitable for resource-constrained devices that are used in IoT [19]. Moreover, in an application implemented with MQTT, new device integration requires minimum effort and time, making the IoT deployment rapid and scalable.

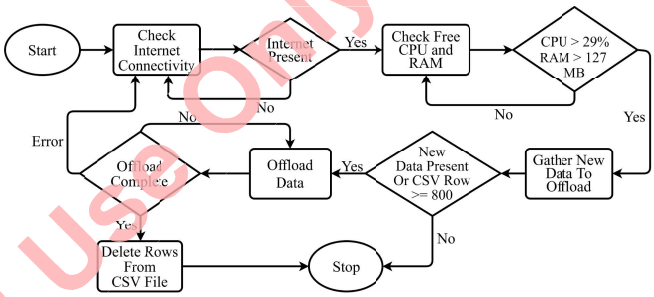

Figure 2: Data offloading mechanism for the SEGA edge gateway architecture

Data Storage Module: consists of InfluxDB, which is an open-source, high-performance time-series database. The data pushed into this database have a tag associated with them that facilitates data grouping based on a sensor node’s identification. This also enables easy traceability of the data.

Visualization module: this uses Grafana for SEGA’s visualization function. Graphs for real-time visualization are enabled by Grafana hosted on a Django server. The user interface rendered by the Django server is accessible over the local network, where it can be viewed independently of the client device.

State Prediction Module: this is responsible for predicting the functional state of the monitored machine. We use a pre-trained k-Nearest Neighbors (KNN) classifier that has been containerized and deployed on the edge gateway device for automatically identifying the machine states. A provision for periodic retraining of the KNN model has been kept on the cloud server. Only the trained model is deployed on the edge gateway device.

Data Offloading Module: this is responsible for sending data to the cloud server from the edge gateway. The data offloading scheme is described in Fig. 2. The edge device checks for the presence of new data to be offloaded to the cloud after confirming the free CPU and RAM of the edge gateway are beyond the pre-set threshold. Using a trial-and-error approach, we set the threshold for free CPU usage to 30%, and the available memory to 128 MB to support basic operations on the edge device.

IV. Secure Data Preprocessing

Data security in the IIoT environment helps maintain the secrecy of the business processes and trade secrets. It prevents unauthorized control of the machinery through compromised nodes in the industrial IoT network. SEGA enforces security at the data generation, communication/transmission, data processing, and storage levels.

Data Generation Security: the ESP32 microcontroller-based sensor nodes at the data generation level implement *Secure Boot* and *Flash Encryption*. The ESP32 hardware consists of a 1024-bit electronic fuse that can be programmed only once. Once the blocks in this fuse are programmed, no software running on the ESP32 device can read or write to these blocks. The Secure Boot mechanism ensures that the execution begins with the"BootROM", a hardware ROM and the electronic fuse, which is also present in the hardware. The BootROM compares the hash of the RSA key of the bootloader image and the hash of the key stored in the electronic fuse. The bootloader is validated using the RSA key that the BootROM validated. The bootloader then checks the integrity of the firmware on the ESP32 device in a similar manner to validate the bootloader. The *Flash Encryption* is a mechanism that keeps the application code on the ESP32 encrypted. The application code is decrypted during the execution and can only be read by the ESP32 hardware, as mentioned earlier. The debugging features such as the JTAG of the ESP32 device are also disabled using the electronic fuse. Fig. 3 shows the schematics of the sensor node and the edge gateway device.

Data Transmission Security (intra-edge): the data to be transmitted from the sensor nodes within the edge uses the key stored in the application firmware to encrypt the data and forward it to the edge gateway device. Once the edge gateway device receives the data, the data is decrypted. The AES encryption scheme is secure as the application firmware is already encrypted by the ESP32, which prevents any adversarial entity from fetching the AES key from the sensor node. In addition, each sensor node uses a different AES key, which ensures that even if a sensor node is compromised, the rest of the sensor nodes' integrity in the network remains intact. The ESP32 microcontroller board in the sensor node was chosen after extensive evaluation of other boards (ESP8266, Raspberry Pi Zero W, Raspberry Pi 3B/4) regarding cost, speed, processing, security, signal converters, and energy consumption profiles. The ESP32 board was chosen for the sensor node as it had more than two analog-to-digital converters (an essential requirement for our sensor node), had enough processing power to support encryption algorithms, had a set of firmware security measures inbuilt, and costs much lesser than other microcontroller boards. The lack of inbuilt ADCs in the Raspberry pi boards meant more add-ons, which would have required higher power consumption, increased costs, and increased the form factor of the sensor node, and hence was discarded. The ESP8266 boards were not powerful or robust enough to match the other boards.

Data Transmission Security (edge to cloud): the Data Offloading Module, which is a separate docker container, is responsible for sending data to the cloud server. It adheres to the offloading scheme described in Fig. 2. The data to be sent is encrypted using a public key. Then the encrypted data is transmitted to the cloud server over a Secure Shell (SSH) connection using TCP. The encrypted data sent to the cloud is then decrypted using the private key and stored in the database, then consumed by applications running on the cloud. Further, the containerized microservices in the edge device and the cloud are loosely coupled and are independent of each other. All the microservices communicate only through APIs, which implements JSON Web Token (JWT) authentication. The JWT tokens generated are renewed every 15 minutes as they are set to expire beyond this time. Therefore, to have access to the docker containers and, in turn, to access the docker container's environment variables, an attacker needs to have access to the edge gateway device as a root user. This authenticated Application Programming Interfaces (API)-based communication ensures that even if one of the microservices is compromised, the rest of the system remains safe.

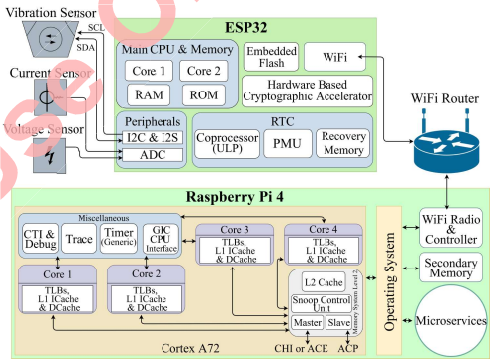

Figure 3: ESP32 and Cortex A72 interoperation

Data Processing Security: the UDP Server receives the encrypted data transmitted by the sensor nodes in the Data Handler Module. The UDP server decrypts the messages, and the decrypted data is preprocessed then sent on various MQTT topics. This topic-based data management provides an added security layer as any data sent over topics other than the predefined ones get discarded. Therefore, an adversary cannot send data to the edge gateway device's modules other than the UDP server without knowing the MQTT topics. Nevertheless, the UDP server checks for the length of the data sent by the sensor nodes. Any mismatch in the data length leads to the discarding of the data. Consequently, this mechanism ensures that the data sent from unknown sources are detected. The containerized microservices also help isolate the application services from the underlying platform of the host system, thereby providing it some immunity from direct attacks on the host.

Data Storage Security: SEGA uses InfluxDB at the gateway as well as the cloud to store the parameters sensed by the sensor nodes. All read and write operations to the database are performed through REST APIs of InfluxDB. The APIs enforce strict authentication and require the user credentials to access it. The users that use the API have only specific rights granted to them. As an additional security measure, the cloud server and the edge gateway device enforce strict access

policies using a firewall and port blocking. It is worth noting that the public-private key pairs used in the system are all different. This ensures the security of modules even if some module of the system gets compromised.

V. Experimental Setup

The proposed sensor node uses the ESP32 microcontroller board. An SCT-013-030 clamp-on current sensor detects and reads the current consumption of the monitored machine into the microcontroller. Although this sensor has a maximum current rating of $30A$, an appropriate sensor module can be easily accommodated with the sensor node for higher current requirements. Further, a ZMPT101B voltage sensor detects the voltage across the monitored machine. The sensor node also uses an MPU-6050 inertial measurement unit (IMU). It consists of a 3-axis Gyroscope and a 3-axis Accelerometer. The ESP32 calculates the various power-related parameters from the voltage and current. These values, along with the vibration readings, are forwarded to the edge gateway over WiFi. Parts of Fig. 1 depicts the hardware implementation of the sensor node.

The edge gateway device is an open-source Cortex-A72 (ARM v8) 64-bit single board computer with a CPU clock speed of 1.5 GHz. It has 4 GB LPDDR4-3200 SDRAM with 2.4 GHz and 5.0 GHz WiFi. The device requires 5 V DC power supply with a minimum of 3 A current. Ubuntu Server 20.04 LTS for ARM is used as the operating system on this single board computer. The ESP32 and Cortex-A72 based sensor node and edge gateway interconnection is depicted in Fig. 3. Once the data from the sensor node is received at the edge gateway, the microservices store it into the Influxdb time-series database. The visualization module plots the data in real-time from the database.

Table I: Parameters for performance evaluation of the proposed system.

Parameter	Description
V_{rms}, I_{rms}, P, S, and Φ	Electrical parameters used in accuracy testing.
Acceleration	It represents the vibrational state of the machine being monitored.
CPU and RAM	Percentage of CPU and RAM consumed by microservices and software adapters on edge gateway device.
MQTT Publish Time	Time taken by the "MQTT Adapter" sub-module of "Data Handler" module to publish the transformed data on to the MQTT topics.
Decryption Time	Time taken by the "Data Handler" module to decrypt the received data (encrypted).
Data Arrival Time	Time between the receipt of successive data packets by the edge gateway device shows the latency of the ESP32 device.
Network Latency	Latency (measured in ms) of the WiFi network in which SEGA was deployed.

VI. Performance Evaluation

The system was tested with an electrical load of 2300 W. During testing, we separately observed that the supply voltage varied between 248 V to 235 V using the AC mains available to us. The sensor node reported a maximum of V_{rms} at 254.9175 V and a minimum value of 233.9435 V. The differences between the reported voltage from the sensor node and our separate observation was approximately ± 7.9 V. Similarly, the I_{rms} value of the difference between our standalone measurement and the one reported by the sensor node was ± 0.5348A. The average Real Power $(P_{average})$ for 274 consecutive readings was 2220.9 W and the average Apparent Power $(S_{average})$ was 2462.4 W, and the phase Φ calculated was 0.9.

A. Effect of Encryption on Edge Gateway Latency

Fig. 4 shows the usual time required for data preprocessing and publishing MQTT topics with no encryption on the edge gateway. We observed that the minimum, maximum, and average preprocessing and MQTT publish time are 37.21 ms, 66.89 ms, and 50.16 ms, respectively. The lower plot in Fig. 4 illustrates the time taken for data decryption, preprocessing, and MQTT publishing by the data handler module when encryption is implemented on the incoming messages to the edge gateway. We observe that the minimum data decryption, preprocessing, and MQTT publish time is 41.68 ms, the maximum time and average time are 73.31 ms and 51.61 ms. Summarily, we infer that the encryption method chosen for securing communication between the sensor nodes and the edge gateway does not significantly affect the network latencies and is comparable.

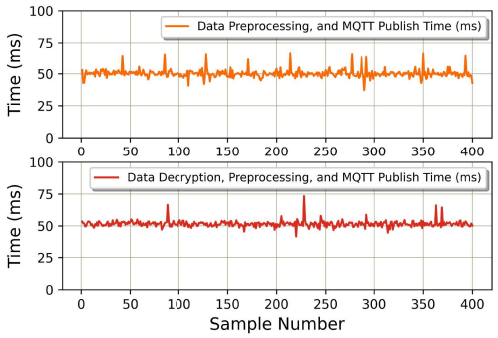

Figure 4: Time for preprocessing, and MQTT transmission (without encryption) and time for decryption, preprocessing, and MQTT transmission (with encryption)

B. Effect of Encryption on Sensor Node Latency

We also measured the successive data arrival times at the edge gateway – both with and without encryption implemented. Fig. 5 shows the successive data arrival times from sensor nodes without and with encryption. The data size measured with encryption is 161 bytes and 151 bytes without encryption. The successive data arrival time indicates the latency of the sensor node. Without encryption, the minimum time between successive data arrival observed is 0.4 ms. The maximum being 999.34 ms and the average time being 37.87 ms. In contrast, the successive data arrival times with encryption implemented shows that the impact of encryption is

significant on the successive data arrival time. The minimum, maximum, and average successive data arrival time with encryption is 4.48 ms, 999.341 ms, and 121.99 ms. From the graphs and the average successive data arrival times, the ESP32 device generated and transmitted data non-uniformly. The maximum times for both scenarios are similar due to the UDP data loss.

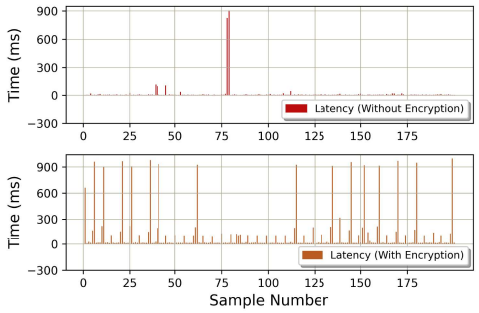

Figure 5: Time between successive data arrival from sensor node to edge gateway (without and with encryption)

C. Effect of Encryption on Edge Gateway Resources

We monitored the edge gateway's resource usage – the amount of free CPU and RAM usage – throughout our experiment. We observed no significant changes to the CPU and RAM usage profiles due to encryption of communication between the gateway and the sensor nodes. For both cases, the maximum CPU utilization observed was 5.6% and the maximum RAM usage with encryption and no encryption were 904.4 MB and 904.2 MB, respectively. However, as per our expectations, the CPU and RAM usage shows a significant increase during the execution of computationally intensive tasks such as a pre-trained KNN classifier for the detection of machine states. In this scenario, the maximum CPU and RAM usage was observed to be 15.4% and 927 MB, respectively.

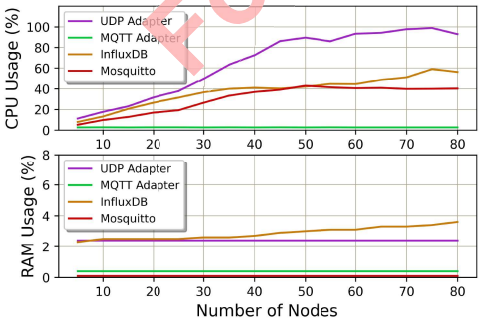

Figure 6: CPU and RAM usage by microservices and adapters of SEGA with increasing number of sensor nodes

D. Effect of Increase in Sensor Nodes

To test the performance of the proposed system, we performed experiments where we increased the number of sensor nodes in the network, 5 nodes at a time. We varied the nodes from 5 nodes to 80 during this experiment. The maximum CPU and RAM usage by the UDP Adapter, MQTT Adapter, InfluxDB microservice, Mosquitto MQTT broker microservice, and the Grafana microservice were recorded. Fig. 6 depicts CPU and RAM usage by the SEGA microservices and the software adapters with an increasing number of sensor nodes.

Grafana microservice: it showed consistent CPU and RAM usage throughout the experiment. It consumed a maximum of 1.6% of RAM and the maximum CPU usage observed is 24.6%. This behavior of the Grafana microservice is justified because the resource usage by the Grafana microservice depends on the number of requests it serves. This is a very scalable behavior and appropriate for an extensive system.

UDP adapter: it consumed the maximum CPU and RAM. We recorded 98.9% and 2.4% CPU and RAM usage, respectively, with 80 sensor nodes connected to the network. The RAM consumed by it remained constant at 2.4% throughout the experiment. However, the CPU usage showed a constant increase with an increasing number of sensor nodes. The minimum CPU usage by it is 11.2% with 5 sensor nodes.

MQTT adapter: it showed a constant RAM usage of 0.4% and the CPU usage only varied between 2.6% and 2.7% throughout the experiment.

Mosquitto MQTT broker: it used a maximum of 42.99% of CPU and a minimum of 5.2% of CPU. Its RAM usage was negligible and remained constant at 0.1%.

InfluxDB microservice: its RAM and CPU usage in the experiment increased with the increase in the number of sensor nodes. The maximum CPU and RAM usage by the Influxdb microservcie were 58.98% and 3.6% respectively with 80 sensor nodes, whereas the minimum CPU and RAM usage were 7.9% and 2.3% with 5 sensor nodes.

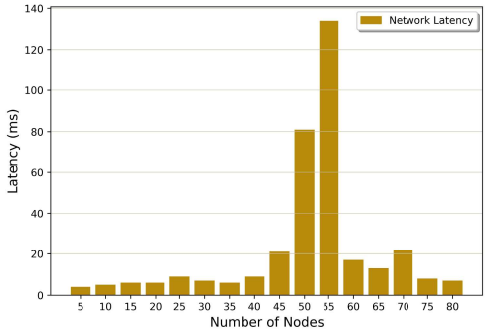

Figure 7: SEGA WiFi network latency with increasing number of sensor nodes connected to the network

E. Effect on Network Latency

Fig. 7 shows the effect of network latency with an increase in edge sensor nodes. For each record of the network latency, we performed 30 consecutive measurements. The maximum value among the 30 measurements was recorded as the network latency with a given number of sensor nodes. The WiFi network experienced a maximum latency of 134 ms and a minimum latency of 4 ms. For only two instances, the network

latency was observed to exceed 22 ms, and for all other measurements, the latency observed was below 23 ms. The sudden spike in the network latency for the two measurements exceeding 22 ms was attributed to multiple sensor nodes trying to transmit data to the edge gateway at the same time over the same UDP port.

F. Effect of Microservice Latency

Focusing on the microservices, we observed that the InfluxDB microservice was consuming significant resources, but the single CPU core usage did not exceed 60%, and the RAM usage was low. Also, the other microservices and the software adapters did not consume much resources. Interestingly, we observed that an increase in the number of sensor nodes increasingly loaded the UDP adapter. As a result, the updated reading arrived at the InfluxDB with an increasing delay for each sensor node added. This delay is due to the UDP adapter handling data from multiple sensor nodes sequentially. Hence, the Grafana plot-based dashboard also showed that the sensor data points in the sensor data plots were updated with delay. We could add only 80 sensor nodes in our experiment till the delays incurred were not sustainable and feasible. This delay can be eliminated simply by implementing multiple UDP Adapters in the SEGA architecture. Hence, much more sensor nodes can be connected to the same SEGA edge gateway device.

VII. Conclusion

The SEGA architecture proposed in this work addresses the industrial requirements of reliable and secure plant machinery monitoring. The system implements microservice architecture for the deployment of the services. We demonstrated an end-to-end edge gateway network to securely collect a machine's power and vibration data and autonomously determine its operational state within the edge at the gateway. Although there are hardly any tangible changes to the gateway device's performance in response to the inclusion of encryption modules, the relatively resource-constrained edge sensor nodes show an increase in network transmission delays by 84.12 ms. In addition to active encryption for securing the sensed data, several passive security measures are automatically associated with the SEGA architecture by the containerized microservices in use. In the future, we plan to incorporate online learning to learn the changing trends in the sensed data from various machines on a factory/plant floor. This would remove the dependence of SEGA on machine-specific datasets for determining machine status and health.

Acknowledgment

The work reported in this paper is partly funded by SERB/ IMPRINT-2 (Grant Ref. No.: SERB/F/12680/2018-2019;IMP/2018/000451), which the authors gratefully acknowledge.

References

[1] J.-S. Fu, Y. Liu, H.-C. Chao, B. K. Bhargava, and Z.-J. Zhang, "Secure data storage and searching for industrial iot by integrating fog computing and cloud computing," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4519–4528, 2018.

[2] T. Wang, P. Wang, S. Cai, Y. Ma, A. Liu, and M. Xie, "A unified trustworthy environment establishment based on edge computing in industrial iot," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6083–6091, 2019.

[3] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service candidate identification from monolithic systems based on execution traces," *IEEE Transactions on Software Engineering*, 2019.

[4] L. Qi, C. Hu, X. Zhang, M. R. Khosravi, S. Sharma, S. Pang, and T. Wang, "Privacy-aware data fusion and prediction with spatial-temporal context for smart city industrial environment," *IEEE Transactions on Industrial Informatics*, 2020.

[5] X. Wang, S. Lu, W. Huang, Q. Wang, S. Zhang, and M. Xia, "Efficient data reduction at the edge of industrial internet of things for pmsm bearing fault diagnosis," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–12, 2021.

[6] T. Wang, H. Ke, X. Zheng, K. Wang, A. K. Sangaiah, and A. Liu, "Big data cleaning based on mobile edge computing in industrial sensor-cloud," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 2, pp. 1321–1329, 2019.

[7] C.-W. Hsu, Y.-L. Hsu, and H.-Y. Wei, "Energy-efficient edge offloading in heterogeneous industrial iot networks for factory of future," *IEEE Access*, vol. 8, pp. 183 035–183 050, 2020.

[8] W. Sun, J. Liu, and Y. Yue, "Ai-enhanced offloading in edge computing: When machine learning meets industrial iot," *IEEE Network*, vol. 33, no. 5, pp. 68–74, 2019.

[9] K. Kaur, S. Garg, G. S. Aujla, N. Kumar, J. J. Rodrigues, and M. Guizani, "Edge computing in the industrial internet of things environment: Software-defined-networks-based edge-cloud interplay," *IEEE communications magazine*, vol. 56, no. 2, pp. 44–51, 2018.

[10] P. K. Deb, S. Misra, T. Sarkar, and A. Mukherjee, "Magnum: A Distributed Framework for Enabling Transfer Learning in B5G-Enabled Industrial-IoT," *IEEE Transactions on Industrial Informatics*, 2020.

[11] L. Ren, Y. Liu, X. Wang, J. Lü, and M. J. Deen, "Cloud-edge based lightweight temporal convolutional networks for remaining useful life prediction in iiot," *IEEE Internet of Things Journal*, 2020.

[12] J. Okwuibe, J. Haavisto, E. Harjula, I. Ahmad, and M. Ylianttila, "Sdn enhanced resource orchestration of containerized edge applications for industrial iot," *IEEE Access*, vol. 8, pp. 229 117–229 131, 2020.

[13] K. A. Abuhasel and M. A. Khan, "A secure industrial internet of things (iiot) framework for resource management in smart manufacturing," *IEEE Access*, vol. 8, pp. 117 354–117 364, 2020.

[14] Y. Tao, P. Xu, and H. Jin, "Secure data sharing and search for cloud-edge-collaborative storage," *IEEE Access*, vol. 8, pp. 15 963–15 972, 2019.

[15] R. Morabito, "Virtualization on internet of things edge devices with container technologies: a performance evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.

[16] B. Mishra and A. Kertesz, "The use of mqtt in m2m and iot systems: A survey," *IEEE Access*, vol. 8, pp. 201 071–201 086, 2020.

[17] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–29, 2019.

[18] N. Naik, "Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http," in *2017 IEEE international systems engineering symposium (ISSE)*. IEEE, 2017, pp. 1–7.

[19] M. A. Prada, P. Reguera, S. Alonso, A. Morán, J. J. Fuertes, and M. Domínguez, "Communication with resource-constrained devices through mqtt for control education," *IFAC-PapersOnLine*, vol. 49, no. 6, pp. 150–155, 2016.