An IoT Management System Using Fog Computing

Berkin Yildiran, Erel Ozturk, Necati Ozkent, Yagizcan Arslan, Ilker Korkmaz
Faculty of Engineering
Izmir University of Economics
Izmir, Turkey
Email: {berkin.yildiran, erel.ozturk98, necati.ozkent, yagizcan.arslan93}@gmail.com, ilker.korkmaz@ieu.edu.tr

***Abstract*—During the past few years, the usage of network-enabled devices has been climbing rapidly. With the continuous evolution of industrial automation and fabrication, new demands have arisen such as constantly monitoring the state of the devices or being able to immediately intervene in any possible emergencies.**

In this paper, an IoT device management system using the fog computing paradigm is presented. The system has also been supported by mobile and web applications. There are four main components of this system; edge devices to receive or send data through the local area network, a local fog node to provide data transmission between the cloud server and edge devices as well as storing data, a cloud server to establish the communication between a distant user and local fog nodes, a mobile device that can go online when required to access the mobile application or a computer that can go online when required to access the web application.

***Keywords*—IoT, fog computing, cloud, smart automation system, mobile application.**

I. Introduction

The Internet of Things (IoT) has a wide range of usage from smart home automation systems to industrial manufacturing systems. With the rapid growth of industry, requirements of the industrial systems such as emergency action, fast and reliable response time, and data confidentiality have also increased.

The Industrial Internet of Things (IIoT) provides a breakthrough in industrial production processes by the utilization of a large number of network-enabled devices [1]. Once the fog computing paradigm [2] is also included, some aspects of the system such as reliability, confidentiality, and quickness, have also been augmented by the promises of fog computing.

Fog computing works like a distributed cloud that resides in the local network and keeps the real data securely without exposing them to the internet. Fogs are located close to edge devices to provide fast data transfer between edge devices and the cloud.

In this study, an IoT device management system using fog computing, with two user-friendly interfaces, a mobile application, and a web application, is being introduced. The objective of the system is to augment the communication between users and edge devices in an industrial workplace. The system is actually a core infrastructure that enables IoT management using fog computing. The reason to use such a system in industrial workplaces is basically its scalability with extra components as well as the quick response time of fog computing. This system can be used in factories, hotels, IT departments of huge institutions, etc.

The main purpose behind this project is to introduce some features (and/or augment the existing ones) to systems in industrial manufacturing areas (i.e., factories) by taking advantage of the promises of fog computing. With the help of fog computing involvement to IoT, the following contributions are aimed to be made:

- by providing the flexibility to set up their network, database, or cloud, each user's privacy can be ensured,
- under the case of an emergency, users will be able to intervene instantly since the response is coming from the local fog node, not the cloud server. The reason behind the fact that users can intervene instantly is related to the response time. The response time is much lower because the latency is also lower in the fog computing paradigm [3].

Taking advantage of fog computing, a new layer of security, storage, communication, computation, networking, and control has been added to the system to make it more secure and reliable.

Besides, the points to be emphasized in this paper are as follows:

- An in-depth, detailed, and varied related work-study.
- A comparison section that will support, strengthen, and facilitate the understanding of these studies.

Although it is an IoT management system in general, it has been used in the fog computing paradigm, unlike other similar systems. The reason for using this paradigm is the advantages it provides in terms of speed and response time.

The rest of the paper is organized as the following structure: Section II explains the related works that have been evaluated about their pros and cons; Section III gives the design and implementation details of the proposed system; Section IV presents on the results; Section V discusses the critics of the authors and some possible future works; Section VI concludes the paper.

II. Related Work

There has been a significant amount of research for IoT device management systems. The common problem is the inaccessibility to devices from out of their range. Also, connectable device compatibility according to their protocols, data exchange with the server, and response time have been considered.

Within the scope of our research, the information gathered and its summarized comparison with our proposed system is presented in Table I. This table clarifies the characteristics of the subjects related to the research conducted.

TABLE I
COMPARISON OF RELATED PROJECTS

Related Work	Features		
	Technologies	*Remote Access*	*Intermediate Node*
Proposed System	Raspberry Pi, nodeMCU	Smartphone, Web Panel	Raspberry Pi
[4], [5]	Arduino BT, ZigBee	Android Phone	-
[6]	TI CC3200 Launchpad	Voice Call	-
[7]	Arduino Uno	Smartphone, Email	-
[8]	WiBro	Mobile Device	Smart Gateway
[9]	Raspberry Pi, Arduino WiFi Shield	Smartphone	-
[10]	Z-Wave, ZigBee	Smartphone	-

A home automation system allowing multiple users to control the appliances through an Android application has been introduced in [4] and [5]. Local hardware is configured to communicate between home appliances and the cloud. The mentioned system is connected to Google App Engine, and the communication between the server and the local hardware is handled via Google Cloud Messaging. Although the use of third-party applications such as Google App Engine and Google Cloud Messaging has eased the implementation phase, the use of such applications might reduce the stability of the system.

Kodali et al. [6] present an IoT based home automation system providing remote security. The system can be controlled from the local area network. Also, while the user is away from home, any triggered sensor can alert them without requiring an internet connection or any interface. This system comes in handy where the internet infrastructure is unstable or unreachable as it alerts the users via a simple voice call to their mobile devices. However, the work limits the ability to manage devices from a distance; so the system needs an internet connection to be more effective on managing and monitoring from outside.

Dutta and Roy [7] present a cost effective smart building system. The system is controllable by a smartphone or a computer. Regarding the system design, it was aimed to include automation as well as reducing unnecessary energy consumption and human effort by incorporating IoT, fog, and cloud. The proposed solution is built by using Arduino Uno, sensors, Bluetooth module, WiFi module, fog gateway server, and the cloud platform. It can also inform users via email in case of a failure of an appliance. Building such a system can be challenging by the means of expenses. Placing a separate fog in each local area would be more expensive than placing intermediate nodes that transmit device data to the corresponding local fog. Also, it avoids extra complexity.

Verma and Sood [8] propose a human-interactive healthcare system. Patients' data (such as heartbeat, oxygen saturation, etc.) are sent from edge devices to a cloud server via fog nodes. In the cloud layer, a decision-making layer is included for general emergencies such as low heart rate and low oxygen saturation. In case of any other emergencies, a responder (e.g., doctor) monitors the data in the fog layer. If the cloud layer is somehow inaccessible, the responder will be able to monitor the patients' data manually through the fog layer. The computation and the data storage services would stop working in the case of an outage in the cloud. Also, communicating via the internet despite being in the coverage area of local fog nodes is a disadvantage since the latency increases.

Krishnan et al. [9] use fog computing to solve some of the problems that can be demonstrated by increasing the number of network devices. In this way, low latency, widespread geographical distance, mobility, and a very large number of nodes can be provided. Fog computing is used as there may be too many sensors and areas. It uses Arduino and Raspberry Pi while uploading the data received in certain periods to the cloud. In the temperature measurement scenario, the temperature sensors and WiFi Shields that send the data are connected to Arduino. The data received with WiFi adapters are stored and processed by all Arduino devices with the Raspberry Pi and then sent to the cloud. Users can monitor these data, which are coming from the cloud. In a scenario where temperature values are measured and sent to the cloud periodically, the same temperature values are held instead of being sent. If the temperature value changes, only this data will be sent, and the same temperature values as before will be predicted by the cloud.

Regarding SmartThings, Samsung is taking a major step to be the core of the smart home today. In [10], SmartThings has a cloud platform, a hub that is a gateway or home controller, and a client application. This application allows users to control, automate, and monitor compatible technologies in their homes via a smartphone. SmartThings can communicate with devices by connecting to the internet router at home. This system works with compatible protocols such as Z-Wave [11], Zigbee [12], and IP-accessible devices. Also, this system only works with devices specified by Samsung. Therefore, the user cannot introduce incompatible protocols to the system.

Yi et al. [13] discuss that the following three topics heavily need the use of fog computing: augmented reality, content delivery and caching, and mobile big data analytics. Regarding augmented reality, since the human brain can notice such short

delays as 10-15 milliseconds, it is crucial to deliver the scene (or sounds) as fast as possible. The key solution to that is fog computing due to its low latency. Regarding content delivery and caching, in a situation where there is a need for constant data delivery, sending requests over and over to a single server can be exhausting in terms of efficiency and price. In such a scenario, using fog computing can be lifesaving since all these requests are going to be shared among fog nodes and processed to a main (cloud) server. Regarding mobile big data analytics, in scenarios where there is big data processing, the system suffers from the cloud's high latency and computational power issue. Instead of transmitting data to the cloud to process and send it back to the user; with the help of fog nodes, many things can be saved such as latency, storage, process power, etc. So, in [13], an example of a large-scale environment monitoring system is given, and it is explained how the local fog nodes can process the corresponding local area's data and then transfer it to the cloud.

Greengrass [14] has an open-source design. Customers can program their devices to act locally on the data they generate. Once the software development completes, Greengrass remotely operates and manages the edge devices without a firmware update. Greengrass provides remarkable services such as offline operations, secure and encrypted communication, and data filtering. Regarding offline operations, Greengrass devices can work locally on the data they generate and respond quickly to the events without any cloud communication. Apart from that, devices can operate normally even if they are offline. When devices reconnect, Greengrass synchronizes the data on devices to the cloud system. Regarding secure and encrypted communication, all communication between both local and cloud devices is encrypted. Data will never exchange without a proven identity. Regarding data filtering, Greengrass passes the data through a filter system before sending it to the cloud. This filtration greatly reduces the data transmitted to the cloud.

Oracle Cloud Observability and Management Platform services [15] are grouped under four main headings as Application Performance Monitoring, Log Analytics, IT Analytics, Infrastructure Monitoring. Application performance monitoring helps to monitor incoming data and system status. This data is forwarded to the system development and operations teams. Thus, end-user data, application performance, and system logs are collected in an organized data set. Log analytics examines and analyzes the log data in this set. In this way, an operational view is formed. It helps to take fast and effective actions. IT analytics and infrastructure monitoring analyze and classify the performance data in the created data sets. These two services are mostly used to inform IT experts and system administrators in areas such as accessibility and system health.

III. System Design And Implementation

In this section, system design and implementation details of our proposed system are elaborated. Mainly, the system architecture and its communication infrastructure are provided. Furthermore, the mobile and web application interfaces are presented.

The proposed system gives its users the ability to manage and monitor fogs, places, and devices locally and remotely. With the help of fog computing, users do not need to log into the cloud, thus they do not need to be online to manage their infrastructure. This enhances user's privacy over the network since the data is stored locally. Moreover, in case of any emergency triggered by any of the edge devices, the involved parties can be alerted instantly thanks to the low latency of fog computing.

A. System Architecture

The system is designed to adapt to different areas such as industrial use and personal use. Overall, our system design looks like in Fig. 1 and Fig 2.

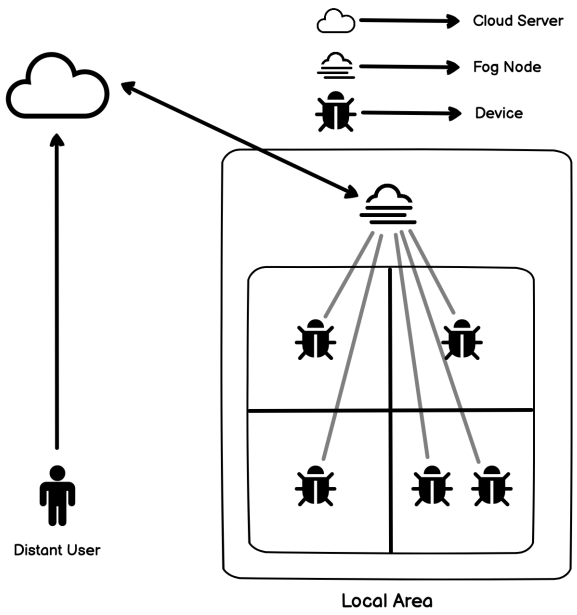

Fig. 1. Overview schema of the system with remote access.

The communication between fog and cloud is done via Hypertext Transfer Protocol Secure (HTTPS) [16] to secure data transmission. When a place, fog, or device has been added, updated, or deleted the given information is shared with the cloud to maintain the integrity between cloud and fog. The fog synchronizes the device data it receives every 15 minutes with the cloud. The data is transmitted in JSON format [17].

Intermediate nodes will be installed in places with high device density. These intermediate nodes will be tasked with connecting the edge devices in the corresponding part of the local fog node. They will transmit the incoming data bidirectionally via Hypertext Transfer Protocol (HTTP) [18] to decrease the performance cost of HTTPS as concluded by Naylor et al. [19] from and to the local fog node. In other words, they will be a bridge between edge devices and fog nodes. Using these intermediate nodes, the structure will be

established to prevent any possible overload by reducing the number of links a local fog node establishes.

In scope of the proposed system, only edge devices with a wireless protocol will be supported, wired connections will not be supported.

As can be seen in Fig. 1, if the user attempts to reach edge devices within the local area network, the user will not need an internet connection and will be able to access devices via that local fog node. On the other hand, if the user wants to reach devices while away from the local area network, the user must have an internet connection to communicate with the cloud server. The cloud server will connect the distant user to edge devices.

Device manipulations (create, edit or delete) can only be accomplished when connected to the local area network. On the other hand, while accessing the system remotely, only the current state of the system, data from the devices and data graphics can be observed.

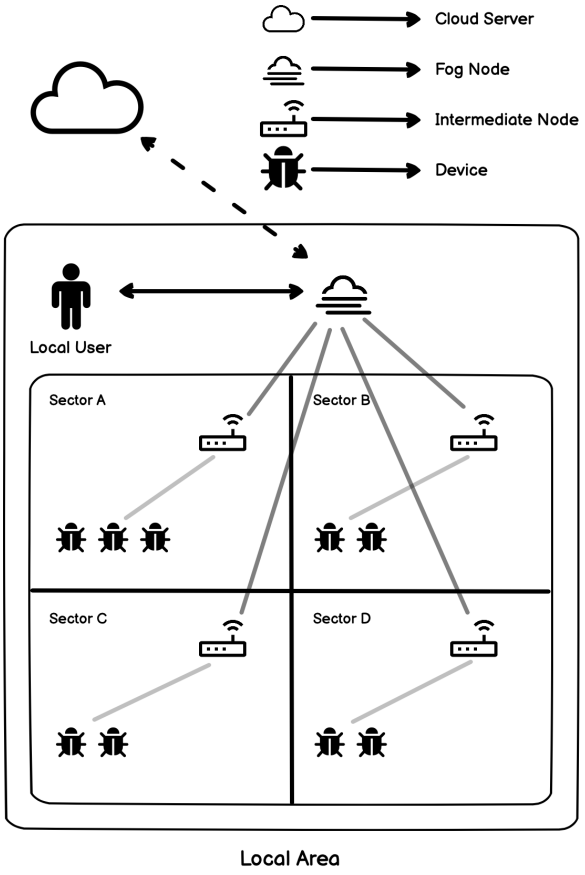

Fig. 2. Overview schema of the system in local area.

With the help of the routing capability of the fog node in Fig. 2, the system will be connected to the internet to get requests from the cloud and synchronize with the cloud. Intermediate nodes and edge devices can still be offline. The user in local area can modify the system directly without the need of internet connection. Later, anytime when the fog re-synchronizes its data with the cloud, the modification on the system will be available to the distant remote users.

B. System Infrastructure

The proposed system consists of a REST [20] API, developed with PHP [21] and Laravel Framework [22], to be consumed by the web and mobile applications and the devices. The data is stored on a MySQL [23] database but can be exchanged easily with other Relational Database Management Systems (RDBMS) supported by Laravel [24].

General actions such as managing places, fog, and devices as well as logging into the system are shown in the use case diagram in Fig. 3.

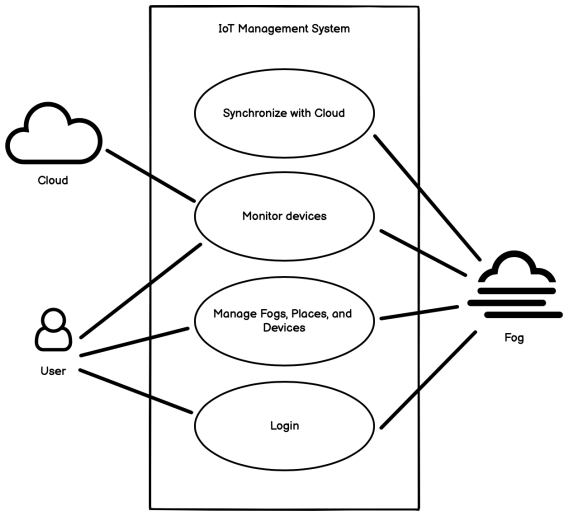

Fig. 3. Use case diagram of the proposed system.

Laravel Framework provides an easy way to manage environments such as fog and cloud via .env files. When setting up a fog, user credentials are entered into the aforementioned .env file, and an Artisan Command [25] is run in the fog to receive a Bearer Token [26] from the cloud. Bearer Tokens are used by the API consumers to authenticate the corresponding user. They are generated by the API and stored in the database.

Synchronization between the fog and cloud is done via a Facade design pattern [27] sending appropriate requests to the cloud with Laravel's HTTP Client while adding the Authorization HTTP Header [28] containing the Bearer Token to the request to authenticate in the cloud.

While implementing the database, version 4 Universally Unique Identifiers (UUID) [29] is used instead of auto-incremented integers to establish uniqueness inside the system. With the use of UUID, primary keys of fields generated during synchronization are preserved. A version 4 UUID is a randomly generated 128-bit long alphanumeric label, guaranteeing uniqueness across space and time by providing a total of 2^{122} possibilities.

The database tables and relationships are created using Laravel's migration system. Places, fogs, and devices have a three-way relationship with each other:

- Place has-many Place
- Place has-one Fog
- Place has-many Device through Fog
- Fog has-many Device

Before using the system, the admin should initially create a place and assign a fog and a number of devices belonging to that place.

When setting up a device, its MAC address and expected parameters must be entered correctly to receive and store the data incoming from device requests. When a device sends its data to the fog, the unique identifier of the user is appended to the path of the URI, and the MAC address is also sent within the request so that the fog recognizes the device.

C. Hardware Installation

In general, appliances that have been used throughout this study can be divided into three groups:

1) Cloud Server: A virtual machine with 2GB RAM, a burstable portion of an Intel v-CPU, 50GB SSD, and 1TB of bandwidth from DigitalOcean [30] has been used as a cloud server.

Apache [31], MySQL, and PHP were installed to run the API server.

2) Fog Node: Raspberry Pi 3 Model B+ with 4GB RAM has been used as a local fog node. Inside Raspberry Pi, a fog server has been installed just like a mirror of the cloud server. The fog has been provided with an internet connection via its WiFi module. A 32GB SD card has been used with its adapter as the storage unit.

Apache, MySQL, and PHP were installed to run the API server on fog node.

3) Edge Devices: The system has been experimented with two basic sensors; a DHT11 sensor that measures the temperature and humidity along with an HC-SR04 ultrasonic sensor that measures the distance. A nodeMCU CH340 (ESP32E) module has been used to collect the sensor data and send it to the server. DHT11 has been powered by the 3V3 pin of the nodeMCU. Since HC-SR04 requires a 5V power supply and nodeMCU has only 3V3 output, an Arduino Uno R3 module's 5V output has been used as an external power supply for HC-SR04. The integrated circuit is shown in Fig. 4.

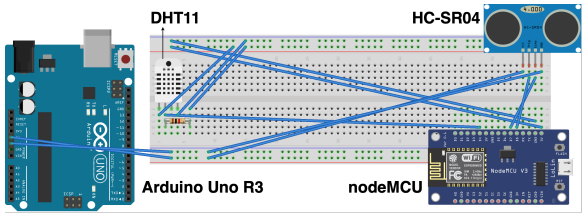

Fig. 4. Circuit scheme of the implemented hardware.

The nodeMCU module has been programmed in a way that the device data can be posted only if at least one of them has changed. In other words, it will not post if no device data is changed. The posting process is a simple HTTP POST request. NodeMCU will read the devices' data and decide to post or not once per 5 seconds.

D. User Interfaces

Two different user interface (UI) parts are provided to users; a mobile application and a web application. Even though the platforms are different, both applications are designed in such a way that they work identically.

A device that can go online when necessary is required. On the mobile end, the user interface is designed using Flutter Framework [32]. On the web end, the user interface is designed on Angular Framework [33] as a single page application [34].

It is important whether the user is logging in via cloud or fog, in other words, whether from a distance or inside the LAN of the local fog node. Users are not allowed to manipulate any information while they are away from their local area, they can only view the data of the devices or control them if possible. When the user logs in, the system will acquire the state of the WiFi connection. If the WiFi is connected to a network where a fog exists, the system will set the base URL to the local fog node's server URL. If the WiFi is not connected, the system will set the base URL to the cloud server's URL. Therefore, it is possible for the system to understand whether the user is inside the local area or not.

In Fig. 1, a distant user that accesses the system through the cloud server has been shown. Since the device needs to reach the cloud, it requires an active internet connection to log in to the system. In Fig. 2, internet connection is not required as long as the device is connected to the LAN.

The communication between user interfaces and the API is done via HTTP requests. Once the user logs in, a Bearer Token is provided by the API to authenticate the client for further requests. Once logged in successfully, the user will be able to monitor, or control the devices. While creating the devices; device information, such as MAC address or parameters, must be entered correctly. Otherwise, the system will not be able to communicate with that device. Once a device is created successfully, it will send data periodically using the HTTP POST method and the UI will visualize the data as graphs that can be filtered as daily, weekly, or monthly along with some important features such as minimum, maximum, and average values.

Moreover, it is possible to integrate the hierarchical place structure into the user interface. Therefore, it will be possible to filter the devices based on where they are located. In Fig. 5, a sample place structure is illustrated. As seen, a factory is divided into four sectors and different devices have been created in each sector. The user can create a new place under the place. Furthermore, the user can add any device (in plug-and-play manner) in these places.

IV. Results

In this section, some test results of the proposed system have been illustrated. In Fig. 6, response times of the cloud and fog servers are given respectively. In Fig. 7, implemented mobile interface screenshots have been provided.

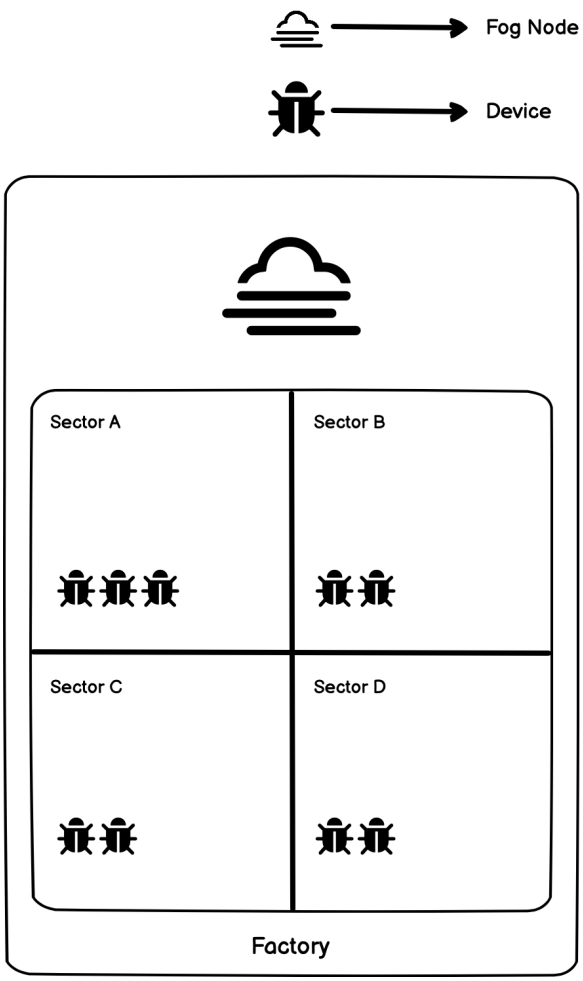

Fig. 5. A sample hierarchical place structure.

Performance testing of the system is done via Apache JMeter [35] tool. The purpose of this test was to compare and provide some solid proof of fog computing being faster than cloud computing. Regarding the test, 1000 sample device data is queried via HTTP requests and corresponding transmission times of the responses (i.e., latency results) are measured. To compare the cloud and fog responses, average latency results for every consecutive 50 HTTP requests within the set of entire 1000 requests are shown in Fig. 6. In Fig. 6, the latency results in milliseconds have been visualized for cloud server and fog server, respectively.

Looking at the test results in Fig. 6, it can be concluded that in fog computing, responses are approximately two times faster than cloud computing. While making this inference, irrelevant spikes on the graphs have been ignored. As an evaluation of the results, it can be clearly stated that using fog computing, a remarkable advantage on response time has been provided to the users.

In Fig. 7, the real device data graphs have been provided. The device is the one that has been clarified in Section III-C. As seen in these charts, the proposed system has been realized

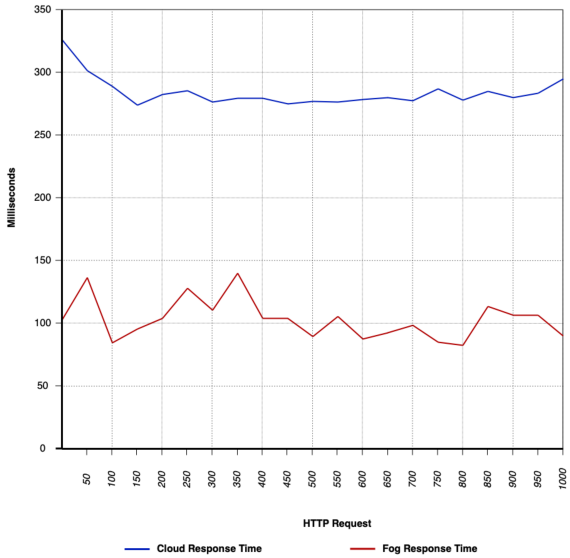

Fig. 6. Cloud and Fog response times.

successfully and an appropriate mobile application has been implemented to monitor its IoT environment. The data that is coming from the device is synchronizing with the UI of the mobile app software instantly. The corresponding output graphs are being consistently drawn by the UI.

Taking the cost of the proposed system into consideration, it would be appropriate to use this system for factories, huge hotels, or larger businesses instead of homes. The system is more costly than other services available in the market because of the extra fog layer included in the system. Any intermediate node can be added in plug and play manner to expand the system into a hierarchical structure.

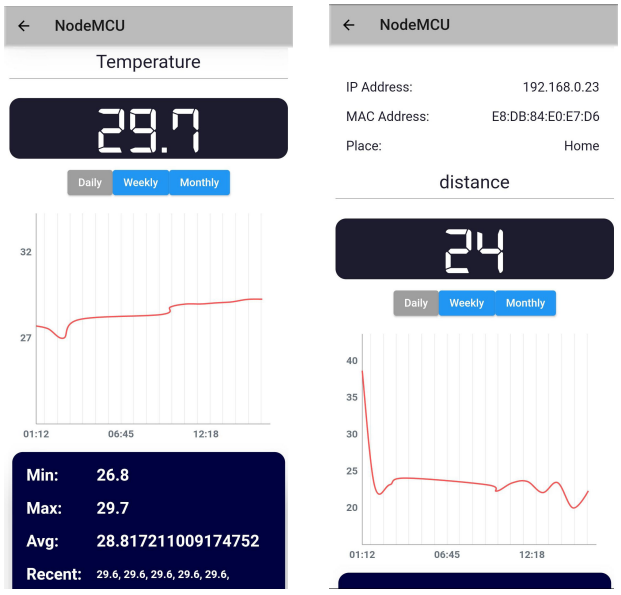

Fig. 7. Sample temperature and distance charts on mobile application.

We did not include manageable devices (air conditioner, led

lights, etc.) in our system because it went beyond our purpose. Our actual focus was to establish a fog-based IoT management system with a wider spectrum.

Despite using a cross-platform framework in the mobile implementation, which is Flutter, the UI is designed according to the Android interface. However, the UI can be adjusted to the iOS interface in the future.

V. Discussion And Evaluation

The proposed system has many advantages as well as some disadvantages. Regarding the most remarkable advantage, we can underline the response times being short with the help of fog support. We have provided a fast, reliable, and secure system to the users. We have also stated that the most significant reason for these aspects is the fog computing paradigm.

Furthermore, integrating a new hardware component to the proposed system is easy since it is plug and play. By this way, third-party software and hardware integration can also be done simply. As an example of this, an alert API can be implemented such that whenever the monitored value of a device exceeds a predefined threshold, the system can trigger an alert. Considering the hardware, it would not consume extra time to implement since the device structure of the system is plug and play. Considering the software, implementing an alert API would be a quite effective and time-saving solution. The structure of such an API is exampled with the following pseudo-code:

Algorithm 1: Trigger alert

```
Function checkTrigger(double threshold):
    if device.value >= threshold then
        notifyUser()
    end
End Function
```

On the other hand, the most remarkable disadvantage of the system is its overall cost. Such an IoT system with fog computing needs some additional hardware components. Therefore, the cost of the entire system increases.

VI. Conclusion

With this system, we combined fog computing, one of the new doctrines of computer networks literature, with IoT, which is a part of today's fast and smart technological life. We built a fast, reliable, and independent platform that can be deployed in huge facilities where fast transactions on many sensor data are required such as the petrochemical industry. Users can connect to system either from mobile or web platforms. Through the UI, they can perform many operations such as examining data of their devices, adding, updating, or deleting devices. The daily, weekly, or monthly changes of the data received by the system are shown to the users by means of tables and graphs.

In the future, many different sensor devices can be included in the system. Tests using larger data can be performed on the functionality of the system and the reaction of the system can be evaluated.

Acknowledgment

The authors would like to thank the reviewers for their useful comments on the paper. The authors also thank Metin Ahmet Ustabasi for his help on the proofreading stage of the paper.

References

[1] K. Tange, M. De Donno, X. Fafoutis, and N. Dragoni, "A systematic survey of industrial internet of things security: Requirements and fog computing opportunities," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2489–2520, 2020.

[2] O. C. A. W. Group, "Openfog reference architecture for fog computing," *OPFRA001.020817*, 2017.

[3] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb)*. IEEE, 2015, pp. 73–78.

[4] A. Gurek, C. Gur, C. Gurakin, M. Akdeniz, S. K. Metin, and I. Korkmaz, "An android based home automation system," in *2013 High Capacity Optical Networks and Emerging/Enabling Technologies*. IEEE, 2013, pp. 121–125.

[5] I. Korkmaz, S. K. Metin, A. Gurek, C. Gur, C. Gurakin, and M. Akdeniz, "A cloud based and android supported scalable home automation system," *Computers & Electrical Engineering*, vol. 43, pp. 112–128, 2015.

[6] R. K. Kodali, V. Jain, S. Bose, and L. Boppana, "Iot based smart security and home automation system," in *2016 international conference on computing, communication and automation (ICCCA)*. IEEE, 2016, pp. 1286–1289.

[7] J. Dutta and S. Roy, "Iot-fog-cloud based architecture for smart city: Prototype of a smart building," in *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence*. IEEE, 2017, pp. 237–242.

[8] P. Verma and S. K. Sood, "Fog assisted-iot enabled patient health monitoring in smart homes," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1789–1796, 2018.

[9] Y. N. Krishnan, C. N. Bhagwat, and A. P. Utpat, "Fog computing—network based cloud computing," in *2015 2nd International Conference on Electronics and Communication Systems (ICECS)*. IEEE, 2015, pp. 250–251.

[10] D. Lee, "Internet of things: Smart home system," https://www.theseus.fi/handle/10024/161349, 2019, (accessed on 13 July 2021).

[11] "Z-wave mesh network protocol specification," https://www.silabs.com/wireless/z-wave/specification, (accessed on 20 January 2020).

[12] "Zigbee - zigbee alliance," https://zigbeealliance.org/solution/zigbee/, (accessed on 20 November 2020).

[13] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 workshop on mobile big data*, 2015, pp. 37–42.

[14] "Aws iot greengrass," https://aws.amazon.com/tr/greengrass/, (accessed on 13 July 2021).

[15] "Oracle cloud observability and management platform," https://www.oracle.com/tr/manageability/, (accessed on 13 July 2021).

[16] "Hypertext transfer protocol (http/1.1): Message syntax and routing," https://datatracker.ietf.org/doc/html/rfc7230, (accessed on 13 July 2021).

[17] "The javascript object notation (json) data interchange format," https://datatracker.ietf.org/doc/html/rfc8259, (accessed on 13 July 2021).

[18] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol–http/1.1," 1999.

[19] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the "s" in https," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 133–140.

[20] "What is rest," https://restfulapi.net, (accessed on 13 July 2021).

[21] "Php," https://www.php.net, (accessed on 13 July 2021).

[22] "The php framework for web artisans," https://laravel.com, (accessed on 13 July 2021).

[23] "Mysql," https://www.mysql.com, (accessed on 13 July 2021).

[24] “Database: Getting started - laravel - the php framework for web artisans,” https://laravel.com/docs/5.1/database, (accessed on 13 July 2021).
[25] “Laravel docs artisan,” https://laravel.com/docs/8.x/artisan, (accessed on 13 July 2021).
[26] “The oauth 2.0 authorization framework: Bearer token usage,” https://datatracker.ietf.org/doc/html/rfc6750, (accessed on 13 July 2021).
[27] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. USA: Addison-Wesley Longman Publishing Co., Inc., 1995.
[28] “The ’basic’ http authentication scheme,” https://datatracker.ietf.org/doc/html/rfc7617, (accessed on 13 July 2021).
[29] “A universally unique identifier (uuid) urn namespace,” https://datatracker.ietf.org/doc/html/rfc4122, (accessed on 13 July 2021).
[30] “Digitalocean – the developer cloud,” https://www.digitalocean.com, (accessed on 13 July 2021).
[31] “Apache,” https://www.apache.org, (accessed on 13 July 2021).
[32] “Flutter,” https://flutter.dev, (accessed on 13 July 2021).
[33] “Angular,” https://angular.io, (accessed on 13 July 2021).
[34] M. A. Jadhav, B. R. Sawant, and A. Deshmukh, “Single page application using angularjs,” *International Journal of Computer Science and Information Technologies*, vol. 6, no. 3, pp. 2876–2879, 2015.
[35] “Apache jmeter,” https://jmeter.apache.org, (accessed on 13 July 2021).