Scalable Incident Reporting Framework: A Sensor and IoT Research

Sidi Mohamed Ahmed Ghaly
Department of Electrical Engineering, College of Engineering, Imam Mohammad Ibn Saud Islamic University, Saudi Arabia | Ecole Normale Suprieure, Nouakchott, Mauritania
smghaly@imamu.edu.sa (corresponding author)

Mohammad Ali Kadampur
Department of Electrical Engineering, College of Engineering, Imam Mohammad Ibn Saud Islamic University, Saudi Arabia
mkadampur@imamu.edu.sa

Received: 1 March 2023 | Revised: 23 March 2023 | Accepted: 26 March 2023

ABSTRACT

The Internet of Things (IoT) is one of the most rapidly emerging technologies. It is observed that while many devices/machines get connected in an application, it is a challenge for the IoT application designer to keep the application scalable. Scalability is the ability of a device/application to adapt to the changes in the environment and meet the changing needs in the future. The paper presents a layered IoT architecture and discusses issues related to the scalability of each layer. The best open-source technologies are explored. A novel system architecture of a scalable IoT framework is conceptualized in this paper. An application covering vehicle accident reporting is designed with the proposed framework. The application is tested in real-time using the standalone hardware and its ability to report the incidents is confirmed. The scalability metrics of the proposed framework are evaluated and the results are reported.

Keywords-IoT; scalability; sensors; computer networks; Message Queue Telemetry Transport (MQTT)

I. INTRODUCTION

Scalability refers to the ability of an application or a computing process to work with ease at varying capacities of increased or decreased inputs and outputs. During the recent years the IoT technology has revolutionized the communication between physical devices, making computers sense information without human intervention. The number of devices that are getting connected is ever-increasing, along with the volume of data being processed. One of the estimations suggests that the number of devices ("Things") that would be connected by 2025 is going to be around 30.9 billion [1]. Therefore, as the number of devices to be handled increases, an IoT application that is developed to handle a particular number of devices will become inefficient or obsolete sooner or later. In order to design any IoT application, it is important to consider current and future workloads, data volumes, number of sensors, security patches, network traffic, etc. [1-5]. What works in the small scale should also work in the large one. However, achieving this is a very challenging task. In this paper, the dimensions of IoT scaling are investigated and a case study on accident management using IoT is presented.

II. UNDERSTANDING IOT SCALING

In order to understand IoT scaling, the basic IoT reference model needs to be revisited (Figure 1). There are 7 conceptual layers in any IoT application. Layer 1, is the perception layer, which consists essentially of sensors and the associated hardware. With the booming areas of application of IoT, including agriculture, healthcare, industrial manufacturing, smart homes, offices, cities, and many more, the number of sensors in the perception layer is always growing. Also, already wired sensors are going to be replaced by upgraded versions. Any performing IoT infrastructure must be able to accommodate these developments in the perception layer by properly recognizing device registry and functionality. Vertical scaling refers to incorporating a better version of the device with more capacity and horizontal scaling refers to incorporating more of the same devices in the system [1, 6]. Layers 2 and 3 are basically communication and computing layers. The most important component of this network layers is the Gateways. Gateways moderate control and data between the endpoints of IoT devices and the edge/cloud [6, 7]. Important parameters such as the amount of data processing expected per device, frequency of data collection, and the types of analytical results expected are to be considered while choosing the gateways for IoT scaling. Gateways that provide multiple internet access (Ethernet, Wifi, global/ 3G/4G/5G/LTE) are preferred in this layer. The gateways that are compatible with rich mainstream protocols (such as MQTT, HTTP, Zigbee, LoRaWAN, Modbus) are best suited to make this layer scalable [6].

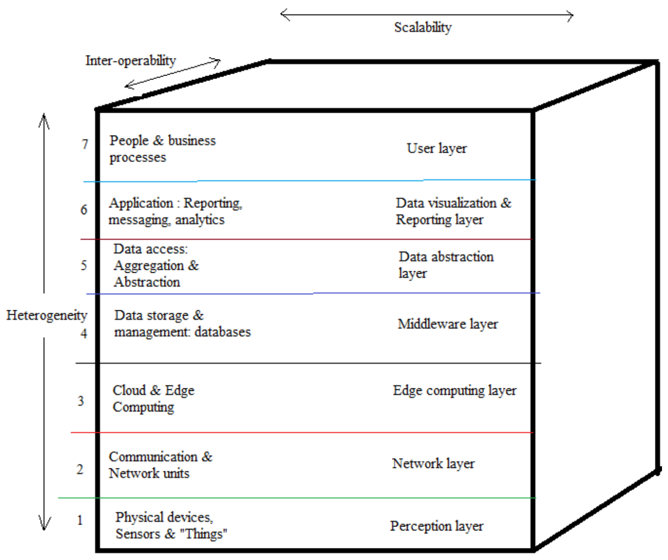

Scalability
Inter-operability
Heterogeneity
7 People & business processes — User layer
6 Application : Reporting, messaging, analytics — Data visualization & Reporting layer
5 Data access: Aggregation & Abstraction — Data abstraction layer
4 Data storage & management: databases — Middleware layer
3 Cloud & Edge Computing — Edge computing layer
2 Communication & Network units — Network layer
1 Physical devices, Sensors & "Things" — Perception layer

Fig. 1. The IoT layered reference model.

Layers 4 and 5 are basically data storage layers. The biggest challenge for scalability in databases is their "cardinality". Cardinality is the number of possible values in a database. A typical IoT deployment produces millions of values. For example, an IoT deployment with 5000 devices, each device having 100 sensors across 100 warehouse results in database cardinality of 5 millions and the number of tables goes on increasing. The table structure is not fixed. In order to address such scenarios, data storage must be carried out in a non-traditional way. No SQL databases or key-value pair databases fit into this. Zigsaw, Influx, Riak TS, mongoDB [8] are some of the preferred databases to make IoT scalable in this layer. Layers 6 and 7 provide a picture of the growing application users and the applications themselves.

Sectors such as agriculture, health, industrial manufacturing, hospitality, finance, transportation, and education have started using the benefits of IoT technology to make smart homes, smart cities, and smart life in general. These observations suggest that during IoT application development, scalability is an important factor to be considered. In this paper, a framework for designing a scalable application is discussed.

III. BACKGROUND AND RELATED WORK

In the past, in order to scale IoT applications, many different architectures have been proposed. The most important among them are cloud-based architectures, fog-based architectures, and, edge-based architectures [1-3]. The cloud-based technology is used in [2]. In this technology, every input data from the sensors has to reach the distant servers for processing. As billions of IoT applications keep sending data to the cloud servers, the server becomes heavily loaded and there will be delayed response. In [10], a kind of modification was done for the cloud approach and small capacity servers were dynamically triggered as middle nodes in the cloud. These dynamic middle nodes were named "fog servers" [9]. Experimental results in fog computing showed that the system performance is not always improved and may become worse sometimes [10]. In [11, 12] edge/cloud computing methods were used. In edge computing, instead of sending the sensor data to distant servers, they are processed in a machine nearest to the source (a machine at the edge of the network). This approach in edge reduces the latency and improves the IoT response time. The major players in contributing IoT scaling are Microsoft Azure, Amazon Web Services (AWS), and Google cloud [6, 8, 9, 13]. However, all these platforms follow the pay-as-you-go pricing model. In view of the needs of researchers and developers, an open-source scalable IoT framework is required. InciComm, is an open-source scalable IoT framework that is based on microservice architecture and provides improved performance. The following sections present the system architecture of the proposed InciComm scalable IoT framework. It is used to implement business rules and behavior of the use case application. The rule engine helps to map hardware pins as input and outputs during the design. Third-party services such as AWS can be included using a plugin module. As and when new devices/sensors arrive, they will be registered and the application starts performing as per the business rules specified in the rule engine. Thus, rapid development is aided by this scalable architecture.

IV. SYSTEM ARCHITECTURE

InciComm architecture is shown in Figure 2. The system architecture consists of components such as a gateway communicator, authenticator, and API modules. The system architecture contains an IoT broker module supporting MQTT, HTTP, Websockets, and CoAP protocols [6, 8]. The broker is an important module that acts as an interface between the input device (publisher) and the output (subscriber) device. A separate module called the device registry module exists to register the identified device with its device ID, authentication captcha, and other attributes. The architecture supports SQL and NoSQL databases, including SQLite, InfluxDB, MongoDB, and Riak T. In order to record, device metadata such as sensor type, units of data collection, etc., a metadata event manager module is included. The shadow module creates a persistent virtual version of the real device. Device shadow helps application development using REST APIs. The rule engine is another important component. It is used to implement business rules and behavior of the use case application. The rule engine helps map hardware pins as input and outputs during the design. Third-party services such as AWS cloud services can be included using a plugin module. As and when new devices/sensors arrive in the applications, they will be registered and the application starts performing as per the business rules specified in the rule engine.

V. REAL-LIFE USE CASE

Figure 3 shows the use case diagram of the application. In this use case, it is intended to use the framework to design an application that reports vehicle accidents in real-time [14, 15]. Device installation, application registration, authentication, sending notifications, identifying the nearest rescue responders, and identifying false positives to close the event cycle are some of the important use cases in the design. Haversine's algorithm [15, 16] is used to find the nearest best location of the rescue responders.

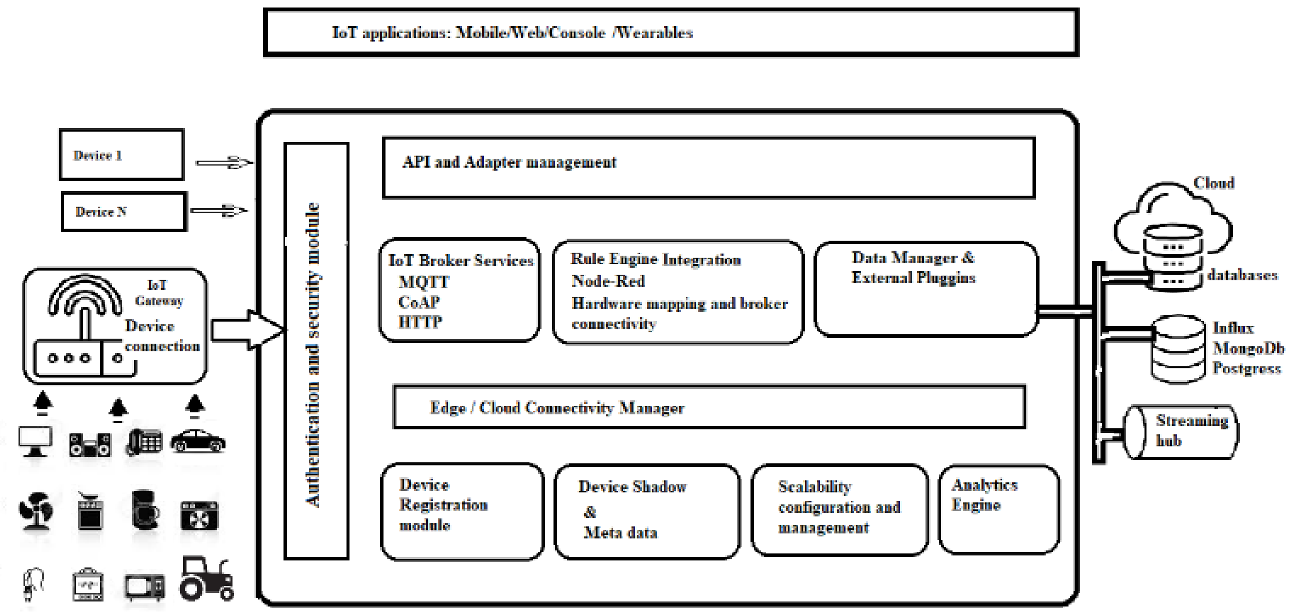

IoT applications: Mobile/Web/Console /Wearables
Device 1
Device N
IoT Gateway
Device connection
Authentication and security module
API and Adapter management
IoT Broker Services MQTT CoAP HTTP
Rule Engine Integration Node-Red Hardware mapping and broker connectivity
Data Manager & External Pluggins
Edge / Cloud Connectivity Manager
Device Registration module
Device Shadow & Meta data
Scalability configuration and management
Analytics Engine
Cloud
databases
Influx MongoDb Postgress
Streaming hub

Fig. 2. InciCom system architecture.

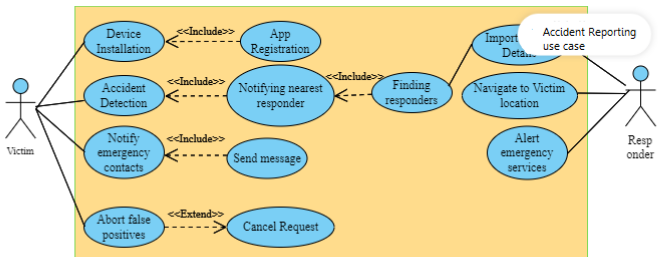


Fig. 3. Use case diagram of an accident reporting system.

The flow of the application starts with the user installing the hardware on the vehicle and registering in the App. The user provides information about device ID, customer ID, car/vehicle ID, service ID, mobile numbers, etc. The system authenticates and sends an email with a valid username and password credentials to log in and use the application. The application becomes active automatically whenever the vehicle starts moving or is made active manually. The active device streams heartbeats to the server. The system is designed in such a way that it reports only the worst-case scenario (i.e. accident). The broker is programmed to provide QoS 1 [17]. A time series with an unstructured database schema is used to store the streaming data. The vehicle health messages however are published (sent) at phased intervals as required by the user service demands. The mounted hardware prototype is shown in Figure 9 and App instances in Figure 10.

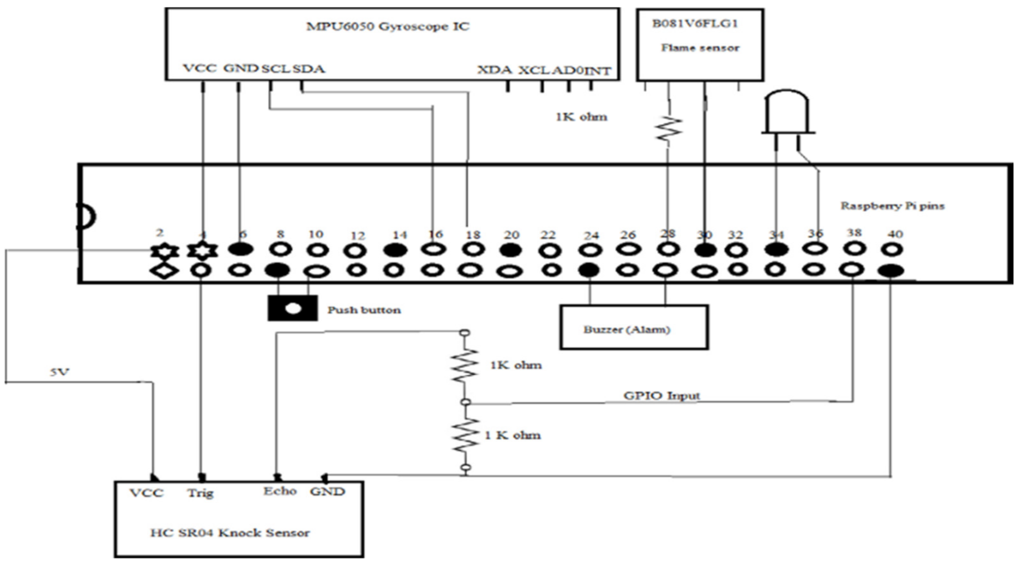


Fig. 4. PIN details and connection diagram using the Raspberry Pi.

VI. IMPLEMENTATION DETAILS

This section provides a high-level description of the implementation of the use case in the InciComm IoT platform. As a proof of the concept, the implementation is carried out using a Raspberry Pi mounted with sensors. The hardware connectivity schema and the actual circuit are shown in Figures 4 and 5, respectively. Three types of sensors were used to sense events such as an accident. The sensor HC-SR04 for distance measurement (S_d), the sensor MP6050 for sensing abnormal gyrometric changes (S_g), and the sensor B081V6FLG1 [18, 19] for sensing fire occurrences (S_f). An accident (A) is defined as a combined abnormal output of these sensors:

$$A = \Psi(Sd, Sg, Sf) \qquad (1)$$

If S_d becomes zero at any time, it signals an incident of touch/impact. The intensity peak at the time of reduced distance indicates the accident. The gyroscopic parameter Sg measures the tilt in the planes and S_f signals the occurrence of fire/sudden rise in temperature. The accident parameter A is calculated based on the majority vote of S_d, S_g, and S_f.

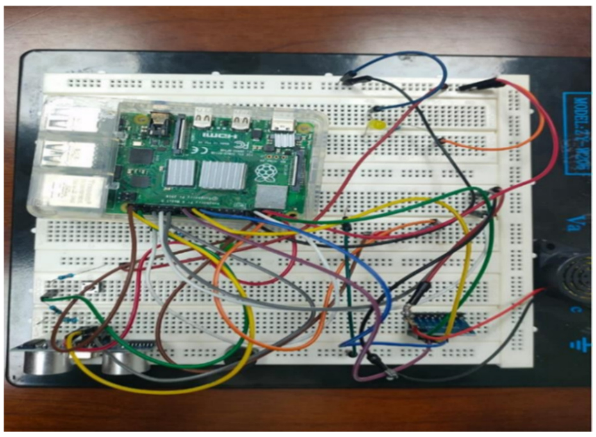


Fig. 5. Accident reporting hardware with Raspberry Pi and sensors.

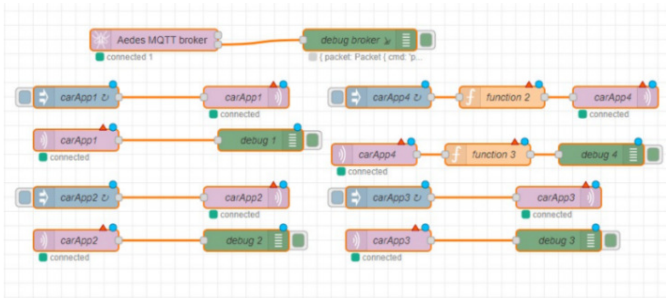


Fig. 6. Node-red rule engine flow

A Python [8] script to sense the sensor data on board the raspberry Pi was written, tested, and saved in a file. The program reads these sensor inputs and computes the accident-reporting parameter. An InciComm account is used to log in to the IoT platform. Devices are created in the framework by using device palletes and linking the device ID with the customer ID, access tokens, and service ID details. Node-red is used as a rule engine (Figure 6) to map the Raspberry Pi pins and other hardware. A dynamic MQTT broker is used to publish and subscribe messages in the cloud/edge networks. Any number of devices can be scaled up and down in the system using this broker. Figure 7 shows the scalable application development block diagram of the framework.

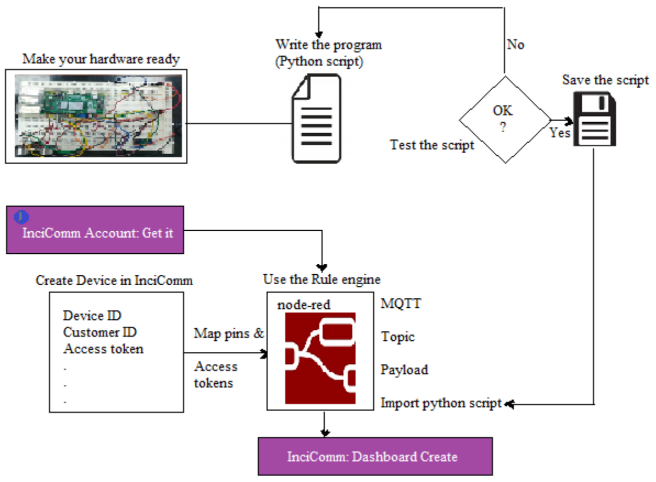
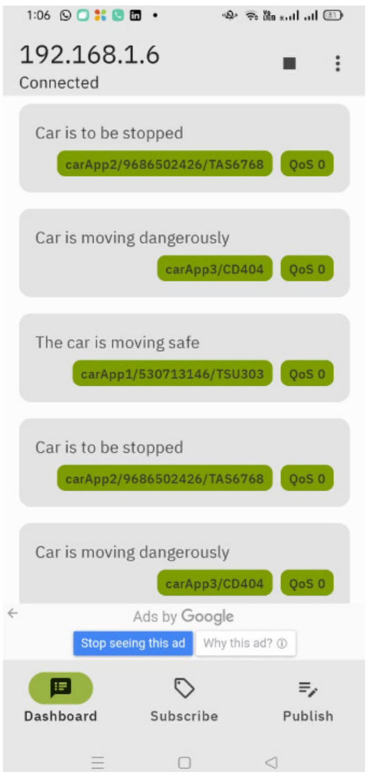


Fig. 7. Scalable application development flow.

VII. EXPERIMENTAL RESULTS

Initially, simulations were conducted using Node-red and MQTT broker. Three hypothetical cars were used. The payload and topics are shown in Figure 8.



1:06

192.168.1.6
Connected

Car is to be stopped
carApp2/9686502426/TAS6768 QoS 0

Car is moving dangerously
carApp3/CD404 QoS 0

The car is moving safe
carApp1/530713146/TSU303 QoS 0

Car is to be stopped
carApp2/9686502426/TAS6768 QoS 0

Car is moving dangerously
carApp3/CD404 QoS 0

Fig. 8. Payload and topic arrivals in hypothetical car tests.

The MQTT server offers three QoS levels, i.e. Q0, Q1 and Q2 [17]. Tests were conducted using Q0. The pay loads (messages) were exactly received for the respective topics. The messages were streamed at intervals of 10s and the broker was found to be working properly. Figure 10 shows instances of these simulations.

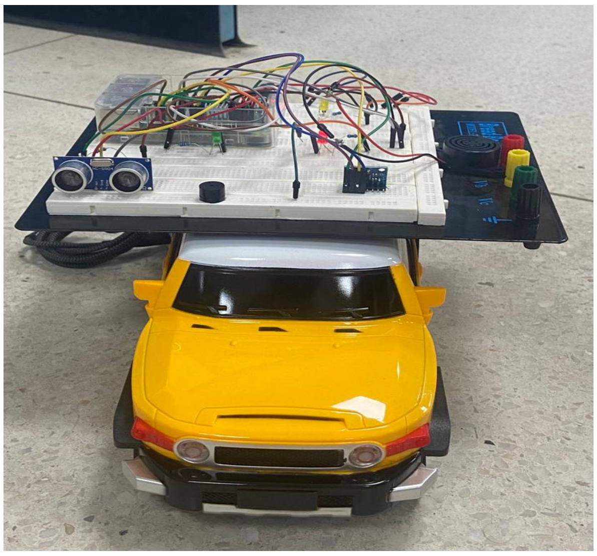


Fig. 9. The mounted prototype of the carApp hardware.

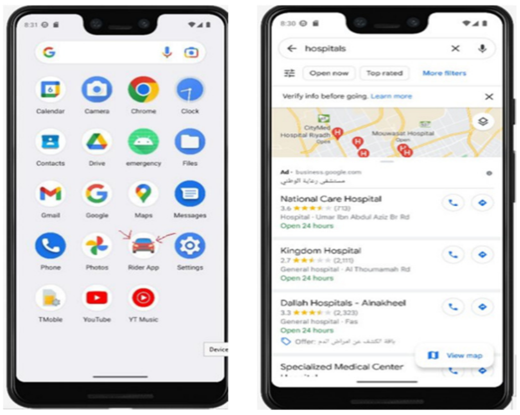


Fig. 10. Instances of real-time App usage and location tracking.

TABLE I. PAYLOADS AND TOPICS IN THE HYPOTHETICAL TESTS

Car No	Payload and topics	
	Payload	**Topic**
1	Car is to be stopped	carApp/9686502426/TAS56768
2	Car is moving dangerously	carApp/CD404
3	Car is moving safely	carApp/530713146/TSU303

As a second experiment, a real-time hardware device was mounted on an experimental car and its locations were traced on the carApp. At different locations, the experimental car was made to crash/tilt and the coordinates of the accident spots were shown on the map. The application was able to send to the nearby rescue responder (police and hospital) successfully.

VIII. DISCUSSION

The objective of this research was to provide a scalable IoT platform for designing IoT applications. A case study of an IoT application for reporting vehicle accidents was designed using the proposed scalable platform and its scalability was tested. The scalability tests and measured values are reported in Table II which lists the Key Performance Indices (KPI's) and the measured values.

Scalability testing is conducted in order to know the response of the system when there are workload variations. In this proposed testing, Assetto Corsa Competizione (ACC) simulations [20] are used and 5 scalability metrics are evaluated (Table II). Device provisioning rate defines the number of devices that can come in into the service of IoT application [19, 21, 22]. A total of 200 devices per second are reported. Over The Air (OTA) update rate is found to be 40. The number of messages that are ingested into the system is 1000 messages per second. The number of messages that fail to reach their subscribers is measured by message failure rate metric and it is found to be 0.2%. The message latency or delay in reaching its subscriber is less than 40 millisecond seconds.

TABLE II. IOT SCALABILITY METRICS

Scalability tests and measured values	
KPI	**Measured value**
Device provisioning rate	200/s
Over The Air (OTA) update rate	40
Message ingest rate	1000/s
Message failure rate	0.2%
Message latency	<40 ms

IX. CONCLUSIONS

In this paper, a scalable IoT framework is discussed along with an applied use case on accident reporting. The proposed framework allows the dynamic discovery of devices and data management. The proposed solution offers scalability with respect to the device deployment, number of users/inputs, and computing environment. A selected list of scalability metrics was evaluated and the scalability of the proposed framework is confirmed. The paper discussed a case study application to accident reporting and demonstrated the development and testing of the same using the proposed framework. In future work, the case study application itself needs to be improved with improved accident parameter definition. The scalability of the framework needs to be tested with the deployment of different applications other than the current case study application deployment.

ACKNOWLEDGMENT

The authors extend their appreciation to the Deanship of Scientific Research, Imam Mohammad Ibn Saud Islamic University (IMSIU), Saudi Arabia, for funding this research work through Grant No. (221414020)

REFERENCES

[1] A. Javed, A. Malhi, T. Kinnunen, and K. Främling, "Scalable IoT Platform for Heterogeneous Devices in Smart Environments," *IEEE Access*, vol. 8, pp. 211973–211985, 2020, https://doi.org/10.1109/ACCESS.2020.3039368.

[2] H. Guo, J. Ren, D. Zhang, Y. Zhang, and J. Hu, "A scalable and manageable IoT architecture based on transparent computing," *Journal of Parallel and Distributed Computing*, vol. 118, pp. 5–13, Aug. 2018, https://doi.org/10.1016/j.jpdc.2017.07.003.

[3] S. Kubler, J. Robert, A. Hefnawy, K. Främling, C. Cherifi, and A. Bouras, "Open IoT Ecosystem for Sporting Event Management," *IEEE Access*, vol. 5, pp. 7064–7079, 2017, https://doi.org/10.1109/ACCESS.2017.2692247.

[4] G. N. Cristina, G. V. Gheorghita, and U. Ioan, "Gradual Development of an IoT Architecture for Real-World Things," in *2015 IEEE European Modelling Symposium (EMS)*, Madrid, Spain, Jul. 2015, pp. 344–349, https://doi.org/10.1109/EMS.2015.57.

[5] M. Tabaa, B. Chouri, S. Saadaoui, and K. Alami, "Industrial Communication based on Modbus and Node-RED," *Procedia Computer Science*, vol. 130, pp. 583–588, Jan. 2018, https://doi.org/10.1016/j.procs.2018.04.107.

[6] P. Waher, *Learning Internet of Things*. Packt Publishing, 2015.

[7] K. Ferencz and J. Domokos, "IoT Sensor Data Acquisition and Storage System Using Raspberry Pi and Apache Cassandra," in *2018 International IEEE Conference and Workshop in Óbuda on Electrical and Power Engineering (CANDO-EPE)*, Budapest, Hungary, Aug. 2018, pp. 000143–000146, https://doi.org/10.1109/CANDO-EPE.2018.8601139.

[8] G. C. Hillar, *Internet of Things with Python*. Packt Publishing, 2016.

[9] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Chapter 4 - Fog Computing: principles, architectures, and applications," in *Internet of Things: Principles and Paradigms*, R. Buyya and A. V. Dastjerdi, Eds. Morgan Kaufmann, 2016.

[10] J. Jermyn, R. P. Jover, I. Murynets, M. Istomin, and S. Stolfo, "Scalability of Machine to Machine systems and the Internet of Things on LTE mobile networks," in *2015 IEEE 16th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Boston, MA, USA, Jun. 2015, pp. 1–9, https://doi.org/10.1109/WoWMoM.2015.7158142.

[11] M. Gheisari, G. Wang, and S. Chen, "An Edge Computing-enhanced Internet of Things Framework for Privacy-preserving in Smart City," *Computers & Electrical Engineering*, vol. 81, Jan. 2020, Art. no. 106504, https://doi.org/10.1016/j.compeleceng.2019.106504.

[12] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Jul. 2016, https://doi.org/10.1109/JIOT.2016.2579198.

[13] D. Minoli, K. Sohraby, and B. Occhiogrosso, "IoT Considerations, Requirements, and Architectures for Smart Buildings—Energy Optimization and Next-Generation Building Management Systems," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 269–283, Oct. 2017, https://doi.org/10.1109/JIOT.2017.2647881.

[14] Z. Alwan and H. Alshaibani, "Car Accident Detection and Notification System Using Smartphone," *International Journal of Computer Science and Mobile Computing*, vol. 4, no. 4, pp. 620–635, Apr. 2015.

[15] M. H. Alkinani, A. A. Almazroi, N. Z. Jhanjhi, and N. A. Khan, "5G and IoT Based Reporting and Accident Detection (RAD) System to Deliver First Aid Box Using Unmanned Aerial Vehicle," *Sensors*, vol. 21, no. 20, Jan. 2021, Art. no. 6905, https://doi.org/10.3390/s21206905.

[16] R. A. Azdy and F. Darnis, "Use of Haversine Formula in Finding Distance Between Temporary Shelter and Waste End Processing Sites," *Journal of Physics: Conference Series*, vol. 1500, no. 1, Dec. 2020, Art. no. 012104, https://doi.org/10.1088/1742-6596/1500/1/012104.

[17] M. Basyir, M. Nasir, S. Suryati, and W. Mellyssa, "Determination of Nearest Emergency Service Office using Haversine Formula Based on Android Platform," *EMITTER International Journal of Engineering Technology*, vol. 5, no. 2, pp. 270–278, 2017, https://doi.org/10.24003/emitter.v5i2.220.

[18] "OASIS: MQTT Version 5.0," Mar. 07, 2019. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[19] F. Alorifi, S. M. A. Ghaly, M. Y. Shalaby, M. A. Ali, and M. O. Khan, "Analysis and Detection of a Target Gas System Based on TDLAS & LabVIEW," *Engineering, Technology & Applied Science Research*, vol. 9, no. 3, pp. 4196–4199, Jun. 2019, https://doi.org/10.48084/etasr.2736.

[20] "assetto-corsa-competizione," *GitHub*. https://github.com/topics/assetto-corsa-competizione.

[21] K. Alsnaie, S. M. A. Ghaly, and M. A. Ali, "Study and Design of a Multi-range Programmable Sensor for Temperature Measurement," *Engineering, Technology & Applied Science Research*, vol. 12, no. 6, pp. 9601–9606, Dec. 2022, https://doi.org/10.48084/etasr.5284.

[22] S. M. A. Ghaly, "LabVIEW Based Implementation of Resistive Temperature Detector Linearization Techniques," *Engineering, Technology & Applied Science Research*, vol. 9, no. 4, pp. 4530–4533, Aug. 2019, https://doi.org/10.48084/etasr.2894.