An IoT ecosystem for the implementation of scalable wireless home automation systems at smart city level

Conference Paper · November 2017

DOI: 10.1109/TENCON.2017.8228095

Jay Lohokare
Stony Brook University
9 PUBLICATIONS **134** CITATIONS
SEE PROFILE

Reshul Dani
University of California, San Diego
8 PUBLICATIONS **88** CITATIONS
SEE PROFILE

An IoT Ecosystem for the Implementation of Scalable Wireless Home Automation Systems at Smart City Level

Jay Lohokare, Reshul Dani, Ajit Rajurkar, Ameya Apte
Department of Computer and Information Technology
College of Engineering, Pune
lohokarejs13.comp@coep.ac.in, reshulsd13.comp@coep.ac.in, rajurkarad13.comp@coep.ac.in, apteaa15.comp@coep.ac.in,

***Abstract*— Today home automation is quite common in developed nations. Remote access of appliances and devices in a home is provided via the internet. The challenge lies in making the system smart and deployable in a scalable manner across an entire city in a developing nation. This paper proposes a framework for making home automation achievable across a smart city in a scalable and data efficient manner. We have proposed an end to end home automation suite with modular implementation using open source components. The backend implemented supports multiple IoT protocols and the system is horizontally scalable. The 'Rules engine' implemented analyzes the data real time thus triggering various actions pre-decided by users. The system involves energy monitoring for enabling judicial energy consumption in a home. Unlike smart-meters, which give data about total energy consumption per house, the proposed solution is an innovative way to get fairly accurate analysis of per device consumption. A single system can be deployed and can handle data coming from homes for an entire city. The implemented system includes web-dashboard and android application for monitoring the consumption. The scalable architecture used ensures that there will be no problem handling data traffic as number of users increase. Experiments on the prototype system showed low latency for real time analytics even with large simultaneous data streams. The unique features of this system are its modular & dynamic nature, scalability and real time analytics for quick decision making.**

Keywords*— *Internet of Things; smart home; cloud computing; home automation; IoT framework; mqtt, wireless networks; rules engine; IoT protocols; real time analytics; spark streaming; apache kafka; emqttd

I. Introduction

IoT has emerged to be a paradigm shifting technology in the past few years. IoT is expanding to many different areas ranging from smart cities, smart industries to smart healthcare and now, IoT can be utilized for improving our day to day life [1]. In order to make our lives simpler and easier, IoT can be harnessed for making our homes automated and smart [2]. Mobile devices can be used to control basic functions [3]. Users can have remote control of various appliances like lights, fans, AC etc. Thus an automation system enables control of electric appliances of various kinds. Once all these devices get connected to each other, they will enable smart processes and services that will support our basic needs and environment. Such devices connected to internet also produce a large amount of data and information. Data analysis can be done on this to find patterns and trends in the user's daily activities.

A benefit of smart home automation is the possibility of saving electric power and energy. Apart from remote control, users should be able to monitor how these devices are being used daily, what is their daily energy consumption etc. Once the users know this, they will take efforts to save energy and their own electricity bill will reduce.

There is a need to implement a complete end to end system for home automation, right from the sensors, to collection of data, transfer of data over the network, storage of this data and analysis of this data over at various stages. For each component, an analysis needs to be done as to the right technology to be chosen so as the system satisfies the needs of scalability, data efficiency and real time analysis.

The primary contribution of this paper is the innovative way to get analysis of energy consumption per device, at less cost. The proposed architecture involves scalable frameworks like eMQTT, Kafka and Spark Streaming. The implementation of this architecture involved development of scalable bridges between these frameworks.

II. Related Work

Related work mainly comprises of systems that utilize IoT for monitoring and managing home automation. In most of these systems, it is shown how data can be sent from a microcontroller to a cloud server and from the cloud to a mobile application [4], [5]. What is lacking in these systems is a proper architecture that can sustain a sharp rise in the number of users and multiple concurrent users.

Furthermore, in smart homes, security is being implemented using IoT which sends mobile alerts if sensors detect intruders [6], [7]. The advantage of our IoT framework is that it is not limited to just security. The user can set rules of various kinds and receive alerts if any of the rules get triggered in their homes. The users have freedom to define these rules. Our "rules engine" defined in the next sections is the advantage compared to this prior work. The actions to be taken after a rule is satisfied include sending an SMS or an email to the user which can easily be implemented [8].

In [9] it is shown how control of appliances can be achieved using IoT; but use of REST APIs for communication with the cloud is less efficient. In our prototype we use MQTT protocol which is data efficient and we also show how our system supports multiple IoT protocols.

In [10], a scalable architecture for home automation is proposed using Apache Storm. However, they rely on local logic for enabling scalability. In this paper, we implement scalability by choosing components on the cloud which can scale up. We show how Apache Spark Streaming handles continuous streams of data and how analysis can be done on these streams with minimal latency.

III. System Components and Requirements

The various devices and appliances in a user's home will need to have microcontroller boards which are connected to WiFi. Soon in the future, every home will be equipped with these smart devices. These devices will publish data about their current state. Example - a light bulb will publish its on/off state; a fan will publish its current speed; an air conditioning unit will publish its temperature. All this data needs to be stored in the cloud. There will be multiple devices in one home and multiple such homes. They will continuously stream data to the cloud. Therefore, the system needs to be such that it can handle the load. Furthermore, this streaming data needs to be analyzed on a near real time basis. For example, a temperature sensor will keep sending data to the cloud about the current temperature in the room. The resident can then set a rule that if the temperature goes beyond a certain point, switch on the air conditioning. There can be many such rule based actions that will need to be triggered in real time when an event happens. We have found that the following components will fulfill all these requirements that need to be met in a home automation system. They are suitable for integrating with the end-to-end automation

A. *IoT Protocol*

Traditional protocols like HTTP cannot be used in a home for sending data because of high internet consumption. There are many IoT enabled open source protocols like MQTT [11], MQTT-SN, AMQP, DDS, and XMPP, CoAP. These communication protocols are characterized by low bandwidth requirement, performance even under unreliable and unstable networks with security and encryption [12]. MQTT is a connectivity protocol for machine-to-machine communication. It has been designed in such a way that it extremely lightweight when it comes to publish and subscribe messaging transport. It has an extremely small code footprint and network bandwidth. Because of its small size and minimum data packets, it is widely used in Internet of Things [13].

Due to its light-weight nature and support to scalability, the implemented solution uses MQTT protocol. MQTT consists of three components: the subscriber, the publisher and the broker.

1) Subscriber: Those devices interested in receiving data register themselves to the broker by „subscribing" to topics of interest. This means that whenever the broker receives a message with a particular topic, it will send this message to all those who have subscribed to that topic.

2) Publisher: The publisher generates data that needs to be sent. It sends this data via a „topic" to the broker.

3) Broker: The broker is a server that contains topics. Each publisher sends information on a specific topic and the subscriber receives automatic messages each time there is a new update in the topic he has subscribed to.

B. *MQTT Broker*

It is possible to cluster MQTT brokers for achieving scalability. The clustered MQTT brokers act together as one broker. Every broker in cluster maintains the topics list. When one broker is unavailable, other broker handles the request. This way, the brokers balance within themselves. This clustering ensures that the system is highly available. Even if one server goes down, other servers will be there in its place to handle the requests and data.

C. *Apache Kafka*

The devices in home can send data from multiple protocols. There can be many protocols involved in the system in the later stages. To connect data coming from all these protocols, Apache Kafka [14] can be used. Furthermore, use of Apache Kafka enables presence of a distributed architecture. Kafka also helps to log the data thus creating a raw data file required for further data analytics. Overall, use of Kafka makes the architecture secure, highly available, scalable and distributed [15].

Implementing Kafka ensures that existing systems and architectures can integrated with the proposed system. The existing protocol brokers will act as Kafka clients and will push the data to the backend as shown in Figure 1.

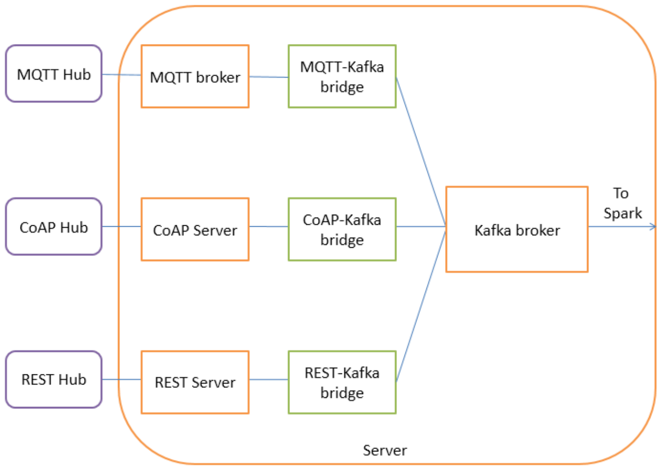

Fig. 1. Kafka as protocol aggregator

D. *Apache Spark Streaming*

A real time rules engine is a core component of home automation. Streaming data from each home needs to be analyzed as it comes and actions need to be triggered. Users also need to have flexibility as to what action they want to trigger. A list of rules needs to be maintained for each user and each rule executed on all data that is being sent from his home. For this, Apache Spark Streaming is an important component of the system because of its near real time analytics and fast large scale data processing. It is known to be 100x times faster than Hadoop [16]. Also, we can combine 'MLib' (machine learning) and streaming seamlessly to determine patterns and habits of the user.

Spark Streaming has a proven scalability and wide-spread use in industry [17], which supports our selection of Spark streaming over other existing platforms like Apache Storm, Apache Flink, Samza, Apache Apex, etc.

For these reasons, the above open source components have been chosen for the design of the home automation suite.

IV. Proposed Architecture

The components described above need to be integrated with each other for the implementation of the end-to-end system. Fig. 2 shows the entire architecture and individual components are discussed in the next section

A. IoT Modules

These are the end points of system – devices like light bulbs, fans, AC and other components in the home. There will also be various sensors like a light intensity sensor or a temperature sensor in a home. These devices will stream data continuously at the interval of two minutes. Each device will also have a state – on/off (explained in next sections). All this data needs to be captured and stored on the server. Communication between these IoT modules and the server can be enabled by implementing publish-subscribe protocols. Simplest implementation will be that of MQTT.

Every wifi-enabled controller module in the proposed system is assigned a unique ID consisting of address-key, room ID and device ID.

Address Key is unique per registered home. This can be same as the electricity meter number, to ensure uniqueness.

The room ID is assigned to every room in the home. Device ID differentiates devices in a particular room.

Publish Mechanism:

Every device in a home will publish data with a unique topic which will have the following format:

$state/address-key/room/device-id
$data/address-key/room/device-id

The reason for choosing this format is that the data coming for different users' homes needs to be differentiated. Therefore, the IoT modules will be allowed to publish with only these topics. They will publish their on/off state with the "$state" topic whenever there is a change in their state. Sensors will publish data at a defined interval with a topic "$data".

The state of a device can be changed either manually – by the user turning of the light switch or increasing the speed of the fan; or remotely through the user interface on android application that the user will have. Whenever there is a state change, the device will publish a sate update with the topic –

$state/address-key/room/device-id/update

Every module will send data as individual JSON strings. For example a light bulb will publish the following data about its state using the MQTT topic

"$state/house1/room1/bulb1/update" – {State: "on"}

The message stored in DB will contain the Timestamp (appended by Spark streaming)

```
{
      State: "on"
      Timestamp: "531423"
}
```

The implemented system uses ESP8266 based smart switches. Smart switch can be made by connecting the Arduino enabled ESP8266 to a relay. The device will make digital output HIGH when it receives State On message from servers, thereby starting the device connected to the smart switch. The State implementation can be extrapolated to include fan speed control and other similar use cases. For example, fan speed control will involve 'State' and 'Speed' parameters in the state message.

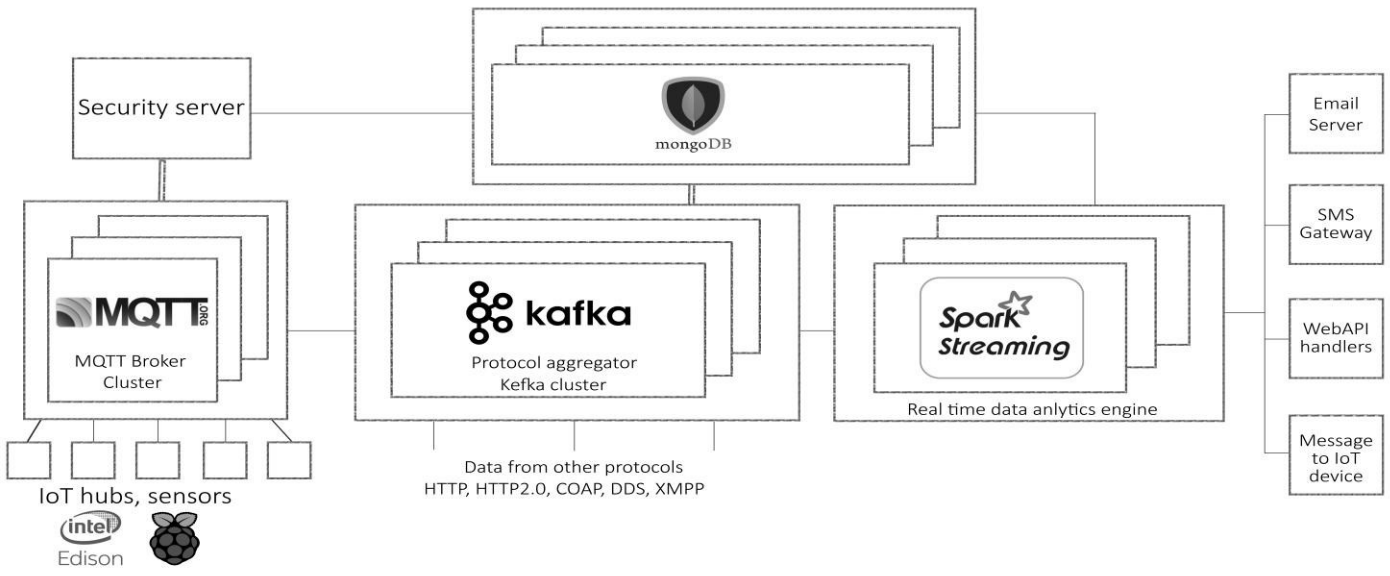

Fig. 2. Proposed end-to-end architecture

Subscribe Mechanism:

Every IoT module will also subscribe to this state topic –

$state/address-key/room/device-id/response

Thus when a state update is sent from the remote trigger (i.e. the android phone) it will be received by the intended device. The mapping between update and response is achieved by the State Database and Analytics engine which is explained in the next sections.

B. Server

The backbone of the system is a server capturing the data and controlling the data flow. The server in the prototype system is an Amazon EC2 instance [18]. Alternatively, a local on premise server can be used. EC2 offers real time scaling capabilities while benefit of on premise server is that there is no dependency on server providers and internet bandwidth utilized in connecting to remote machine is minimized. The interface between the IoT modules present in each home and the server will be an MQTT broker. The MQTT broker in the prototype is a highly scalable eMQTT server [19] (Erlang MQTT service). The eMQTT broker can be clustered for achieving high availability.

The IoT modules in a home can communicate with the server via multiple protocols like MQTT, CoAP etc. so multiple brokers for these protocols can be added. All of these brokers will stream data to Apache Kafka. The reasons for using Apache Kafka have already been highlighted in the previous section. Thus, the broker will have a Kafka producer that will publish every message that it gets to the Kafka cluster. The MQTT-Kafka Bridge as shown in Figure 1 takes any message received by the broker and publishes it to Kafka. It appends the original MQTT topic with the original MQTT message and produces this new message with topic "mqtt".

Spark Streaming will get this message, and analyze the content to perform actions based on the topic. Similarly there will multiple bridges like these for CoAP and other protocols.

Consider message: {State:"on"} received by Kafka. Kafka appends MQTT topic and sends it to Spark Streaming as

```
{"topic":"$data/house1/bedroom/tempsensor",
    "message": [{State: "on"}]}
```

C. Analytics Engine

The data collected from Apache Kafka needs to be analyzed real time. This data will be collected by the rules engine based on Apache Spark Streaming. The Streaming application will have a Kafka DStream [20] which will consume all messages with topic "kafka". Actions will be triggered depending on data collected. The proposed solution has a rules engine where user can set notifications when certain reading surpasses a threshold. For example, if the temperature exceeds 40 degree, the user can set to automatically switch on the Air Conditioner. Or if the energy consumption of TV exceeds 100 units for the month, the user can set that an email/SMS be sent to him. These rules will be user specific and set through a web dashboard. The details about rules per user will be stored in MongoDB collections. These rules will be transformed into SQL queries by the spark streaming application and executed on each RDD (Resilient Distributed Datasets - spark data abstraction).

For example, consider the following JSON string is received by Spark Streaming application:

```
{
    "topic":"$data/house1/bedroom/tempsensor",
    "message": [{"temperature": 50}]
}
```

The spark streaming application will first retrieve the rules corresponding to the topic $data/house1/bedroom/tempsensor and the corresponding action (Fetched through the database).

D. State Database

Whenever the analytics engine receives a message which has a state update topic, it writes the state in the MongoDB [21] present on the Server. We will call this database "State DB". It will have one collection per house in a city and each collection will have the documents for devices present in the home. The write from Spark to MongoDB is achieved via the MongoDB Spark Connector [22]. To decide which collection to write to, the topic string is parsed. There is a collection corresponding to every Address Key.

E. Flow of data in backend

When a new device is introduced to a system, it first gets a new ID of format "address-key/room/device-id". The user then sets username and password for that device, which will act as authenticator with the backend. The device ID along with username and password ensures that a device doesn't publish or receive data related to some other device.

Users can add Rules to the backend through the web-dashboard. Every rule corresponds to a MQTT topic. Rules are actions to be performed when data coming from the MQTT topic matches a particular condition (Like temperature>20). For example –

1. If user wants to get email when temperature in room1>50, user will select the appropriate MQTT topic to be analyzed, select action as 'Email' and add necessary extra parameters through the dashboard.

Topic: $data/12345/room1/tempsensor /
Condition: "select * from message where temperature >50"
Action: Email

    To: xyx@abc.com
    Message: message + "Is the data received"
    From: noreply@xyz.com
    Subject: "Room temperature exceeded"

2. If user wants to start AC when temperature > 50, the parameters saved into MongoDB will be following:

Topic: $data/12345/room1/tempsensor /
Condition: "select * from message where temperature >50"
Action: republish

    Topic: $state/12345/room1/ac/update
    Message: {State:"on"}

The implemented system supports messages of 2 types:

1. State – Data related to state of the device
2. Data – All other data messages

The device can update its state by publishing to topic:

$state/address-key/room/device-id/update

When the device publishes data to MQTT, the MQTT-broker checks the ID-username-password mapping. This ensures data security and privacy. The MQTT broker then publishes the data via an asynchronous Kafka client, to ensure non-blocking functioning. The MQTT broker appends the message MQTT topic in the message, and publishes it to Kafka broker with topic 'kafka'. Apache Spark Streaming gets this data in form of RDDs.

Spark Streaming analyzes these RDDs and forms maps of messages to their topics. It then fetches the rules from MongoDB, and performs the actions accordingly.

State of a particular device can be requested by publishing a MQTT message of following format:
If state needed is for device ID 'address-key/room/device-id', an empty MQTT message with topic:

$state/address-key/room/device-id/request

Should be published. Spark Streaming will detect this request, fetch the state from state DB and publish it with topic:

$state/address-key/room/device-id/response

F. Energy Audit

The proposed system uses a simple but intuitive method for monitoring the energy consumption of the appliances at home. The backend of the system captures the timestamps corresponding to the State "on" and "off" messages received. Using these timestamps, the backend is able to find the total amount of time the devices were switched on for. The users can input the average energy ratings of these devices, which will enable the system to calculate the approximate energy consumption of every device. The method though not extremely accurate, will help the users to get estimates of energy consumption of devices, there by knowing which devices lead to maximum energy consumption.

The conversion of on/off time-stamps to total energy consumption is achieved by using Apache Spark jobs. Per device, the Spark job gets timestamps, converts them to total time the device was on for, and then multiplies it with average energy rating to get total energy consumed.

V. Qualitative Analysis

We implemented the entire architecture proposed in this paper and deployed our system on Amazon EC2 instance cloud. This includes the clustered MQTT brokers, Apache Kafka, rules engine and databases. As IoT modules, we used Raspberry Pi and Wemos D1 Mini which were Wi-Fi enabled. These modules will be present in all homes and will publish with certain topic and subscribe to specific topics as defined in the section above. The use of the proposed architecture leads to the following unique features of the home automation suite:

Single, scalable and Generic System. All homes in a city can exploit the benefits of automation by connecting to the same backend. The system will thus be a reliable one stop platform that will enable the automation of all homes in a city. Adding new homes and devices to the system is extremely easy and can be done via simple interfaces. As the backend reads actions to perform from a database, adding new devices or actions is just a task of adding corresponding entry to the database. As shown in the quantitative analysis, the architecture proposed has a potential to handle huge data generating from millions of homes and devices in a city.

Remote Control. The control of home appliances via the internet is one of the basis features of any home automation system. The user should first of all be able to control the state of each and every device in his house. For example, he should be able to turn the lights on and off, reduce the intensity etc. He should be able to turn the AC on before coming home from his smartphone. This is easily achievable through the use of our system and we demonstrated it by the creation of an android application. The application used Eclipse Paho MQTT client. It subscribed to $state/home1/+/+/response. That is, it subscribed to all state update that will be sent from all devices in a home ("+" in MQTT means all). Whenever a device's state is changed manually, it will publish with the update topic. The android phone will receive this message and update its user interface to show the current state of this device. When the user changes a device state from the android phone (eg: changes the speed of a fan from the user interface), the phone will publish this change with the MQTT topic - $state/home1/room1/fan1/update. Since the fan1 will have subscribed to the response topic, it will receive this message and change its speed. Thus remote control of all devices was achieved in this manner.

Fail Proof Pub-Sub Mechanism. The state of a device is always synced with servers and controlling applications. Suppose a user starts a fan from web-dashboard, this change of state is immediately reflected to all other users too. If a user changes state of a device while that device is not connected to internet, the database will record this change; and push it to the device when it comes online.

Real Time rules engine. The use of Spark Streaming enables near real-time processing of data that is being received from all homes. The distributed nature of data processing in Apache Spark Streaming will ensure that the data analysis is scalable. For every real time data batch received, the data will be divided between the Spark Streaming slaves (Cluster worker nodes) by the Spark Streaming master node. Thus, every spark cluster worker will receive certain part of the real time data received. The spark node will then analyze the data that it gets to pass on the results to the master. The node will first find the home (MQTT topic) corresponding to each message received. It will then find rules corresponding to that particular topic and execute the Spark SQL queries on the input JSONs. If any JSONs satisfying the query are found, the action corresponding to the rule will be triggered.

Energy audit. A web dashboard was implemented where the user will be able to see all details about every connected device. The user can see his monthly consumption of energy, which device was running the longest, which device is consuming the most etc. This can be shown to him through the medium of monthly and yearly graphs. The rules and action corresponding to each rule for the rules engine were also taken from the user via a web dashboard and stored in MongoDB. With the presence of this dashboard, the end-to-end home automation suite is complete.

VI. Quantitative Analysis

Data efficiency. We use a light weight IoT protocol MQTT for enabling data efficiency. In order to avoid flooding the network with sensor data, the system sends only changes in the state of a device. The proposed system required total 60KB data per 5 hours, with data update frequency of 5 minutes; where HTPP based systems take 320KB. The proposed system thus reduces data consumption by ~80%

Scalability. Use of scalable components and asynchronous request calls makes the proposed system highly scalable. In an experimental setup, the system was deployed on 2 Amazon web services (AWS) Elastic cloud compute (EC2) instances with 1GB RAM each. Apache J-Meter was used to load test the servers.

In the experimental setup, every AWS node had a Spark slave and MongoDB slave. Half the nodes had MQTT and the other half had Kafka nodes. The spark streaming window used was of 2 seconds. Every AWS spark instance constitutes 1 spark executor. Table I shows the performance of our system.

TABLE I. PERFORMANCE OF THE CLUSTER SYSTEM

Messages	1 node	2 nodes	3 nodes	5 nodes
1,000	6ms	4ms	3ms	2ms
5,000	36ms	23ms	15ms	9ms
10,000	79ms	49ms	34ms	21ms
1,00,000	850ms	530ms	390ms	220ms
5,00,000	4300ms	2500ms	1900ms	1300ms

VII. Conclusion

In this paper we identified and studied the problem of deploying a home automation system at smart city. There was a need of selecting the right components which enabled the implementation of the above features keeping in mind the need for scalability. We proposed a scalable architecture for achieving home automation with modular and open source implementation. Apart from remote control which existing solutions provide, this system provides actions based on rules and per device energy audit. Quantitative and qualitative analysis showed the benefits of the architecture proposed by us. The system is dynamic and future ready in the sense that it can easily scale up with increasing number of connected users. Thus, an integrated open-sourced IoT ecosystem was developed for Home Automation. It handled the entire data lifecycle and made real time analytics possible. The system provides a linear scalability which has been proven by the experimental setup. The system implemented allows tracking energy usage per device, giving micro-level analysis in energy audits.

References

[1] H. N. Saha, A. Mandal and A. Sinha, "Recent trends in the Internet of Things," *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, 2017, pp. 1-4.
doi: 10.1109/CCWC.2017.7868439

[2] G. Kesavan, P. Sanjeevi and P. Viswanathan, "A 24 hour IoT framework for monitoring and managing home automation," *2016 International Conference on Inventive Computation Technologies (ICICT)*, Coimbatore, 2016, pp. 1-5.

[3] K. Mandula, R. Parupalli, C. A. S. Murty, E. Magesh and R. Lunagariya, "Mobile based home automation using Internet of Things(IoT)," *2015 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, Kumaracoil, 2015, pp. 340-343.

[4] M. Thiyagarajan and C. Raveendra, "Integration in the physical world in IoT using android mobile phones," *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, Noida, 2015, pp. 820-826.

[5] D. Pavithra and R. Balakrishnan, "IoT based monitoring and control system for home automation," *2015 Global Conference on Communication Technologies (GCCT)*, Thuckalay, 2015, pp. 169-173.

[6] V. Patchava, H. B. Kandala and P. R. Babu, "A Smart Home Automation technique with Raspberry Pi using IoT," *2015 International Conference on Smart Sensors and Systems (IC-SSS)*, Bangalore, India, 2015, pp. 1-4.

[7] R. K. Kodali, V. Jain, S. Bose and L. Boppana, "IoT based smart security and home automation system," *2016 International Conference on Computing, Communication and Automation (ICCCA)*, Noida, 2016, pp. 1286-1289.

[8] Jain Sarthak, Anant Vaibhav, Lovely Goyal, "Raspberry Pi based interactive home automation system through E-mail", *Optimization Reliabilty and Information Technology (ICROIT) 2014 International Conference on. IEEE*, 2014.

[9] R. K. Kodali, S. Soratkal and L. Boppana, "IOT based control of appliances," *2016 International Conference on Computing, Communication and Automation (ICCCA)*, Noida, 2016, pp. 1293-1297.

[10] M. Frincu and R. Draghici, "Towards a scalable cloud enabled smart home automation architecture for demand response," *2016 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, Ljubljana, 2016, pp. 1-6.

[11] MQTT , [online] Available: http://mqtt.org/documentation

[12] Y. Chen and T. Kunz, "Performance evaluation of IoT protocols under a constrained wireless access network," *2016 International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT)*, Cairo, 2016, pp. 1-7

[13] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight Internet protocols for web enablement of sensors using constrained gateway devices," *2013 International Conference on Computing, Networking and Communications (ICNC)*, San Diego, CA, 2013, pp. 334-340.

[14] Apache Kafka, [online] Available: https://kafka.apache.org/

[15] Z. Wang *et al*., "Kafka and Its Using in High-throughput and Reliable Message Distribution," *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, Tianjin, 2015, pp. 117-120.

[16] Apache Spark Streaming, [online] Available: http://spark.apache.org/streaming/

[17] S. Chintapalli *et al*., "Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming," *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Chicago, IL, 2016, pp. 1789-1792.

[18] Amazon EC2, [online] Available: https://aws.amazon.com/documentation/ec2/

[19] EMQTT Scalable MQTT Broker, [online] Available: http://www.emqtt.io

[20] Spark Streaming + Kafka Integration, [online] Available: https://spark.apache.org/docs/latest/streaming-kafka-integration.html

[21] MongoDB, [online] Available: https://docs.mongodb.com/

[22] MongoDB-Spark connector, [online] Available: http://www.mongodb.com/products/spark-connector