TELKOMNIKA, Vol.17, No.3, June 2019, pp.1385~1392
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v17i3.11793

Lightweight IoT middleware for rapid application development

A. Karim Mohamed Ibrahim[1], Rozeha A. Rashid*[2], A. Hadi Fikri A. Hamid[3], M. Adib Sarijari[4], Muhammad Ariff Baharudin[5]
Advanced Telecommunication and Technology Research Group (ATT),
Department of Communication Engineering, Faculty of Electrical Engineering,
Universiti Teknologi Malaysia
*Corresponding author, e-mail: kemosom@gmail.com[1], rozeha@utm.my[2], hadifikri@gmail.com[3], madib@utm.my[4], mariff@utm.my[5]

Abstract

Sensors connected to the cloud services equipped with data analytics has created a plethora of new type of applications ranging from personal to an industrial level forming to what is known today as Internet of Things (IoT). IoT-based system follows a pattern of data collection, data analytics, automation, and system improvement recommendations. However, most applications would have its own unique requirements in terms of the type of the smart devices, communication technologies as well as its application provisioning service. In order to enable an IoT-based system, various services are commercially available that provide services such as backend-as-a-service (BaaS) and software-as-a-service (SaaS) hosted in the cloud. This, in turn, raises the issues of security and privacy. However there is no plug-and-play IoT middleware framework that could be deployed out of the box for on-premise server. This paper aims at providing a lightweight IoT middleware that can be used to enable IoT applications owned by the individuals or organizations that effectively securing the data on-premise or in remote server. Specifically, the middleware with a standardized application programming interface (API) that could adapt to the application requirements through high level abstraction and interacts with the application service provider is proposed. Each API endpoint would be secured using Access Control List (ACL) and easily integratable with any other modules to ensure the scalability of the system as well as easing system deployment. In addition, this middleware could be deployed in a distributed manner and coordinate among themselves to fulfil the application requirements. A middleware is presented in this paper with GET and POST requests that are lightweight in size with a footprint of less than 1 KB and a round trip time of less than 1 second to facilitate rapid application development by individuals or organizations for securing IoT resources.

Keywords*: application programming protocol (API), Internet of Things (IoT), middlewar*

1. Introduction

The internet paradigm has evolved from digital connection connecting computers to connecting people, thus creating social interactions and bringing humans closer to each other. Presently, the internet and the things such as, sensors, actuators, and smart devices have created an ecosystem called The Internet of Things (IoT), which is the next technological revolution since the internet [1-4]. The term IoT is mainly used to refer, the network of smart things connected by internet, the enabling technologies, such as machine-to-machine communication, RFID, Wireless sensor Network (WSN), sensors/actuators, and the set of applications that embraces the vision of this technology to open up a new business frontier [4, 5].

IoT has a layered architecture to meet the requirements and demands of the industries and society. Although there is no consensus on the layered architecture, Figure 1 show the conventional generally accepted layered model for IoT to encapsulate the idea in a structured manner. It consists of the followings: the edge layer which contains the physical abstraction layer that represents objects such as smartphones, sensors and actuators responsible for edge functionalities such as collecting information and providing point to point communication support. On top of the edge layer is the gateway layer. This layer focuses on exposing the edge devices to a larger network while providing the connected edge devices an end to end connectivity. On top of it is the middleware layer which is responsible for bridging the devices and becomes the gateway to the application layer and to ensure a seamless operation between the application layer and the edge layer. Finally, the application layer is the interface to a service

Received November 10, 2018; Revised January 30, 2019; Accepted February 28, 2019

provider or the IoT user, and it is responsible for providing application services based on the user or service provider requirements [6].

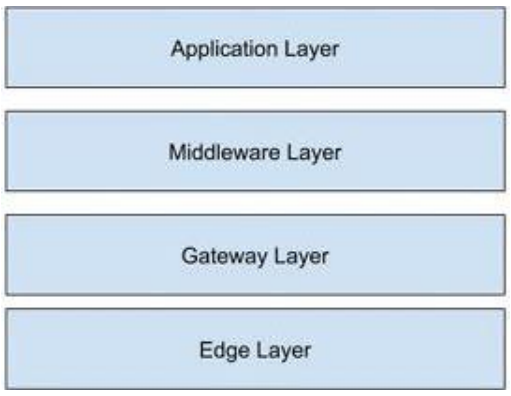


Figure 1. Basic layered model for IoT

IoT has helped in fostering the development in various industries, such as healthcare [7], industrial applications [8], agriculture [9], smart cities [10], and home appliances. In these landscapes, IoT devices provide crucial data to the respective industry players by sensing and communicating the data to the business entities for better understanding of the nature of their businesses as well as for providing services to the public and companies [11-13]. Network providers, internet engineers, and data analysts agreed that the number of connected devices will increase rapidly in the coming years [14]. According to Cisco IBSG, IoT will have more than 40 billion connected objects in 2020. With the plethora of connected devices, IoT platforms have been introduced, such as Google Cloud Platform [15], IBM BlueMix, and ThingWorx which help to create and deploy IoT applications [16]. Normally, a platform comes in a complete package which consists of hardware and software. IoT middleware, on the other hand, is the glue between devices and applications, a type of distributed system that provides services to devices. According to [17], IoT middleware has several merits in IoT world. These include providing a chain combining effect for IoT heterogeneous components, management of data as well as enabling an abstraction of physical layer while hiding all the details and complexity stemmed from the disparate technologies that make up the IoT ecosystem.

To enable IoT application, one of these services must be available such as Platform as a Service (PaaS,) Software as a Service (SaaS) and Backend as Service (BaaS). For business entities and consumers alike, this could potentially result in a huge security problems [18, 19]. Sharing resources is a sensitive matter that should be handled carefully. Furthermore, owing to the fact that every IoT application has its own requirement, it is difficult for some companies or individuals to use the generic services provided by the available platforms due to the general workflow provided by most platforms do not conform to their requirements. For example, some platforms are specific to one domain such as healthcare or agriculture which has different requirements such as data latency and privacy as well as approach. To overcome these issues, a lightweight IoT backend that makes an IoT application feasible and programmable by individuals who are not necessarily experts in pervasive computing is proposed. Almost everything on the internet requires a lightweight IoT due to the fact that most devices are resource constrained [20, 21]. According to [17], several IoT middlewares have been developed but a lightweight plug-and-play IoT backend is still lacking. In this paper, we propose a future-proof lightweight IoT backend that can be hosted in any on-premise data center for the purposes of securing the resources, avoiding the pitfall of cloud services and facilitates a mechanism for resource sharing in an efficient manner.

The remainder of this paper is organized as follows. First, the motivation for the implementation of lightweight IoT middleware and related works are presented in Section II. Then, Section III introduces the representational state application programming interface (API) and its role in IoT middleware. The methodology is discussed in Section IV, followed by the preliminary results in Section IV. Lastly, the conclusion is drawn in Section V.

2. Motivation

By 2020, IoT world is expected to have millions of connected things; the rapid expansion of IoT market share creates business opportunities for industries [22]. This expansion requires a precise and secure lightweight middleware that provides services for the devices and service provid [23, 24]. Middleware is the crucial component of IoT and it lies between a bank of devices and a cloud service [25]. In IoT, the system should run on edge devices and not in the cloud. Therefore, at the edge or the border, an integration between connected devices and the cloud service is necessary to facilitate the IoT ecosystem [26, 27]. Most companies use default hybrid integration such as PaaS and SaaS. However, nowadays, the integration does not only depend on the technology but different locations as well. Furthermore, the variety of communication protocols and the amount of data that goes through the middleware requires a secure framework that ensures smooth integration in IoT system. Currently, many IoT middlewares are widely deployed in an environment that is subject to rules provided by the service providers with resource sharing method known to others, which gives rise to security issue. Our scope of the research work is to provide the developers with the necessary tools to access the web with respect to HTTP methods using RESTful APIs with lightweightness in mind.

3. Related works

Many IoT middlewares are being deployed in tandem with the growth of IoT. Calvin [28], an open source IoT middleware from Ericsson, is one of the few lightweight IoT middlewares. Using proprietary programming model, Calvin provides minimum latency, and high CPU utilization [16]. The limitation in Calvin is that it does not offer Graphical User Interface (GUI). The security and device discovery also are not addressed. Ptolemy Accessor Host, is an actor-oriented framework for modeling, simulating, and designing of concurrent, real-time, embedded devices [29]. Like most of the middlewares, they follow the basic model of computation which is data/event flow. In Ptolemy, guaranteed execution of real-time distributed IoT application is provided. However, the limitation in Ptolemy is until now, it does not support device discovery. Project Flogo [30] on the other hand is a ultra-lightweight middleware that focuses on edge integration from device to gateway and from gateway to cloud. It is an open source middleware that can run on devices and requires less than 5 MB memory space [30]. Project Flogo is written in Golang programming language which is not as popular as Node.js. The community in Node.js provides supportive libraries and codes. The related works are summarized in Table 1. In a nutshell, the developers can make their own choices when developing the middleware.

Table 1. Summary of Related Works

No	Title	Author/year	Description	Limitation
1	Calvin—Merging cloud and IoT [5].	P. Persson and O. Angelsmark/ 2015.	Aims to provide a unified programming model which is light-weight and portable.	No GUI. Security and device discovery are not addressed.
2	A development platform for integrating wireless devices and sensors into ambient intelligence systems [10].	M. Eisenhauer, P. Rosengren, and P. Antolin/ 2009	It provides support for using devices as services by embedding services in devices.	Need for programmer. It does not help the user to quickly, create, search and deploy data collection and application analysis.
3	Edge Machine Learning with Flogo [7].	Gaurav Subhedar/2018.	An open source IoT middleware, consumes 3MB memory space. Uses Golang programming language. Main focus of this project was edge integrations.	The programming language is not popular among developers.
4	A visual tool for writing the internet of things [11]	Node,RED 2015	An open-source IoT middleware platform from IBM. It is based on node.js It provides a visual tool that simplifies the job of composing IoT devices	Limited security Limited libraries and modules No device discovery

4. Application Programming Interface (API)

According to [31], API is an interface to a software component that can be invoked at a distance over a communications network using standards-based technologies. Applications use APIs like we use web pages as an interface. In IoT, many sensors are connected and share information with other entities in a relatively complex ecosystem. In such scenario, API provides the mean of communicating between different entities while maintaining data integrity and consistency. API is actually an old technology that came up first in 1970 because of distributed systems, and then grew in popularity. In 2000, Roy Thomas introduced in his Ph.D dissertation representational state transfer API (RESTful API) [31]. API provides great services to IoT world. Designing an API is not a difficult matter, but designing an efficient one that meets business requirements is not a trivial task. In an API, the customer is the developer, not the end user. Figure 2 shows the importance of the API due to its sensitive position in developing process. From Figure 2, business assets of a company can consist of products information, services, and the developers who develop IoT applications. Therefore, the customer of an API is not the end user but the developer. So designing an API that meets the developer's requirement is the key to effective API. By adding some sort of storage and a user interface like a web page, a lightweight IoT middleware can be realized.

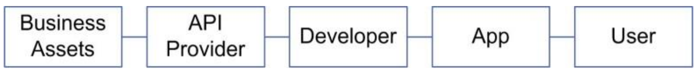


Figure 2. API lies in between two main attributes in IoT environment

5. Middleware Architecture

The proposed middleware could be deployed in a distributed manner and coordinate between themselves to fulfill the application requirements. Application abstraction level in the architecture of IoT middleware is usually used for providing an interface to applications, where context analysis level and the remote database makes it necessary to have distributable IoT middleware. The proposed middleware adopts the actor-based architecture described in [32]. The actor-based architecture shown in Figure 3 is based on lightweight middleware which provides services as actors. The actor (functionality/services) can be deployed in all layers such as sensors, mobile and cloud layer. For example, if a device in the sensor layer has a actor-based middleware that lacks storage services, another actor-based middleware that has storage service can be downloaded from the cloud [32].

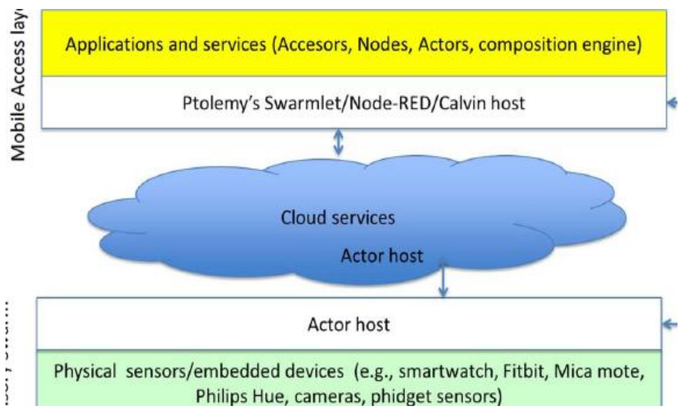

Mobile Access lay
Applications and services (Accesors, Nodes, Actors, composition engine)
Ptolemy's Swarmlet/Node-RED/Calvin host
Cloud services
Actor host
Actor host
Physical sensors/embedded devices (e.g., smartwatch, Fitbit, Mica mote, Philips Hue, cameras, phidget sensors)

Figure 3. Actor-based architecture

6. Research Methodology

In this section, the flow of the project will be explained. For this middleware, Node.js will be used for simplicity as well as due to its un-opinionated nature. As a proof of-concept, one endpoint and a hard-coded database are included. In addition, it can be connected to a local or cloud-based database as well. Figure 4 shows a client which can be a web application and the

RESTful server. The communication between the client and the server is sent via HTTP using JavaScript Object Notation (JSON) format. Users can send requests using HTTP methods available such as GET, POST, DELETE and PUT. There are also PATCH, HEAD, and CONNECT requests. This paper will illustrate the use of GET, POST, PUT, and DELETE only. Figure 5 shows the flow chart of the middleware, which is a server running on port 3000. Once a request is made, the middleware will first handle any errors that are present. If there is none, then the middleware chooses which route to take and processes it, then responds back to the user.

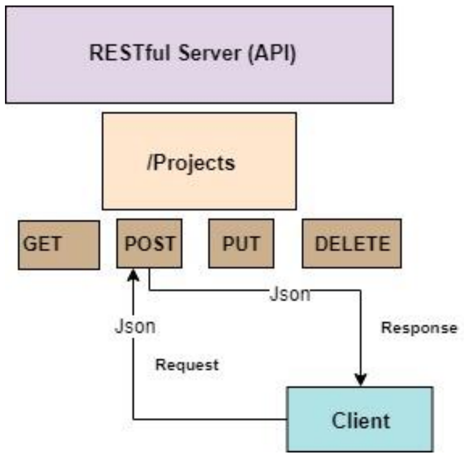


Figure 4. General view of the middleware

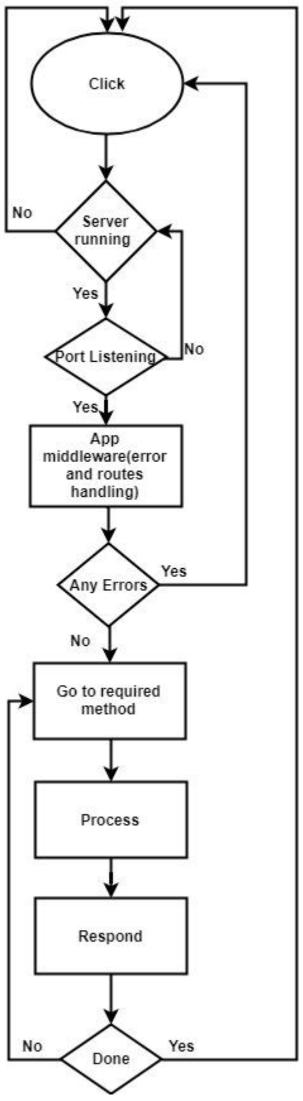


Figure 5. Middleware flow chart

7. Results and Analysis

The middleware is structured as follows. First, the middleware was developed using Node.js with JavaScript libraries and Node Package Manager (NPM) which is one of the largest ecosystems of open source libraries and they reduce the complexity of creating a web applications. Second, we show how GET, POST, PUT, and DELETE requests allow the user to interact with the database resources. Then, we show the results of GET and POST requests sent by the client to the database through the backend. The backend exposes its functionality as an API. These sets of APIs are parsed with the data and they are addressable to the clients/devices. The clients/devices are accessible through REST APIs provided by the IoT backend. Figure 6 shows a server set to listen on port 3000, and an environment variable port, which is necessary for applications working in other environments. Figure 7 shows GET request. It provides the functionality of retrieving the informations from a specified resource. GET request is the most common HTTP request and it is usually used for retrieving non-sensitive datadata. In this middleware, for proof of concept, it is allowed to retrieve all the resources as shown in Figure 7.

```
Const http = require ('http');
Const app = require (./app');
Const port = process.env.PORT || 3000;
Const server = http.createServer(app);
Server.listen(port);
```

Figure 6. Crating a server

```
router.get('/', (req,res, next) => {
     res.send(projectsdb);
     res.status(200).json({ message:
Projects were fetched'});
}
```

Figure 7. Get request

POST request is a sensitive request. It is used for adding content to the database. As can be seen in Figure 8, if a post request is made, first the middleware will validate the request. After validation, if there is no error, the middleware increases the database by index 1 and inserts the new post. PUT request is used for updating the content of the database as shown in Figure 9. When a request is made for a specific project, the middleware looks for the requested project in the database by using a respective identification (id). If there is no match, the response of (404) which is an HTTP status code is returned with notation of Not Found. Else, an update and a return acknowledgement of a new project is given. DELETE request follows the same method. The requested project is searched in the database. If found, a return acknowledgement on the deleted project is given. Figure 10 shows DELETE request for deleting a specified resource.

```
router.post('/:id' ,function(req, res){
    const { error } =
validProject(req.body);
    if ( error){
res.status(400).send(error.details[0].mes
sage);
    return;
}
const project = {
     id: projectsdb.length + 1,
     name: req.body.name
};
projectsdb.push(project);
     res.send(projectsdb);
});
```

Figure 8. Posting to database

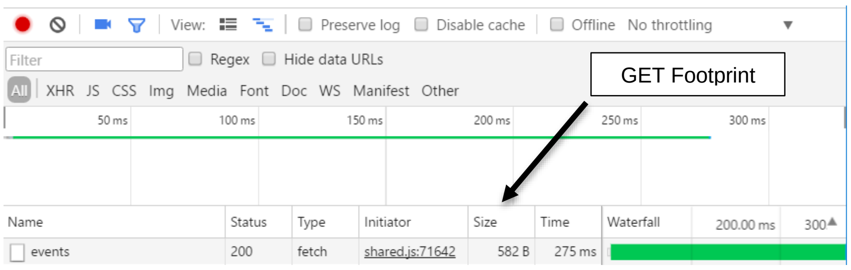
```
router.put('/:id' ,function(req, res){
  const project = projectsdb.find (p =>
p.id === parseInt(req.parms.id))
  if (!project) res.status(404).send('
there is no such project with this id')
  const {error} =
validateProject(req.body);
  if {error } {

res.status(400).send(error.details[0].mes
sage);
    return;
  }
  project.name = req.body.name;
  res.send(project);
):
```

Figure 9. Updating specific project

```
router.delete('/:id', function(req,res){
const project = projectsdb.find (p => p.id ===
parseInt(req.parms.id))
if (!project) res.status(404).send('there is no such project with
this id')
  const index = projectsdb.indecOf(projectsdb);
  projectsdb.splice(index, 1);
  res.send(project);
});
```

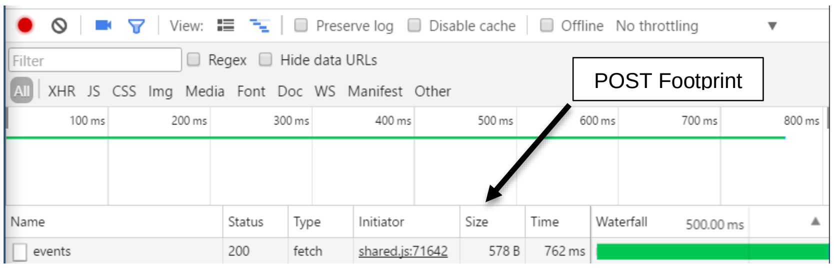
Figure 10. Deleting project

In order to carry out the test and analyze the performance of the middleware, we use POSTMAN software which is installed in HP laptop with 8GB RAM and Windows 10 OS. Also, the network used is an Ethernet connection with 17.3 MB download and 95.01 MB upload. POSTMAN is used to test the endpoints and sends the requests and responses. The objective was to justify that the middleware is lightweight by assessing the required time and the size of the response of GET and POST requests to the database resources. Figures 11 and 12 show the computational footprint of GET and POST requests, which are summarized in Table 2. GET request uses 582 B and takes a time of 275 ms. POST uses 578 B and takes 762 ms. It can be noted in this respect that the proposed middleware promotes a lightweight approach for rapid application development and ease of deployment. The proposed lightweight IoT middleware has a great advantage in terms of security, compared to heavyweight IoT middleware which usually uses markup languages such as XML, which it difficult to be deployed on resource-constrained devices. Most of heavyweight IoT middlewares are based on Service-oriented architecture (SOA), that are deployed on multiple nodes. A middleware written in Java occupies 180 MB Memory footprint compared to lightweight IoT middleware written in Node.js which occupies 74 MB.



Name	Status	Type	Initiator	Size	Time	Waterfall 200.00 ms 300
events	200	fetch	shared.js:71642	582 B	275 ms	

Figure 11. GET footprint



Name	Status	Type	Initiator	Size	Time	Waterfall 500.00 ms
events	200	fetch	shared.js:71642	578 B	762 ms	

Figure 12. Post footprint

Table 2. GET/POST Footprints

Request	Size in Bytes	Time in ms
GET	582	257
POST	587	762

8. Conclusion

IoT middleware is the core of IoT, yet deploying the middleware in constrained environment comes with associated challenges such as security issue. In this paper, we have proposed a lightweight IoT middleware for rapid development and deployment of IoT applications to provide a more secure way to protect resources for organizations and individuals by using the concept of REST API interface. The middleware uses RESTful services and enables the developers to create and modify their resources according to application needs. The REST API and its importance in the middleware is also discussed. Finally, a few simple codes for deploying such lightweight IoT middleware platform are demonstrated. Future works will involve adding a database to store data, and to develop a friendly graphical user interface (GUI) front end. The database will be based on document model such as MongoDB. Error handling mechanism will be included, especially cross-origin (CORS) errors. Google developer tools will be used to analyze the platform. Well organized understandable documentation also will be provided for the developer.

Acknowledgement

The authors wish to express their gratitude to Ministry of Higher Education (MOHE), Malaysia and Research Management Center (RMC), Universiti Teknologi Malaysia for the financial support of this project under GUP research grant no: Q.J130000.2523.19H25.

References

[1] Raggett D. The web of things: Challenges and opportunities. *Computer*. 2015; 48(5): 26-32.
[2] Want R, Schilit BN, Jenson S. Enabling the internet of things. Computer. 2015; 48(1): 28-35.
[3] Baresi L, Mottola L, Dustdar S. Building software for the internet of things. *IEEE Internet Computing*. 2015; 1(2): 6-8.
[4] Atzori L, Iera A, Morabito G. The internet of things: A survey. *Computer networks*. 2010; 54(15): 2787-805.
[5] Liu T, Lu D. *The application and development of IoT*. Information Technology in Medicine and Education (ITME), 2012 International Symposium. 2012; 2: 991-994.
[6] Evans D. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*. 2011; 1: 1-1.
[7] Islam SR, Kwak D, Kabir MH, Hossain M, Kwak KS. The internet of things for health care: a comprehensive survey. *IEEE Access*. 2015; 3: 678-708.
[8] Miller D. Blockchain and the Internet of Things in the Industrial Sector. *IT Professional*. 2018; 20(3): 15-18.
[9] Shifeng Y, Chungui F, Yuanyuan H, Shiping Z. Application of IOT in agriculture. *Journal of Agricultural Mechanization Research*. 2011; 7: 190-193.
[10] Zanella A, Bui N, Castellani A, Vangelista L, Zorzi M. Internet of things for smart cities. *IEEE Internet of Things journal*. 2014; 1(1): 22-32.
[11] Paridel K, Bainomugisha E, Vanrompay Y, Berbers Y, De Meuter W. Middleware for the internet of things, design goals and challenges. *Electronic Communications of the EASST*. 2010; 28.
[12] Gubbi J, Buyya R, Marusic S, Palaniswami M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*. 2013; 29(7): 1645-1660.
[13] Dohr A, Modre-Opsrian R, Drobics M, Hayn D, Schreier G. *The internet of things for ambient assisted living*. Information technology: new generations (ITNG), 7th international conference. Las Vegas. 2010: 804-809.
[14] Ofcom. Communications market report. UK, London, 2017. [Online] Available: https: //www.ofcom.org.uk/__data/assets/pdf_file/0017/105074/cmr-2017-uk.pdf
[15] Google Cloud. (2018). Google Cloud IoT–Fully managed IoT services from Google. [online] Available at: https: //cloud.google.com/solutions/iot/ [Accessed 26 Jun. 2018].
[16] Nakhuva B, Champaneria T. Study of Various Internet of Things Platforms. *International Journal of Computer Science & Engineering Survey (IJCSES)*. 2015; 6(6): 61-74.
[17] Tiburski RT, Amaral LA, de Matos E, de Azevedo DFG, Hessel F. The Role of Lightweight Approaches Towards the Standardization of a Security Architecture for IoT Middleware Systems. *IEEE Communications Magazine*. 2016; 54(12): 56-62.
[18] Barona R, Anita EAM. *A survey on data breach challenges in cloud computing security: Issues and threats.* 2017 International Conference on Circuit, Power and Computing Technologies (ICCPCT). Kollam. 2017; 1-8.
[19] Rahulamathavan Y, Rajarajan M, Rana OF, Awan MS, Burnap P, Das SK. *Assessing Data Breach Risk in Cloud Systems*. 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). Vancouver. 2015; 363-370.
[20] Zhou J, Cao Z, Dong X, Vasilakos AV. Security and privacy for cloud-based IoT: challenges. *IEEE Communications Magazine*. 2017; 55(1): 26-33.
[21] Wang K, Wang Y, Sun Y, Guo S, Wu J. Green industrial internet of things architecture: An energy-efficient perspective. *IEEE Communications Magazine*. 2016; 54(12): 48-54.
[22] Singh S, Singh N. *Internet of Things (IoT): Security challenges, business opportunities & reference architecture for E-commerce*. Green Computing and Internet of Things (ICGCIoT), 2015 International Conference. Noida. 2015; 1577-1581.
[23] Wang K, Wang Y, Sun Y, Guo S, Wu J. Green industrial internet of things architecture: An energy-efficient perspective. IEEE Communications Magazine. 2016; 54(12): 48-54.
[24] da Cruz MA, Rodrigues JJ, Al-Muhtadi J, Korotaev VV, de Albuquerque VH. A reference model for internet of things middleware. *IEEE Internet of Things Journal*. 2018; 5(2): 871-883.
[25] Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*. 2016; 3(5): 637-646.
[26] Maksimovic M. *Necessity of the Internet of Thins and Fog Computing Integration*. International Scientific Conference on Information Technology and Data Related Research (SINTEZA). 2017.
[27] Persson P, Angelsmark O. Calvin–merging cloud and iot. *Procedia Computer Science*. 2015; 52: 210-7.
[28] Ptolemy II. (1996.) [Online]. Available: http: //ptolemy.eecs.berkeley.edu. [Accessed 26 Jun. 2018].
[29] Ellis M, Rope D. (2017). Edge Machine Learning with Flogo. [Online]. Available: D2wh20haedxe3f.cloudfront.net. [Accessed 28 Jun. 2018].
[30] 3Scale. Whats is API, your guide to the internet business revolution". [Online]. Available: https: //www.3scale.net/wp-content/uploads/2012/06/What-is-an-API-1.0.pdf [Accessed 28 Jun. 2018].
[31] Fielding RT. Architectural styles and the design of network-based software architectures. PhD dissertation. Irvine: University of California; 2000.
[32] Ngu AH, Gutierrez M, Metsis V, Nepal S, Sheng QZ. IoT middleware: A survey on issues and enabling technologies. *IEEE Internet of Things Journal*. 2017; 4(1): 1-20.