IOT Application Layer Protocols: A Survey

Shivangi Verma
Department of Electronics and Communication Engineering
BBDU, Lucknow, U.P, India.
Email-vshivangi1512@gmail.com

Mr. Ashutosh Rastogi
Assistant Professor
Department of Electronics and Communication Engineering
BBDU, Lucknow, U.P, India.
Email- ashutoshrastogi1984@bbdu.ac.in

Abstract-* The 'Internet Of Things' connects everyday objects or 'things' embedded with sensors and electronic software to the internet which enables them to interact, collect and exchange data. IOT has made its mark in every sphere of our lives viz. business, healthcare, retail, security etc. This creates a requirement for a unified IOT architecture and standardization in IOT protocols to optimize communication. There is a need to device new approaches and technologies suitable for IOT environments for collection, processing and enrichment of the enormous amount of data generated from billions of devices. The application layer enables delivery of data and communication between IOT devices. Communication in IOT world would be non-existent in the absence of application layer protocols. This paper gives a survey of the existing IOT application layer protocols like CoAP, XMPP, MQTT, DDS, AMQP, REST, WebSocket and JMS.

Keywords – ***AMQP, DDS, CoAP, HTTP, IOT, JMS, MQTT, REST, WebSocket, XMPP.***

I. INTRODUCTION

The term 'Internet Of Things' was first devised by Kevin Ashton, a British entrepreneur, in 1999. Since then IOT has come a long way. The scope of internet is expanding beyond computing and computer devices being connected. It is envisaged to provide advanced level of services to society, businesses, etc. IOT will lead to unification of technologies viz. cloud computing, low power embedded systems, machine learning, big data and networking; and an era of ubiquity which means connectivity at anyplace and anytime. An enormous amount of data would be generated by billions of users which has to be handled and processed which requires a much wider and more complex network than the present day internet. There is a need for standardization in IOT protocols which is virtually non-existent to face the challenges of energy efficiency, interoperability, scalability, security, bandwidth management, interfacing, etc[1,2,3].

In an IOT environment, the application layer protocols come into picture where the devices are internet enabled and need communication channels to interact with each other. They are the messaging protocols used by IOT devices to transport data in the absence of which devices would not be able to communicate with each other [4, 5, 6]. They provide for secure and efficient data transmission which is a major factor for the usefulness of IOT devices in today's world. This paper highlights various existing application layer protocols like CoAP, MQTT, XMPP, AMQP, DDS, REST, WebSocket and JMS. We also discuss about their advantages and shortcomings. Based on certain parameters, a comparison between different application layer protocols has been provided. Further, based on our discussion a conclusion and some future work is suggested.

II. PROTOCOLS

1) Constrained Application Protocol (CoAP): CoAP was introduced by IETF CoRE working group for resource constrained nodes and networks. It provides a web transfer protocol based on Representational State Transfer (REST) on top of HTTP functionalities [7, 8] which makes it interoperable with HTTP. As opposed to REST, CoAP runs on UDP to eliminate the overhead involved in TCP and reduce bandwidth requirements and supports both multicast and unicast. But as CoAP is based on REST, the CoAP-REST proxies have straightforward conversion between these two protocols. CoAPHTTP Mapping allows CoAP clients to access HTTP server resources via a reverse proxy which performs translation of HTTP Status Code to the CoAP Response Codes [9]. CoAP enables low power, communication and computational ability devices to utilize RESTful interactions.
CoAP architecture has two sublayers: the messaging sublayer which provides reliability over UDP by utilizing exponential back-off as no built-in error recovery scheme is present in UDP and performs duplication detection; the REST communications are handled by the *request/response sublayer*. The DTLS protocol runs over UDP and provides for secure CoAP transactions. It provides authentication, automatic key management, confidentiality, cryptographic algorithms and data integrity [9]. However, the suitability of DTLS is argued as it was not designed for IOT. Multicast is not supported in DTLS while CoAP supports both unicast and multicast. Also, additional packets are required for DTLS handshakes which reduces the device lifespan, increases congestion and occupies additional resources. CoAP is a lightweight request/response application layer protocol with both asynchronous and synchronous responses. CoAP has following message types and response modes: *confirmable (*requires an acknowledgement either within the ACK or with a separate message*), non-confirmable (*no need for ACK*), reset (*the reception of a message which could not be processed is confirmed*) and acknowledgement (*reception of a confirmable message is confirmed*), piggybacked response* (receiver response is piggybacked on ACK message) and *separate response* (receiver response in a different message than ACK after some time). Similar to HTTP, CoAP also uses GET, PUT, POST and DELETE methods for creating, retrieving, updating and deleting operations respectively. Following are the main components of CoAP protocol:

- Endpoint: Identified by the IP address and UDP port number, the endpoint is the destination or source of a CoAP message.
- Sender: Generator of message (source endpoint).
- Recipient: Consumer of message (destination endpoint).
- Client: Instead of a message, a client can place a request. It is similar to sender or recipient.
- Server: Destination endpoint of a request and the originating endpoint of a response.
- Origin Server: A given Resource is created or resides on this server.
- Intermediary: It is an endpoint which towards an origin server can act as both a server as well as a client.
- Proxy: Performs protocol translation, namespace support and packet forwarding.

Operating systems like TinyOS and Contiki have adapted CoAP. It was designed for M2M applications viz. home automation and smart cities. Some important features of CoAP are [10]:

- *Resource Observation*: On-demand subscriptions using publish-and-subscribe method to monitor internet resources.
- *Block-wise resource transport*: No need to update the complete data in order to exchange transceiver data. This reduces the communication overhead.
- *Resource discovery*: URI paths are used which make use of web link fields in CoRE link format to enable resource discovery for clients.
- *Interacting with HTTP*: CoAP easily interact with HTTP due to the common REST architecture which leads to flexibility of communication.

CoAP libraries exist in most of the programming languages like JAVA, C, C#, Python, Erlang, Go, Javascript, Ruby, etc. Libraries for android as well as iOS are present. CoAP status codes uses the format X.YY which is influenced by HTTP status codes. Example: “4.04 Not Found” in CoAP and “404- Resource Not Found” in HTTP. CoAP has lower consumption of memory and CPU and is simple but on the other side it has bad packet delivery, high latency and difficult to use on complex data type [5, 11].

2) Message Queue Telemetry Transport (MQTT): In 1999 MQTT was introduced by Arlen Nipper of Arcom (now Eurotech) and Andy Stanford Clark of IBM and was standardized by OASIS in 2013[7]. It aims at using middleware and applications to connect network and embedded devices. Routing mechanisms viz. one-to-one, one-to-many, many-to-many; are used for connection operation. To provide simplicity and transition flexibility, asynchronous publish/subscribe model is used in MQTT. Hence, request updates are not needed by the clients which results in a decreasing requirement of network bandwidth and computational resources that further expands the lifespan of battery operated devices. A common example of this is Facebook Messenger. Although MQTT is built on TCP, it is suitable for resource constrained devices which use low bandwidth and unreliable links as it is designed in a way to have low overhead in comparison to other TCP-based application layer protocols. MQTT v3.1 and MQTT-SN V1.2 are two major specifications of MQTT. The main components of MQTT are: subscriber (application interested in sensor data/topic), publisher (lightweight sensors that publish any topic) and broker (connect publishers and subscribers and classify sensor data into topics). Various MQTT methods are: connect, disconnect, subscribe, unsubscribe and publish. MQTT is event driven. Clients do not need to know each other. They only communicate over the topic. This architecture enables highly scalable solution without dependencies between data producers and data consumers. Topics can be exact or wildcards which can be of multilevel. Figure 1 shows the MQTT architecture.

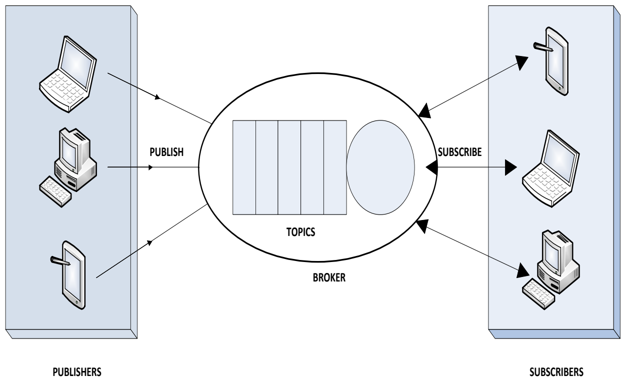

Figure 1. MQTT Architecture

MQTT has three levels of QoS [12]:

- QoS0: ACK is not needed and best effort of the network is used to deliver the message only once. There are no retransmissions and data may get lost.
- QoS1: ACK (called PUBACK) is required. The message is sent for a minimum of one time and duplicate messages might exist. This improves reliability while increasing the overhead.
- QoS2: PUBREC (PUBlish RECeived) and PUBREL (PUBlish RELeased) messages are exchanged to ensure that message is delivered. Here delay is large.

The TLS/SSL protocol provides secure MQTT transactions. It is designed for small code footprints (e.g. eight bit, 256 kb ram) and can run on devices with less than 65kb RAM. Industrial giants actively promote and support MQTT. Mosquitto and HiveMQ are MQTT specific brokers [16]. Several programming languages provide MQTT libraries. An important example of MQTT library is Eclipse Paho. MQTT has lower delays and higher throughput than CoAP for low packet loss but then again CoAP creates less traffic for reliability assurance. It does not support discovery and is not extensible[13].

Secure MQTT (SMQTT) is an extension of MQTT having a broadcast encryption feature. Four main stages of SMQTT are: *setup*, *encryption*, *publish* and *decryption.*

3) Extensile Messaging And Presence Protocol (XMPP): XMPP was developed by Jabber Open Source Community and was standardized by IETF [15] in RFC3920 (now in RFC6120). XMPP is based on both request/response model (allowing bidirectional communication) and publish/subscribe models (allowing multidimensional communications) and runs over TCP. It is a message oriented middleware that runs in a decentralized fashion that gives it high scalability. Clients are connected to servers using XML stanzas that are codes consisting of three parts: message stanza (identify destination and source addresses, kinds and IDs of XMPP entities), presence stanza (notifies consumers about status updates) and *iq* stanza (message senders and receivers are paired). The TLS/SSL (secure socket layer) protocol provides for secure XMPP transactions. XMPP is an instant messaging standard used for video calling, voice calling, chatting and telepresence independent of the operating system. It provides encryption, authentication, privacy measurement, access control and compatibility with various other protocols. Low latency and small message support make it a good choice for IOT. XMPP is extensible which provides it with the ability to work in infrastructure-less environment and its functionality is increased by allowing XEP (XMPP extension protocols) specifications. XMPP is used in Cisco/WebEx, Jabber.org and GoogleTalk [14]. Microsoft has integrated XMPP gateways in their messaging systems and its Microsoft Messenger Service has XMPP interface. XML is used for text based communications in XMPP which may cause additional overhead (this issue can be solved by compressing XML using EXI) and require XML parsing which needs higher computational ability. This protocol consumes more memory and bandwidth and there is no guarantee for QoS. Some overhead may also incur due to the use of TCP in comparison to other protocols. Figure 2 shows XMPP communications.

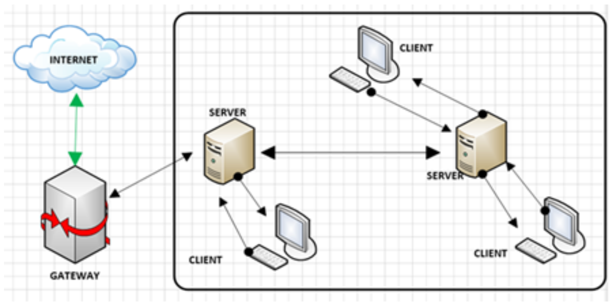

Figure 2. XMPP Communications

4) Advanced Message Queuing Protocol (AMQP): AMQP was introduced by John O’Hara in 2003 at JPMorgan Chase in London and was standardized by OASIS in 2012. AMQP is an open standard binary protocol for message-oriented middleware. AMQP is based on asynchronous publish/subscribe model and runs over TCP. Various ways to transfer message in AMQP are: publish and subscribe, point to point and store-and-forward. The main components of an AMQP system, each of which have the ability to run on autonomous hosts, are:

- Publisher: Creates message and sends it to the broker.
- Subscriber/Consumer: Receives messages through a broker from one or more publishers.

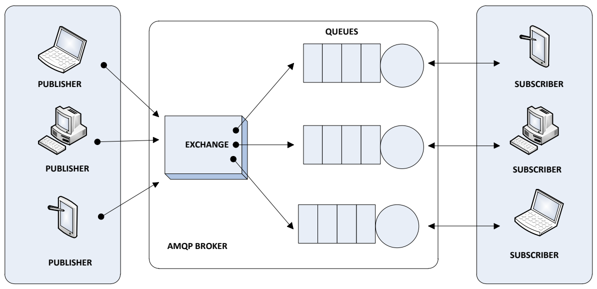

Figure 3. AMQP Publish/Subscribe Mechanism

- Server/Broker: A program or a server which consists of exchanges, virtual hosts, message queues and bindings.
- Virtual Host: Daemon program having services like exchanges, message queues, hosts and bindings.
- Exchange Queue: Messages are routed in a suitable order to appropriate queues.
- Message Queue: It stores the messages and sends them to the receiver. The messages are routed between message queue and exchange queue on the basis of some pre-defined conditions. A messaging layer is defined on top of AMQP transport layer that handles the messaging capabilities. Two types of messages are defined in AMQP: bare message (supplied by sender) and annotated message (seen by receiver).The store and forward feature ensures reliability even after network disruptions. The message delivery guarantees of AMQP are [15]:
- At most once: Message is sent only once without considering the fact if it was delivered or not.
- At least once: Definite delivery of message one time and possibly more.
- Exactly once: Message is delivered only one time.

AMQP connects across organizations (exchanging business messages between applications), technologies, space and time and provides security, reliability, extensibility and interoperability. AMQP is a wire level protocol (which indicates the format of data sent across the network as a byte stream).Therefore, irrespective of the implementation language, any tool which conforms to this format can interoperate with other compliant tools. The TLS/SSL (secure socket layer) protocol provides for secure AMQP transactions. The automation discovery mechanism is not supported by this protocol. It consumes more power than MQTT and is not suitable for real time applications and constrained environments [15].

5) Data Distribution Service (DDS): The DDS was introduced by Real-Time Innovations (a government of US contractor) and Thales Group (a French Defense Company) in 2001. It was developed by Object Management Group (OMG). It is based on publish/subscribe model but has a broker-less architecture which is suitable for real-time constrained IOT and M2M applications. Two layers of interfaces are defined in DDS specifications:

- Data-Centric Publish-Subscribe (DCPS): Responsible for efficient information delivery to the subscribers.
- Data Local Reconstruction Layer (DLRL): This layer is optional and provides for a simple DDS integration into the application layer. Distributed data can be shared between distributed objects with the help of this layer.

DCPS involves the following entities for data flow: *Publisher* (promulgates data); *DataWriter* (applications connect to the publisher using data writer to know about values and variations of data of specific type); *Subscriber* (receives and delivers published data to applications); *DataReader* (used by subscriber for accessing the received data); and *Topic* (DataReaders and DataWriters are related by a topic which is identified by a name and data type). A virtual environment called DDS Domain allows data transmission for connected publishers and subscribers. High reliability and QoS is achieved by multicasting. 23 QoS policies are supported by DDS [14] with the help of which a developer can address a number of communication criteria's like durability, reliability, urgency, priority, security etc. DDS supports dynamic discovery, decentralized data space and adaptive community. Access control is not provided by network/transport security solutions and for high-fanout group communications, TLS/DTLS are not suitable. Hence, DDS aims to provide data-centric security which provides access control to DDS global data space, is multicast friendly, extensible and customizable. DDS libraries exist in most of the programming languages like Ada, C, C++, Java, Scala, Lua, Pharo and Ruby.

6) Representational State Transfer (REST): In 2000, Roy Fielding introduced REST. It was developed by W3C Technical Architecture Group. It is actually an architectural style which uses HTTP as application layer protocol. It works on the principle that every logical entity and physical object is a *resource* having a particular *state* which can be manipulated [14].REST aims to maximize the scalability and independence of component implementation while minimizing the network communications and latency. RESTful service is a service based on REST. It is mainly used to build maintainable, lightweight and scalable web services. REST employs HTTP GET, POST, PUT and DELETE methods and supports request/response messaging model. All the commercial M2M cloud platforms support REST which makes it important for IOT. HTTP libraries are available for all OS therefore it can be easily implemented in smartphone and tablet applications. It uses the built-in HTTP header to indicate data format. The content type may be JASON (JavaScript Object Notation) or XML. REST architecture can completely utilize HTTP features. Following properties and features are desirable for RESTful services: *representations, URIs, uniform interface, statelessness, links between resources and caching [16].* REST has advantages like tighter integration with HTTP to exchange message and secure communications and less parsing complexity but generates significant overhead and power consumption which is not desirable for lightweight IOT applications and does not provide QoS options.

7) WebSocket: WebSocket was first introduced as TCP Connection in HTML 5 specification and standardized by IETF in 2011 as RFC 6544. It has a client-server architecture. Messages are exchanged via asynchronous full-duplex communication channels over a single TCP connection. For this a standardized way is provided by which a server can send content to browser without seeking the client and while keeping the connection open allowing the messages to be passed back and forth. The connection is established by 'upgrading' to HTTP 1.1 from HTTP [11]. A session is established by initializing a handshake between the client and server which is similar to HTTP. HTTP Upgrade header is used in WebSocket handshakes to change from HTTP protocol to WebSocket protocol for achieving compatibility. The session is terminated when both the client and server end it. It does not implements its own reliability mechanisms but WebSocket transactions can be secured over TLS/SSL if needed. WebSocket reduces latency against the half-duplex polling of HTTP, is secure, provides for real-time communication, minimizes overhead and can provide an efficient publish/subscribe messaging system with the use of WebSocket Application Messaging Protocol (WAMP) which is a sub-protocol of WebSocket. However, its client-server architecture is not suitable for resource constrained devices and IOT applications. Most of the browsers like Google Chrome, Internet Explorer, Safari, Opera, Firefox and Microsoft Edge support this protocol [17].

8) Java Message Service (JMS): JMS was first introduced by a Sun Microsystem developed specification and is a part of Java EE (Java Platform, Enterprise Edition). It is a message-oriented application programming interface used to create, send, receive and read messages. JMS supports both the point-to-point (queues of incoming messages are maintained by consumers) and publish/subscribe models (publish messages on a particular 'topic'). JMS consists of following components: *provider, client, producer/publisher, consumer/subscriber, message, queue and topic.* Some common JMS providers are Apache ActiveMQ, IBM MQ, Open Message Queue from Oracle, TLS/SSL and normally used with JAAS API. The main shortcoming of JMS is that it is not a wire-line protocol due to which different JMS implementations from distinct vendor may not interoperate [14, 18]. Also, it is not suitable for real-time applications.

TABLE I. COMPARISON BETWEEN DIFFERENT IOT APPLICATION LAYER PROTOCOLS

S.NO.	**Protocol**	**Transport**	**Architecture**	**Security**	**QoS**	**Mode Of Message Exchange**	**Responsiveness**
1	CoAP	UDP	Request/Response	DTLS	YES	Synchronous/ Asynchronous	Real-time
2	MQTT	TCP	Publish/Subscribe	TLS/SSL	YES	Asynchronous	Real-time
3	AMQP	TCP	Publish/Subscribe	TLS/SSL	YES	Asynchronous	Non Real-time
4	XMPP	TCP	Request/Response Publish/Subscribe	TLS/SSL	NO	Synchronous/ Asynchronous	Real-time
5	DDS	TCP UDP	Broker-less Publish/Subscribe	SSL DTLS	YES	Synchronous/ Asynchronous	Real-time
6	REST	HTTP	Request/Response	HTTPS TLS/SSL	NO	Synchronous	Non Real-time
7	WebSocket	TCP	Client/Server Publish/Subscribe	TLS/SSL	NO	Asynchronous	Real-time
8	JMS	Typically TCP	Point-to-Point Publish/Subscribe	Vendor Specific, TLS/SSL	YES	Asynchronous	Non Real-time

IV.CONCLUSION AND FUTURE SCOPE

The IOT applications have an interdisciplinary nature and need reliable, efficient and secure communication in constrained environments. Out of the many protocols that have been highlighted above, it is difficult to say which one is perfect. The suitability and applicability of any protocol depends on the application for which it is needed. This paper highlights the existing IOT application layer protocol. It also focusses on the pros and cons of each protocol and provides a comprehensive description which can help developers to select appropriate protocol suitable for their application. Table I gives a comparison between different iot application layer protocols.

The future aim is to implement these protocols and provide an experimental comparison for a variety of IOT scenarios. Also there is a possibility to work towards the creation of an integrated platform that can support multiple application layer protocols and dynamically select a suitable one depending upon the application, environment and network conditions. This would result in better integration and enhancement of the overall performance of the IOT systems.

REFERENCES

[1] A. Chaudhary, S.K. Peddoju, and K. Kadarla, “Study of Internet-of-Things Messaging Protocols used for Exchanging Data with External Sources”, 2017, IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems.

[2]R.P. Kumar , Dr. S .Smys, “A Novel Report on Architecture, Protocols and Applications in Internet of Things (IoT)”, Proceedings of the Second International Conference on Inventive Systems and Control (ICISC 2018) IEEE Xplore Compliant - Part Number:CFP18J06-ART, ISBN:978-1-5386-0807-4.

[3] R. Khan, S.U. Khan , R.Zaheer and S.Khan, “ Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges”, 2012 10th International Conference on Frontiers of Information Technology, 978-0-7695-4927-9/12 $26.00 © 2012 IEEE, DOI 10.1109/FIT.2012.53

[4]U. Tandale, Dr.B. Momin, D.P. Seetharam,”An Empirical Study of Application Layer Protocols for IoT”, International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS-2017),978-1-5386-1887-5/17/$31.00 ©2017 IEEE

[5] A. Chakrabarti, “Emerging Open and Standard Protocol Stack for IoT”, Mphasis.

[6] T. Kaukalias and P. Chatzimisios, “Internet of Things (IoT) C Enabling technologies, applications and open issues”, Encyclopedia of Information Science and Technology (3rd Ed.), IGI Global Press, 2014

[7] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things : A Survey on Enabling Technologies, Protocols, and Applications.” IEEE Communication Surveys & Tutorials, Vol. 17, No. 4, Fourth Quarter 2015.

[8] Z. Shelby, K. Hartke, C. Bormann, “ The Constrained Application Protocol (CoAP)”, Computer SciencePublished in RFC 2014,DOI:10.17487/RFC7252

[9] V. Karagiannis, P. Chatzimisios, F. V. Gallego, J.A. Zarate, “A Survey on Application Layer Protocols for the Internet of Things”, Transaction on IoT and Cloud Computing 2015

[10] C. Sharma, Dr. N. K. Gondhi, “Communication Protocol Stack for Constrained IoT Systems”,978-1-5090-6785-5/18/$31.00 © 2018 by IEEE.

[11] S. Bandyopadhyay, A. Bhattacharyya, “Lightweight Internet Protocols for Web Enablement of Sensors using Constrained Gateway Devices”,2013,International Conference on Computing, Networking and Communications, Workshops Cyber Physical System; 978-1-4673-5288-8/13/$31.00 ©2013 IEEE.

[12] S. Mijovic, E. Shehu, C. Buratti, “Comparing Application Layer Protocols for the Internet of Things via Experimentation”, 2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 978-1-5090-1131-5/16/$31.00 ©2016 IEEE.

[13] D. Thangavel, X. Ma, A. Valera, H.X. Tan, C.K.Y. Tan, “Performance Evaluation of MQTT and CoAP via a Common Middleware”, IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 21-24 April 2014, pp.16.

[14] V. Gazis, M.G¨ortz, M. Huber, A. Leonardi, K. Mathioudakis, A. Wiesmaier, F. Zeiger, E. Vasilomanolakis, “A Survey of Technologies for the Internet of Things”, 978-1-4799-5344-8/15/$31.00 ©2015 IEEE

[15] M.B. Yassein, M.Q. Shatnawi, D. Al-zoubi, “Application Layer Protocols for the Internet of Things: A survey”, 978-1-5090-5579-1/16/$31.00 ©2016 IEEE.

[16] M. Asim, “A Survey on Application Layer Protocols for Internet of Things (IoT)”, Volume 8, No. 3, March – April 2017 International Journal of Advanced Research in Computer Science.

[17] P. Sethi and S.R. Sarangi “Internet of Things: Architectures, Protocols, and Applications”,Hindawi Journal of Electrical and Computer Engineering Volume 2017, Article ID 932403

[18] R.Want, B.N. Schilit, and S.Jenson, “Enabling the Internet of Things”, Computer Published By The IEEE Computer Society, 0018-9162/15/$31.00 © 2015 IEEE