International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 7, Issue 8, August 2017)

A Survey on the Challenges and Solutions in the Design of an IoT Application layer Protocol

Arunita Kundaliya[1], Hem Dutt Dabral[2]

[1]*M.Tech, School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, 110067, India*
[2]*M.S. (Software Systems) Birla Institute of Technology & Science, Pilani - 333031. Rajasthan, India*

***Abstract*— In recent years, the increasing human desire to be connected with anything anytime anywhere has coupled with the swift development in the areas of information technology (IT), sophisticated and ubiquitous sensing and development in the field of miniature device in electronic industry. This has resulted in the formation of a well connected community where objects are connected to mobile devices and the Internet. Moreover, these objects communicate with each other frequently. At the heart of this well-connected community is IoT, Internet of Things, that involves "Internet" services and "Things", which are end-devices such as sensors nodes. These end-devices contribute data to the actuators and/or to cloud or to gateway. The rapid increase in the volume of interconnected devices together with the future requirement of the network enabled devices to communicate with each other and/or with devices on internetwork has necessitated in the development of the set of rules i.e. protocols. In the protocol stack, application layer is responsible for providing services to the end-users and in determining the set of protocols to be used for device-to device and inter-process communication. A major design challenge in the application layer protocol in IoT is to develop an IP-compatible protocol which is bandwidth-efficient, energy-efficient and is capable of working with hardware resources which are limited in computational capability and battery power.**

In this paper, we studied existing application layer protocols- CoAP, MQTT, AMQP, XMPP, RESTFUL services, DDS, WebSocket and provided a comparative analysis of the protocols studied. The performance of these protocols is compared on the parameters such as protocol architecture, scalability, privacy, security, latency, memory foot print used, computational complexity, bandwidth requirement, support for power constrained devices, messaging model used, heterogeneity, QoS support, robustness, and fault tolerance.

***Keywords*— 6LoWPAN, AMQP, CoAP, DDS, DTLS, IEEE 802.15.4, LoRaWAN, MQTT, MOM, Neul, RESTFUL services, RPL, Sigfox,Thread, WebSocket, XMPP.**

I. INTRODUCTION

In the last decade, Internet is abuzz with the term "Smart world" which in itself includes Smart devices, Smart phones, Smart watches, Smart cars, Smart homes, Smart cities and the list continues.

The word "Smart" is touching every aspect of human life. For example, these days many buildings already have sensors to save energy, home automation is occurring where people want to operate or control home appliances like AC, TV, fridge etc via small applications with Internet running on their smart phones. People are running many useful applications on their Smart phones with sensors. Cars, taxis, and traffic lights too have sensors to improve safety and transportation, industrial plants are connecting to the Internet and health care services are trying to use the increased home sensing via sensors in lights, TVs, body nodes which are used for measuring heart rate and temperature, and sleep apnea machines to support remote medicine[1]. The core of these "Smart" services is IoT, "Internet of Things", which is heavily dependent on the ubiquitous sensing, actuators and the presence of communication subsystem. IoT commonly refers to the interconnection of different types of computing devices to support various monitoring and control applications which requires IoT applications to interact with lots of sensors and actuators to perform the task of controlling and monitoring of their ambient environment. The end-devices in the IoT network often contain a large number of low-end, resource-constrained devices but the devices may vary in capability ranging from constrained devices with sensors to smart devices with some computing capabilities to devices with high processing capacities. These devices are required to operate over long time periods (e.g., a year) on battery. The design of these devices is often driven by the low manufacturing and operational cost together with efficient energy utilization. The design of a simple IoT application involves large number of deployed and interconnected sensors, actuators, gateways, and optionally cloud based application. The software and hardware components required to design a simple IoT application is available in [26]. The sensors measure the parameters of the physical environment and send the data to a gateway. The gateway aggregates the data from various sensors and then sends it to a server/broker. The clients may connect to the server to obtain the sensor data.

This integration of sensor devices into the Internet and the human desire to monitor and control the device parameters requires an IP-compatible application protocol. The protocol is desired to be efficient in terms of bandwidth and energy and should be capable of working with limited capability hardware resources. As most IoT applications interact with lots of sensors, actuator, gateways and/or cloud to perform various monitoring and control tasks other features are also desired in the protocol. These features include scalable support for naming configuration for the existing devices, discovery of new devices added to the framework, security protection and privacy on the data acquisition and actuation operations, message model imposing minimum overhead on protocol operation, policies to handle dynamic network environments such as intermittent connectivity, QoS as per user requirement, issues related to interoperability of heterogeneous devices, Robustness and fault tolerance of the IoT network.

The lack of optimized application protocols can cause performance degradation in terms of bandwidth usage and battery lifetime by imposing high overhead on protocol operation and resource utilization. The subsequent sections of the paper presents, commonly used protocol stack in an IoT application, challenges encountered in the design of an application layer protocol, state-of-art application layer protocols and comparison among these protocols on the basis of the parameters discussed.

II. PROTOCOL STACK IN IOT

In this section, we describe the commonly used protocol stack, in-line with TCP/IP protocol stack.

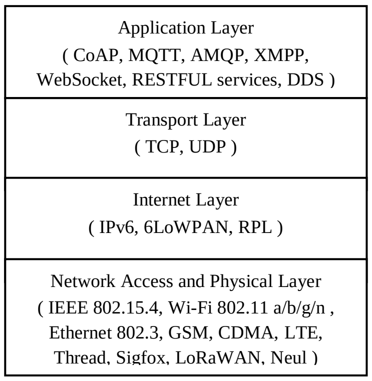
Application Layer
(CoAP, MQTT, AMQP, XMPP, WebSocket, RESTFUL services, DDS)
Transport Layer
(TCP, UDP)
Internet Layer
(IPv6, 6LoWPAN, RPL)
Network Access and Physical Layer
(IEEE 802.15.4, Wi-Fi 802.11 a/b/g/n , Ethernet 802.3, GSM, CDMA, LTE, Thread, Sigfox, LoRaWAN, Neul)

Figure 1: Protocol Stack used in an IOT application

- *Application Layer:* The commonly used protocols at application layer are CoAP, MQTT, XMPP, AMQP, RESTful Services, DDS, WebSocket. These protocols are described at great length in later section of the paper.
- *Transport Layer:* The protocols used here are TCP and UDP for example, CoAP uses UDP while MQTT is run over TCP. UDP is preferred in IoT application layer protocol because it is connectionless and has low overhead due to its smaller packet size. Thus, it requires smaller memory size and lower computational power device to examine and forward the UDP packets. TCP being a connection-oriented protocol needs to reserve the resources while establishing the communication path. Moreover, its larger packet size needs more memory and higher computational power devices.
- *Internet Layer:* It uses 6LoWPAN, an acronym for IPv6 over low power wireless personal area networks. It is an open standard defined under RFC 6282 by the Internet Engineering Task Force (IETF). It enables communication using IPv6 over the IEEE 802.15.4 [11] protocol, which support much smaller packet sizes. This standard defines an adaptation layer between the 802.15.4 link layer and the transport layer. 6LoWPAN devices can communicate with all other IP based devices on the Internet. RPL used in Internet layer is a Routing protocol for low-power and lossy network [12].
- *Network Access and Physical Layer:* IEEE 802.15.4 is a standard for wireless communication that defines the Physical layer (PHY) and Media Access Control (MAC) layers. It is standardized by the IEEE similar to IEEE 802.3 for Ethernet. IEEE 802.15.4 focuses on communication between devices in constrained environment with low resources (memory, power and bandwidth). It enables communication between compact and inexpensive low power embedded devices that need longer battery life. IEEE 802.11 standard is for wireless LANs (WLANs) or Wi-Fi. Thread is a new IP-based IPv6 network protocol mainly designed as a complement to Wi-Fi. Sigfox is wide-range technology using UNB (Ultra Narrow Band). Neul operates in sub-1GHz band i.e. high-quality UHF (Ultra High Frequency) band. LoRaWAN is a Low Power WAN using license free sub Gigahertz radio frequency bands. It is a MAC layer protocol for managing communication between low power WAN gateways and end-devices.

III. Challenges At Application Layer In An IoT Application

The rate at which the volume of interconnected "Things" i.e., end-devices is increasing, it seems in near future trillions of "Things (objects/end-devices)" will be connected to the Internet. To allow users, to be able to control this large volume of objects it is necessary to develop an IP-compatible application protocol. The protocol is expected to have an adequate architecture that permits easy connectivity, control, and communications that will be useful in developing IoT applications. In this section, we enumerate some of the challenges faced while designing an application layer protocol.

- *Protocol Architecture:* As millions of devices are connected to the Internet and many more to follow this course, there is a rise in the demand of protocol architecture at application layer that will allow easy connectivity, control, use, and communication with the underlying devices and/or with other devices on Internet.
- *Heterogeneity of end-devices:* The Internet of Things (IoT) aims to interconnect large numbers of heterogeneous devices to provide advanced applications that can improve our quality of life. This implies the efficient and seamless interconnectivity of large number of devices, allowing them to be discovered and accessed by the services in an abstracted way. A major challenge is to provide support for interconnecting heterogeneous devices. The reason is that integrating multiple systems is very challenging as each individual system has its own assumptions and strategy to control the physical world variables without much knowledge of the other systems [1].
- *Scaling:* The colossal increase in the volume of smart devices deployed on Internet suggests that eventually more than hundreds of millions of things will be on Internet. This massive scaling asserts challenges such as how to name, authenticate access, use, and provide support and security for such a large scale of things. Also, addition of networks and devices must not deteriorate the performance of the network, the functioning of the devices, the reliability of the data over the network and the effective use of the devices from the user interface.
- *Robustness:* The most challenging task in an IoT application is to ensure that the IoT application is able to withstand its dynamically changing wireless environment with frequent link failures due to interconnectivity of used Wi-Fi devices.

  Another concern in the operation of IoT devices is clock synchronization. It is common for the devices to have synchronized clocks and have coherent set of parameter setting such as consistent wake-up/sleep schedules, using appropriate power level for communication. But a clock drift causes nodes to have different times which may result in application failure [1]. Thus, robustness describes how the protocols will behave when underlying end-devices or technology used by them changes. It is desirable that the protocol should seamlessly allow the changes without requiring user application to accommodate the changes.
- *Security:* In modern times, the connectivity of sensors with Internet has led to a rapid increase in the real-time sensor data made accessible to all Internet users. This has made the IoT applications vulnerable to security attacks where integrity of IoT application may be compromised. The security problem is further aggravated due to transient and random failure of devices which is common in an IoT application. These vulnerabilities can be easily exploited by hackers. The situation is worse for the IoT applications as most IoT applications use resource-constrained devices. Mitigation of security risk involves ascertaining current security threat and anticipating future security problems together with remedial measures to evade security threats. These measures need heavy computations and large memory requirement. The security issue becomes more difficult when legacy devices are used in the application. The legacy devices may themselves be having their own security measures but those measures may not be compatible with measures devised for the intended IoT application.
- *Privacy:* Most IoT applications are customized and are designed to work for a particular user. It is important to ensure that the collected data and subsequent inferences are associated with the intended user.
- *Openness:* Traditionally, majority of sensor based systems have been closed e.g. cars, airplanes etc have their own networked sensor systems operating within that vehicle [1]. But, evolving user demands require diverse systems to communicate with each other for example we may want cars to communicate with each other to avoid collisions. To reap these benefits, the systems need to be open so that communication between diverse devices can be established. The most challenging task is to balance the benefits of openness with access to the functionality of concerned devices, along with security and privacy of the IoT application.

- *Bandwidth efficiency:* This issue is primarily of physical layer. The IoT devices use several LPWA (Low Power, Wide Area) techniques such as RIFD, UWB (Ultra Wide Band), Blue tooth, NFC (Near Field Communication) for device communication. The same bandwidth may be used by devices which are not part of the IoT application. This can lead to interference with and between LPWA transmissions in the closed vicinity of the deployed IoT application. The real issue is how the application layer takes note of the interference and handles it by reducing channel contention for optimal usage of the bandwidth.
- *Energy efficiency:* Majority of the devices used in an IoT application is battery operated with no means to recharge their battery or replacement of battery is infeasible. This necessitates that the application protocol employ low power consumption technique such as using simple message formats that needs less computation and power to transmit, decode and receive message.
- *Addressing issues:* The success of an IoT application is heavily dependent on its ability to uniquely identify "Things" (end-devices or objects). This will allow unique identification of billions of devices and also control of remote devices through the Internet. The key issue involved for every element that is already connected and those that are going to be connected is how these objects can be identified by their unique identification, location and functionalities. Addressing these issues requires formation of addressing schemes, may be similar in line with Uniform Resource Name (URN) system. URN creates replicas of the resources that can be accessed through the URL [6].
- *Message model:* To communicate with the under lying devices or with remote devices, an IoT application needs to exchange messages. The syntax and semantics of the messages should consider the heterogeneous nature of wireless nodes, variable data types used by the devices, resolving concurrent operations by the underlying or remote devices and handling rapid generation of data from the interconnected devices.
- *Information sharing:* It addresses a key issue of how the devices in an across the applications can share information of their ambient environment. [2,3,4] Consider a scenario where several systems responsible for energy management (controlling thermostat, doors and windows) and home health care (controlling lights, TV, sleep apnea machines and body sensors measuring temperature and heart beat) are integrated. If these systems share information then energy management system would adjust room temperature as per need of user of home health care system. Also, the integration would not allow medical appliances to switch to power saving mode when they are being used.
- *Synchronous/Asynchronous mode:* This mode of communication determines whether the message exchange takes place in synchronous/ asynchronous mode. In synchronous mode a request is submitted by the application to a device and a response from the devices is awaited while in asynchronous mode the application submits a request and continues its own execution. When the response is received from the device, it is returned to the application through callback.
- *Latency/Response in real time:* Knowing the real-time nature of many IoT applications, a quick response in expected, especially in safety critical ones. These applications require run time assurance which are not expected to change and is known to the user. For example, consider a fire fighting system deployed in a sky scraper office building to detect fires, alert fire stations and aid in evacuation. A late response will jeopardize the entire fire fighting system.
- *Memory footprint:* It denotes main memory that is used or referenced by the protocol. IoT devices are often constrained in memory capacity. Therefore, it is a challenge to use complex algorithms which often need more memory than the tiny devices have.
- *Computational Complexity:* It denotes the amount of resources needed by the protocol in terms of time and space complexity. Most IoT devices are limited in computing power which makes it difficult to run complex algorithms such as cryptographic algorithms.

IV. The State-Of-Art Application Layer Protocol

In this section, we discuss some commonly used application layer protocol in an IoT application.

A. *CoAP (Constrained Application Protocol)*

CoAP was designed by the IETF for use with low-power and constrained networks. It supports connectivity of these simple low-power electronic devices such as wireless sensors with Internet based systems. CoAP is a good choice for devices, which operate on battery or employ energy harvesting techniques. The protocol gets easily translated to HTTP for simplified integration with the web, while meeting the specialized requirements such as: multicast support, very low overhead, and simplicity of constrained devices [10].

The protocol is semantically aligned with HTTP and even has a one-to-one mapping to and from HTTP. The protocol is based on request/response architecture where each device acts as "client" or "server". Here, a request is sent to the device and a response from the device is awaited to get connected. Clients may use GET, PUT, POST and DELETE the standard HTTP messages. It is run over UDP and the messages are exchanged asynchronously. CoAP provides Datagram Transport Layer security (DTLS) which is a TLS/SSL counterpart that runs on UDP. DTLS provides the same security features on UDP or Datagrams, the way TLS/SSL takes care of security for TCP communication. DLTS has poor scalability but improved robustness than TLS. CoAP messages are encoded in a simple binary format. Packets are simple to generate and can be parsed in place without consuming additional RAM in constrained devices.

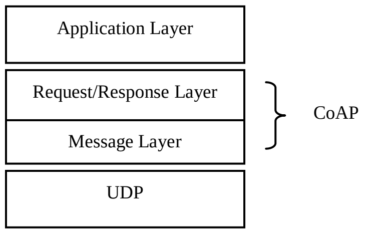

Figure 2: CoAP Architecture

The protocol architecture is divided into two layers

- *Message layer:* This layer is responsible for reliability and sequencing.
- *Request/Response layer:* This layer is responsible for mapping requests to responses and their semantics.

CoAP runs over the unreliable UDP so, it has its own mechanism for achieving reliability. It provides the desired Quality of Service (QoS) levels via two messages-

- *Confirmable messages*: These messages must be acknowledged by the receiver with an acknowledgment (Ack) packet.
- *Nonconfirmable messages*: These messages are "fire and forget" type i.e., a message that does not need to be acknowledged.

CoAP uses the "coap" and "coaps" URI schemes for identifying and locating CoAP resources. Resources are organized hierarchically and governed by a potential CoAP origin server listening for CoAP requests ("coap") or DTLS-secured CoAP requests ("coaps") on a given UDP port. The CoAP URI scheme is described as [16]

coap-URI = "coap:" "//" host [":" port] path-abempty ["?" query]

B. *MQTT (Message Queue Telemetry Transport)*

It is an ISO standard based, on lightweight publish-subscribe messaging protocol. It runs over TCP/IP protocol. It is designed for applications with remote locations where a small code footprint is required or network bandwidth is limited. Publish/Subscribe is event-based where messages can be pushed to clients. A MQTT broker is at the center of communication and is responsible for dispatching messages between senders and intended receiver. Each client which publishes a message to broker includes a topic into the message. The topic is routing information for the broker. Each client that wants to receive messages subscribes to a topic and all messages with related topic are delivered to it by the broker. It defines methods (Connect, Disconnect, Subscribe, Unsubscribe, and Publish) to indicate desired action to be performed on a resource. The messages need not be responded implying low overhead as compared to other TCP-based application layer protocols. MQTT has latency in seconds.

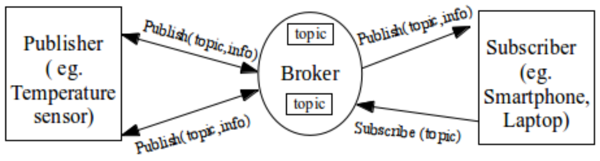

Figure 3: MQTT Publish/Subscribe Architecture

To provide security it uses TLS/SSL. MQTT ensures reliability by providing three levels of QoS levels:

- *Fire and forget*: Message is sent once with no need of acknowledgement.
- *Delivered at least once*: A message is sent at least once and an acknowledgement is received.
- *Delivered exactly once*: A four-way handshake mechanism is used to ensure that the message is delivered exactly once.

Eclipse Mosquitto is an open source message broker that implements MQTT.

C. *AMQP (Advanced Message Queuing Protocol)*

[18] It is an open standard, wire-level application layer protocol that provides semantic framework for message-oriented middleware (MOM). A wire-level protocol is a description of the format of the data that is sent across the network as a stream of bytes.

MOM is an software or hardware infrastructure supporting sending and receiving messages between distributed systems and allows application modules to be distributed over heterogeneous platforms. The main characteristics of AMQP are message orientation, queuing, routing, reliability, and interoperability due to MOM and security. Messages are sent over reliable TCP connection and can be stored in message queues before being sent to receivers. Message queuing is a mechanism by which sender sends a message to receiver asynchronously. Sender and receiver may or may not interact with the message at the same time. The message is sent to a queue where it is stored until the receiver retrieves the message. These queue(s) can be on a single or multiple servers and can reside in-memory or persisted on disk. AMQP addresses the problem of transmitting value-bearing messages across and between applications in a synchronous manner. AMQP can use both store-and-forward and publish/subscribe semantics in a single application. The basic architecture is simple, where client applications called producers create messages and deliver it to an AMQP server also called broker. Inside the broker the messages are routed and filtered and arrive to queues where other applications called consumers are connected and receive the messages.

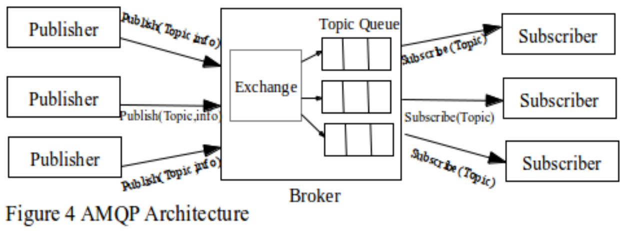

Figure 4 AMQP Architecture

Communications from the publishers to exchanges and from queues to subscribers use TCP. Also, endpoints must acknowledge acceptance of each message. AMQP is designed to support message oriented communication with QoS such as

- *At-most-once* where each message is delivered once or never.
- *At-least-once* where the message is certain to be delivered at-least once.
- *Exactly-once* where the message is certainly delivered but only once.

It provides security via TLS and/or SASL (Simple Authentication and Security Layer).

D. XMPP (Extensible Messaging and Presence Protocol)

[19, 24] Its an open standard application layer protocol standardized by IETF for MOM and is based on XML, making person-to-person communication natural. It provides near-real-time, asynchronous exchange of structured data between two or more connected device. XMPP was developed for Instant Messaging to connect people to other people via text messages. Its response time is in seconds and thus good for person-to-person communication. It is designed to be extensible and uses publish/subscribe architecture.

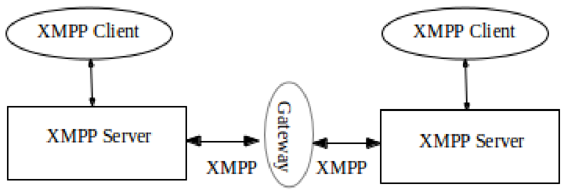

Figure 5: XMPP Architecture

Its key strength is its addressing scheme which offers an easy way to address devices. Security is provided by TLS/SASL. It does not support QoS but supports interoperability and is scalable. XMPP is not designed to be fast. In fact, most implementations use polling or checking for updates only on demand. It provides an easy way to connect your home thermostat to a web server so that it can be accessed via Smartphone.

E. RESTFUL Services (Representational State Transfer)

[22, 5] The Representational State Transfer (REST) is not really a protocol but an architectural style. It is layered, stateless and support caching. It uses Client/Server architecture. Clients initiates request to server. The server processes the request and replies back to client. This request/reply message exchange is done using HTTP. REST uses the available HTTP methods GET, POST, PUT, and DELETE to provide a resource-oriented messaging system where all actions can be performed simply by using the synchronous request/response HTTP commands.

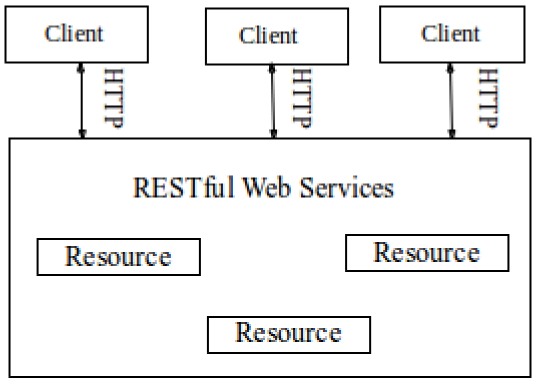

Figure 6: RESTful Services Architecture

It uses the built-in accept header of HTTP to indicate the format of the data that it contains which can be XML or JSON (JavaScript Object Notation) and depends on the HTTP server and its configuration. Message latency is of several seconds. The universal availability of HTTP stack for various platforms has made REST popular for web API design model. The protocol is scalable and supports interoperability. It can make use of TLS/SSL for security. In REST, the HTTP polling repeats header information each time when the data transmission rate increases, this increases the latency of the application. REST can be easily implemented in Smartphone and tablet applications as it only needs an easily available HTTP library. Restful HTTP over TCP is used primarily for connecting consumer premise devices e.g. home energy management.

F. *Websocket*

[5, 20, 25] It is not a protocol that was designed for use in IoT product deployments but instead, for use with the web. It lets web servers and web browsers to communicate continuously by using a message-based, full-duplex, real-time, low-latency communication between a server and a client over TCP connection. To establish a WebSocket connection, the client sends a WebSocket handshake request, for which the server returns a WebSocket handshake response. Clients and Servers can exchange messages in an asynchronous full-duplex connection.

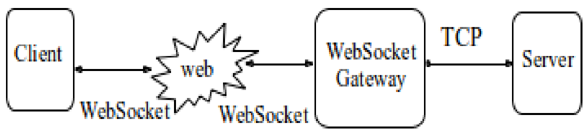

Figure 7: WebSocket Architecture

Websocket was created to reduce the Internet communication overhead while providing real-time full-duplex communications. There is also a Websocket sub-protocol called Websocket Application Messaging Protocol (WAMP) that provides publish/subscribe messaging systems. It may use TLS/SSL for security. It incurs more overhead as it requires a TCP implementation, along with an HTTP implementation for the initial connection setup. Websocket is not designed for resource constrained devices but it is designed for real-time communication. It is secure, minimizes overhead and with the use of WAMP it can provide efficient messaging systems.

G. *DDS (Data Distribution Service for Real-time Systems)*

[21] It is an Object Management Group (OMG) machine-to-machine standard which enables network interoperability for connected end-devices. DDS connects devices together into working distributed applications. DDS is a data-centric middleware standard which targets devices that directly use device data. It provides scalability, QoS, and high-performance of data exchanges using publish–subscribe pattern. Nodes that produce information are publishers who create "topics" (e.g., temperature, location, pressure) and publish "samples". Any node can be a publisher, subscriber, or both simultaneously. DDS delivers the samples to subscribers that declare an interest in that topic. DDS can be deployed in platforms requiring low-footprint devices to the Cloud and supports efficient bandwidth usage. It provides a global data space for analytics and enables flexible real-time system integration. DDS handles message delivery transparently without requiring intervention from the user applications, which include:

- determining who should receive the messages
- where the recipients are located
- what happens, if the messages cannot be delivered

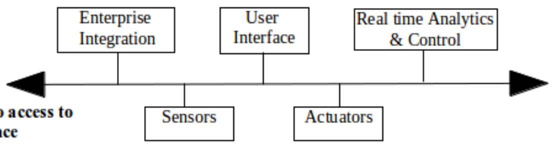
Enterprise Integration
User Interface
Real time Analytics & Control
Databus to access to shared space
Sensors
Actuators

Figure 8: DDS architecture

DDS is both language and operating system independent. Its shared memory based deployment architecture encourages low latency. DDS runs over UDP but it may also use TCP. To provide interoperability of devices it uses a dynamic protocol, namely DDSI (DDS interoperability wire protocol). DDS's applications include military systems, hospital integration, medical imaging etc.

The protocols described above are compared and their comparison is presented in the table given below along with the parameters used for comparison.

Table 1:
Comparison of IoT application layer protocols

Protocols → Parameters ↓	CoAP	MQTT	AMQP	XMPP	RESTful Services	Web-Socket	DDS
Architec-ture	Request/ Response	Publish/ Subscribe	Publish/ Subscribe	Publish/Subscribe	Request/ Response	Handshake over full-duplex connection	Publish/ Subscribe
Interoper-ability	Yes (Seamless transformation to HTTP request)	Yes	Yes	Yes	Yes	Vertical Scaling is easier than Horizontal scaling	Yes
Scalability	Low	High	Low	High	High	High	High
Robust-ness (Fault Tolerance)	De-centralized	Broker is the single point of failure	Implementation specific	De-centralized	De-centralized	De-centralized	De-centralized
Transport	UDP	TCP	TCP	TCP	TCP	TCP	UDP, TCP
Security	DTLS	TLS/SSL	TLS/SSL	TLS/SSL	TLS/SSL	TLS/SSL	TLS/SSL, DTLS
Bandwidth Require-ment	Low	Low	High	Low	High	Low	Low
Suitable for Energy constrain devices	Yes	Yes	No	No	No	No	No
Message/ Data Centric	Message	Message	Message	Message	Message	Message	Data
Synchronous/ Asynchronous	Asynchro-nous	Asynchro-nous	Asynchro-nous	Synchro-nous, Asynchro-nous	Synchro-nous	Asynchro-nous	Asynchro-nous
Memory Footprint	Low	Low	High	High	Low	Low	Low
Computa-tional complex-ity	Low	Low	High	High	High	High	Low
QoS Option	Yes	Yes	Yes	No	No	No	Yes

V. CONCLUSION

The emerging trend in Internet of Things (IoT) is aiming to improve the quality of life by connecting smart devices, technologies, and applications.

In this paper we presented an overview of the state of art application layer protocol in an IoT stack to gain an insight into the protocols architecture and understand some of the challenges and issues that pertain to the design and deployment of an IoT application layer protocol. A deeper understanding of these protocols and the specific requirement(s) of applications is necessary to select the most suitable protocol for the application at hand.

We found that the fundamental goals of all protocols differ, their architectures differ, and also their capabilities. It is important to understand the requirement(s) of the intended application and choose the protocol carefully, especially when key system requirements such as performance, scalability, QoS, interoperability, fault tolerance, resource constraints (if any), and security are taken into account. Some protocols may be best suited for resource constrained devices while few others may be well suited for real-time, event-based communication. Some protocols scale better than others while other protocols may be supporting secured transmission with little communication overhead.

REFERENCES

[1] Stankovic, John A. "Research directions for the Internet of things." IEEE Internet of Things Journal 1.1 (2014): 3-9.

[2] P. A. Vicaire, Z. Xie, E. Hoque, and J. A. Stankovic, Physicalnet: A Generic Framework for Managing and Programming across Pervasive Computing Networks, RTAS, 2010.

[3] J. Lu, T. Sookoor, V. Srinivasan, G. Gao, B. Holben J. Stankovic, E. Field, and K. Whitehouse, The Smart Thermostat: Using Occupancy Sensors to Save Energy in Homes, ACM SenSys, 2010.

a. Wood, J. Stankovic, G. Virone, L. Selavo, T. He, Q. Cao, T. Doan, Y. Wu, L. FANG, and R. Stoleru, Context-Aware Wireless Sensor Networks for Assisted Living and Residential Monitoring. IEEE Network 22, 4, Jul./Aug. 2008, pp. 26–33.

[4] Karagiannis, Vasileios, et al. "A survey on application layer protocols for the internet of things." Transaction on IoT and Cloud Computing 3.1 (2015): 11-17.

[5] Gubbi, Jayavardhana, et al. "Internet of Things (IoT): A vision, architectural elements, and future directions." Future generation computer systems 29.7 (2013): 1645-1660.

[6] Raisinghani, Vijay T., and Sridhar Iyer. "Cross-layer feedback architecture for mobile device protocol stacks." IEEECommunications Magazine 44.1 (2006): 85-92.

[7] Guinard, Dominique, Vlad Trifa, and Erik Wilde. "A resource oriented architecture for the web of things." Internet of Things (IOT), 2010. IEEE, 2010.

[8] Hong, Yao-Win, and Anna Scaglione. "A scalable synchronization protocol for large scale sensor networks and its applications." IEEE Journal on Selected Areas in Communications 23.5 (2005): 1085-1099.

[9] Palattella, Maria Rita, et al. "Standardized protocol stack for the internet of (important) things." IEEE communications surveys & tutorials 15.3 (2013): 1389-1406.

[10] D. Culler and S. Chakrabarti, "6lowpan: incorporating IEEE 802.15. 4 into the IP architecture, IPSO Alliance," White Paper, 2009.

[11] Jain, Raj. "Networking Layer Protocols for Internet of Things: 6LoWPAN and RPL." (2015).

[12] MQTT Source: http://mqtt.org/

[13] CoAP Source: https://en.wikipedia.org/wiki/Constrained_Application_Protocol

[14] CoAP Source: http://coap.technology/

[15] CoAP Source: https://tools.ietf.org/html/rfc7252/

[16] XMPP source: https://xmpp.org/

[17] AMQP Source: https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol

[18] XMPP Source: https://en.wikipedia.org/wiki/XMPP/

[19] WebSocket Source: https://en.wikipedia.org/wiki/Websocket

[20] DDS Source: https://en.wikipedia.org/wiki/Data_Distribution_Service

[21] Representational State Transfer Source: https://en.wikipedia.org/wiki/Representational_State_Transfer

[22] Babovic, Zoran B., Jelica Protic, and Veljko Milutinovic. "Web performance evaluation for Internet of things applications." IEEE Access 4 (2016): 6974-6992.

[23] XMPP Source: https://xmpp.org/rfcs/rfc3920.html

[24] WebSocket Source: https://websocket.org/

[25] Arunita Kundaliya, Hem Dutt Dabral, " A survey of Hardware and Software Components Required to Build an IoT solution. "International Journal of Emerging Technology and Advanced Engineering, vol 7, Issue 7, July 2017.