A Comprehensive Survey on Application Layer Protocols in the Internet of Things

Hemant Sharma, Ankur Gupta, and Madhavi Latha Challa

Abstract The number of connected IoT devices deployed globally is increasing at a tremendous rate. The IoT has a wide range of application domains providing extensive IoT-based services. The basic idea is to deliver a new set of applications where smart devices collaborate without any human interference. The variability and visibility of IoT based services lead to the development of the full spectrum of protocols. With the exponential growth of applications, it is essential to analyze the existing application layer protocols being used to exchange information among devices. In this paper, a detailed analysis of existing popular application layer protocols like Constrained Application Protocol (CoAP), Message Queue Telemetry Transport (MQTT), Advance Message Queuing Protocol (AMQP), Extensible Messaging and Presence Protocol (XMPP), etc. has been done to categorize them based on well-known properties such as architecture, energy consumption, reliability, QoS, and security aspects.

Keywords Internet of things · CoAP · MQTT · XMPP · AMQP

1 Introduction

The Internet of Things is becoming very popular nowadays both from the technical and commercial point of view due to its simplicity, low cost, and easy deployment [1]. IoT is emerging at this rate because of its various real-time applications deployed in different domains like transportation, energy, healthcare, agriculture, education,

H. Sharma · M. L. Challa
University of Gondar, Gondar, Ethiopia
e-mail: hem.s1209@gmail.com

M. L. Challa
e-mail: saidatta2009@gmail.com

A. Gupta (✉)
MNIT Jaipur, Jaipur, Rajasthan, India
e-mail: gupta1990.cs@gmail.com

V. K. Gunjan and J. M. Zurada (eds.), *Proceedings of International Conference on Recent Trends in Machine Learning, IoT, Smart Cities and Applications*, Advances in Intelligent Systems and Computing 1245,
https://doi.org/10.1007/978-981-15-7234-0_17

smart vehicles, environment, industrial automation, etc. The IoT can be defined as "Enabling the physical objects to sense, think, communicate, control, and have opportunities to interact and collaborate with other physical objects over the network." To enable these abilities, the physical objects are embedded with electronic computers, sensors, software, and network to monitor collecting and exchanging data among them, and hence called smart objects. These smart objects must be low-cost, energy efficient, secure, and interoperable with software applications. Figure 1 shows the domain-specific applications with smart objects performing their supposed task in the vast field of applications of IoT [2].

According to Cisco IBSG [3], IoT starts from a point when the number of devices connected to the internet exceeds the number of people connected. In 2003, 500 million devices were connected to the internet, and the world population was about 6.3 billion giving the ratio of 0.08 devices to people. Therefore, IoT didn't exist at that time based on CISCO's definition. IoT came into existence between the years 2008 and 2009, where the number of devices exceeds the world population. CISCO predicted that 50 billion devices would be connected by the year 2020 in a world with a projected entire population of 7.6 billion. It gives a ratio of nearly seven devices to people having approximately 80 times growth overpopulation increase, as shown in Fig. 2. It may be noted that the internet never remains static, these estimates are based on the information known to be true today, and also the ratio prediction is based on

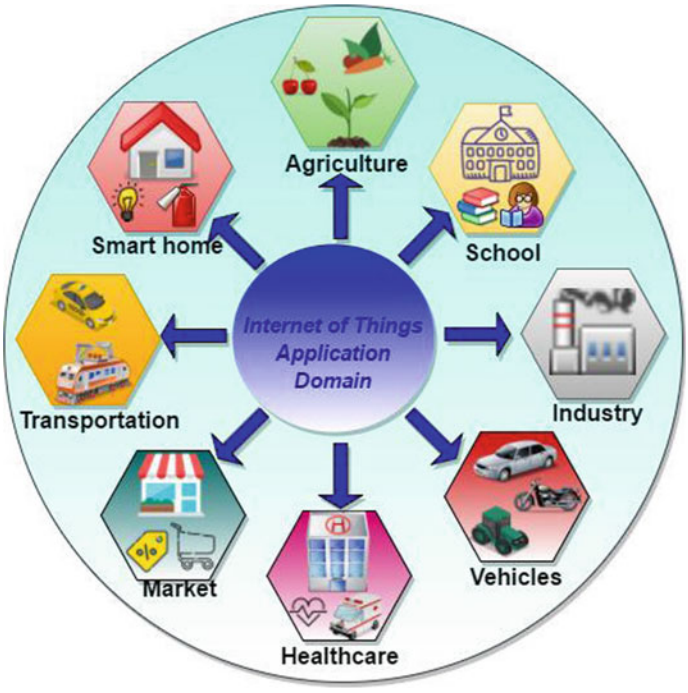

Fig. 1 IoT application domain with the potential market

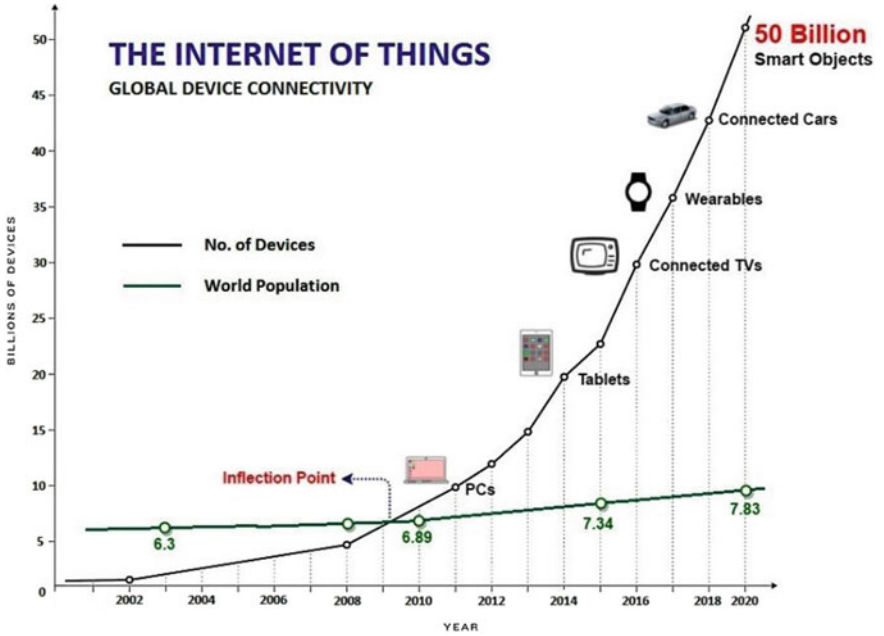

Fig. 2 Cisco IBSG IoT prediction

the population of the whole world although much of which is not connected to the network yet.

The remaining part of this paper is structured as follows. Section 2 describes the industrial opportunities for IoT device developers. Section 3 gives an in-depth survey of specific application layer protocols and their uses in the real world. Section 4 provides the comparative evaluation and analysis of these different application layer protocols. In the end, Sect. 5 concludes the paper with pointers to future work.

2 Industrial Opportunity

Billions of physical devices or objects connecting to the internet are attracting many big organizations to invest their trillions of money in IoT projects. According to the McKinsey Global Institute [4], in the last 5 years, the number of increase in connected devices reaches 3 times. This also gives a great market and industrial opportunities for device manufacturers, software developers, and service providers. IoT has the biggest economic impact on healthcare [5] and manufacturing applications [6]. Many medical applications including remote health monitoring services, diagnosis, treatment, fitness programs, etc. are predicted to create about 2.5 trillion dollars annual growth to the global economy by 2025, as shown in Fig. 3.

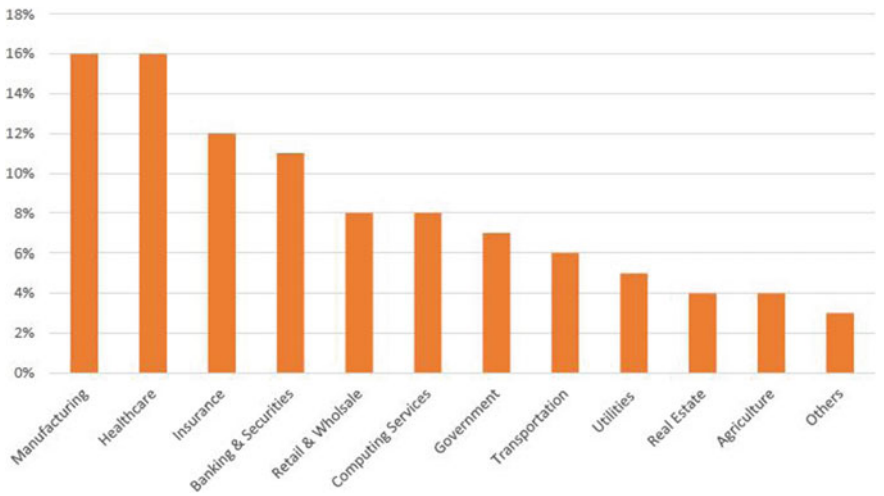

Fig. 3 IoT market

All these facts point to the fast growth and potentially powerful impact of the IoT in the coming years. By considering the above estimates, the challenge in IoT is not a single technology challenge but the entire area of computing. IoT redefines the thinking of the fundamentals of computation and communication paradigms, software, security, and privacy issues. The main central issue in IoT is how to make the full interoperable interconnected devices possible. To provide them with a higher level of smartness by enabling them to adapt to dynamic and autonomous behavior, guaranteeing trust, privacy, and security.

3 IoT Application Protocols

The application layer is the core layer of every communication system model. It is responsible for the effective communication services within the networks and their interconnection. There are many different application protocols that exist with different functionalities in different perspectives. IoT application layer protocols can be categorized into two categories based on the communication-based architecture model. These models are Publish/Subscribe model and Request/Response model. Some of the protocols existing today are IETF's CoAP [7], IBM's MQTT [8], Jabber's XMPP [9], OASIS's Standard AMQP [10], MQTT-SN [11], etc. Next, we will discuss these protocols in detail with their advantages and disadvantages.

3.1 CoAP (Constrained Application Protocol)

The Internet Engineering Task Force (IETF) Constrained RESTful Environments (CoRE) Working group has created a specialized web transfer protocol for constrained devices, known as CoAP [12], for M2M and IoT applications. CoAP is designed to implement a subset of REST architecture [13] with common HTTP for simplified integration with the web while offering specialized features for M2M applications such as built-in discovery, multicast support, low overhead, asynchronous message exchanges and simplicity for constrained environments. Unlike HTTP, CoAP uses UDP as a transport layer protocol making it more suitable for IoT applications.

CoAP is a request/response interaction model similar to client/server architecture and uses HTTP methods such as GET, POST, PUT and DELETE to request an action on a resource (using Uniform Resource Identifiers) on a server. CoAP uses REST-CoAP proxies to interconnect with HTTP which acts both as a server and a client for internetworking communication. Figure 4 shows the overall functionality and the mapping between HTTP and CoAP enabling traditional web clients to access CoAP servers transparently.

CoAP logically uses a two-layer approach, the request/response sub-layer, and the messaging sub-layer. The request/response sub-layer handles the REST communications using Method and Response Codes, and the messaging sub-layer deals with UDP transport layer and the asynchronous message interactions. Figure 5 shows the CoAP protocol stack generally use in the constrained environment.

CoAP defines four types of messages—Confirmable message, Non-Confirmable message, reset message, and Acknowledgment message. Some of these messages include Method and Response codes which allows them to carry requests or responses. Since the exchange of messages is asynchronous between two endpoints over UDP for communication, it marks a message as Confirmable (CON) to provide

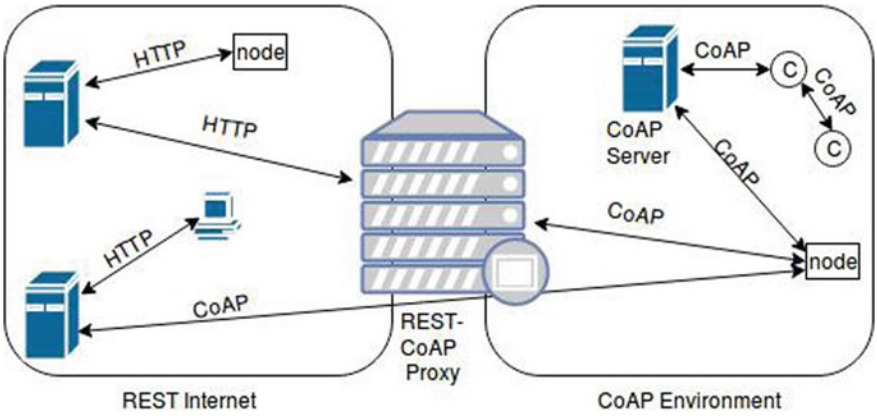

Fig. 4 CoAP architecture

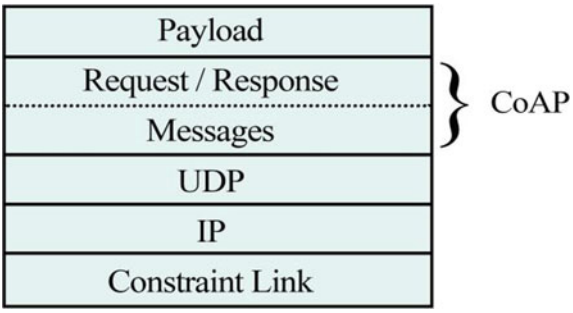

Fig. 5 CoAP protocol stack

reliability. Sender waits for an Acknowledge (ACK) message and retransmits the confirmable message with exponential back-off after the default timeout. A recipient may confirm the CON message by sending a corresponding ACK message or it may reject by sending Reset (RST) message if it is unable to process the confirmable message because of a lack of context information. A non-confirmable message (NON) is sent if the CoAP message does not require reliability. Duplicate Non-confirmable messages are detected by Message-ID which is always present in all header formats. CoAP messages are simple and use binary format to encode. Figure 6 [7] shows the header format of CoAP message. Each message contains a 4-byte fixed-size header followed by a Token value of variable length between 0 and 8 bytes long, again followed by an optional payload field in datagram packet.

CoAP is suitable for low constrained devices because of features like resource observation, Block-wise resource transport, resource discovery, interacting with HTTP and DTLS security [7, 12, 14]. CoAP client and server are implemented in many languages to be directly used in any programs. Some of the implementations are libcoap in C, aiocoap in python 3, californium in java, cantcoap in C/C++, copper in javascript, icoap in objective C, coaprs in rust [15, 16], etc.

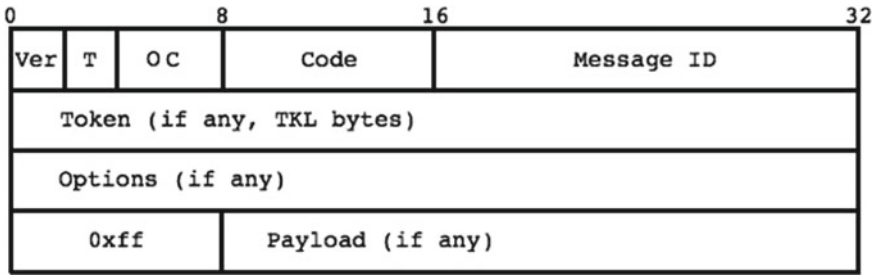

Fig. 6 CoAP header format

3.2 MQTT (Message Queue Telemetry Transport)

In 1999, IBM's Andy Stanford-Clark and Eurotech's Arlen Nipper collaboratively developed an asynchronous publish/subscribe "lightweight" messaging transport protocol for Machine-to-Machine communications in wireless networks. MQTT [8] became an OASIS standard in 2014 and recently approved as an ISO standard (ISO/IEC 20922) in January 2016. MQTT protocol is well suited for the IoT applications which require less bandwidth and low power consumption in unreliable networks.

The Publish/Subscribe messaging is an event-driven and requires a broker to push messages to clients. MQTT broker acts as a central point of communication and is responsible for dispatching all the messages between the senders and the interested recipients based on the topic contained within the message. A sender publishes a message with a certain topic to the broker and all the clients that are subscribed to that topic with the broker receive the message. Clients do not require to know each other rather all the communication takes place over the topic. Figure 7 clearly shows the architecture of MQTT using a publish/subscribe pattern providing simple and highly scalable solutions for the IoT applications without having dependencies between the Publisher and the Subscriber.

In contrast to HTTP where the client pulls the information from the server, MQTT broker pushes all the information to the interested clients. Therefore, each MQTT subscriber has to maintain a permanent open TCP connection to the broker. If the subscriber's connection is broken due to any reason, the MQTT broker buffers all the messages and deliver it later when the client is back online. To ensure reliability, MQTT uses three levels of QoS messages. QoS 0 (At most once delivery) means that client sends the packet only for one time and does wait for the ACK packet from the broker. If the data gets lost, there will be no retransmission of that packet. QoS 1 (At least once delivery) means that the user must ensure the delivery of packet for at least one time. After transmitting the PUBLISH message, client waits for the predefined amount of time to receive PUBACK packet or it retransmits the data. QoS 2 (At most

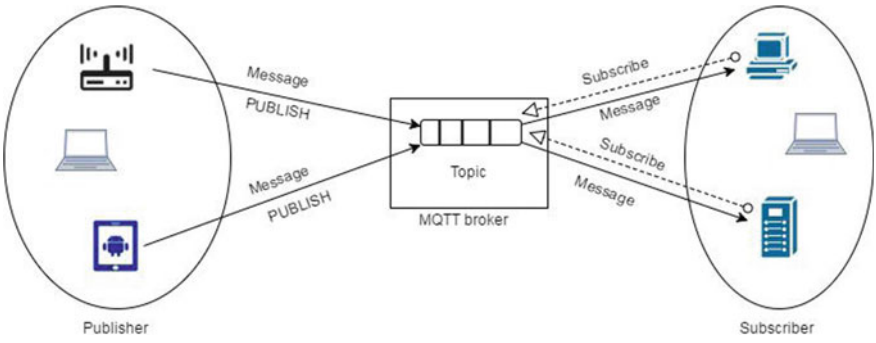

Fig. 7 MQTT architecture

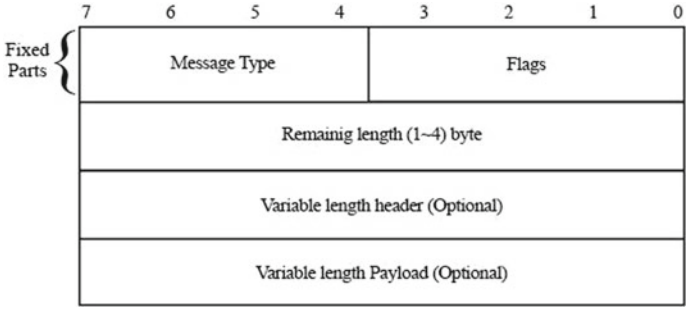

Fig. 8 MQTT header format

once delivery) means that packet must be delivered to the subscriber-only one time. The client sends the PUBLISH message and waits for the PUBREC message. After receiving PUBREC message, the Client sends the PUBREL message to the broker and discards the references to the published data.

MQTT protocol works efficiently by exchanging series of MQTT control packets in a systematic way. All the MQTT Control packet format contains three parts in the order illustrated in Fig. 8 [8].

The first part is the fixed header of 2 bytes which is present in all packets. The first byte of a fixed header tells the MQTT Control packet type and their specific flags. The remaining length field tells the total number of bytes present within the current packet including data in the variable header and the payload. The next field is the variable header which is present in some of the MQTT control packet types. The last field is the payload which is also optional.

MQTT protocol is an ideal messaging protocol for the various applications in IoT and M2M communications such as Health care, Monitoring, smart electricity metering, etc. and used by many companies for their communications. Some of the real-world applications are IBM mobile messaging [17], Facebook messenger, EVRY-THNG IoT platform, Amazon Web Services [18], AdafruitIO cloud service [19], etc. and many more.

3.3 MQTT-SN (MQTT for Sensor Networks)

MQTT-SN is a publish/subscribe protocol [11] specially designed for the wireless sensor networks having the main focus on the constrained devices. It is a variant of the MQTT protocol adapted to the nature of the wireless communication environment. MQTT-SN does not require TCP/IP stack for its operations therefore it is supported by more protocols like Zigbee, Z-Wave, and so on.

MQTT-SN uses 2-byte Topic ID instead of Topic name in the header which makes it lighter than MQTT and ideal for implementation on low-cost, battery-operated devices. It uses MQTT-SN gateway and forwarder to translate packets into

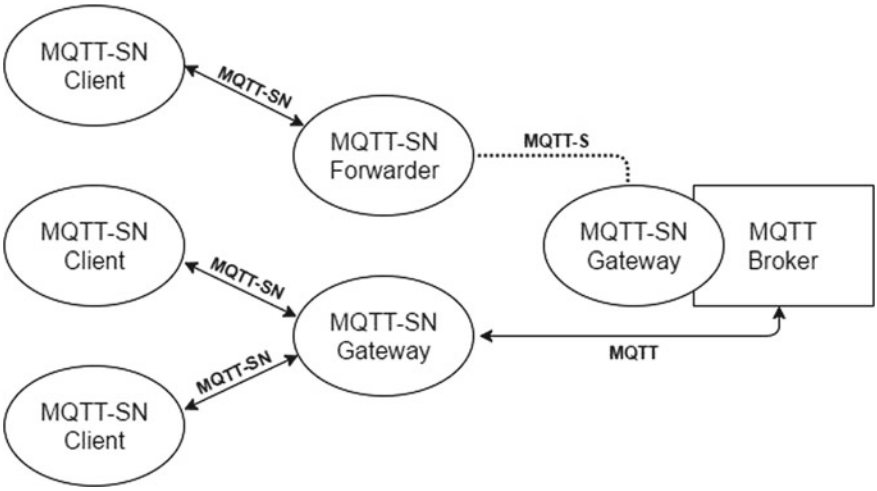

Fig. 9 MQTT-SN architecture

the standard MQTT format [20]. The Gateway then connects to the broker using TCP connection (Fig. 9).

3.4 AMQP (Advance Message Queuing Protocol)

AMQP is an open standard protocol [10] built by John O'Hara at JPMorgan Chase in the UK. The protocol efficiently supports a wide range of messaging applications and communication patterns. AMQP is a message-oriented middleware protocol including features like message orientation, routing, queuing, security, and reliability.

Like MQTT, AMQP is an asynchronous Publish/Subscribe messaging [21] which requires a third-party broker to deliver messages to the interested clients. AMQP broker uses two key components for communication: Exchanges and Queues, as shown in Fig. 10.

Exchange routes the messages to their appropriate queue. Message queues are used to store the messages and then send them to the subscribers. AMQP exchanges ensure the interoperability between different clients.

AMQP is a binary, wire-level protocol providing reliable communication with message-delivery guarantees using QoS such as at least once, at-most once, and exactly once delivery and uses TCP as reliable transport layer protocol. AMQP defines the messaging scheme as bare message and annotations may be added by the intermediaries before or after the bare message during transmit to ensure end-to-end encryption and integrity check as shown in Fig. 11.

AMQP is widely used in business applications and financial trading, RabbitMQ is an open-source broker for AMQP. Some of the real-world applications of AMQP

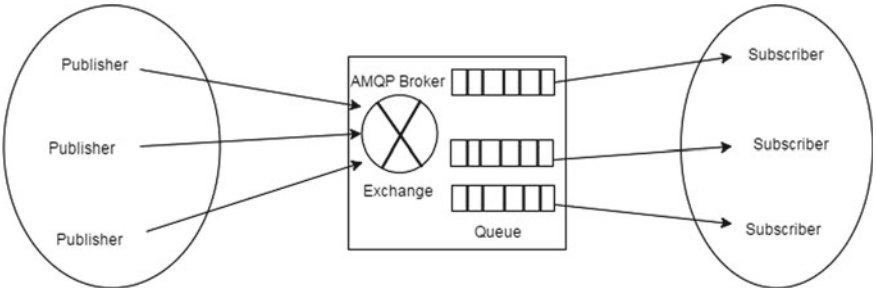

Fig. 10 AMQP architecture

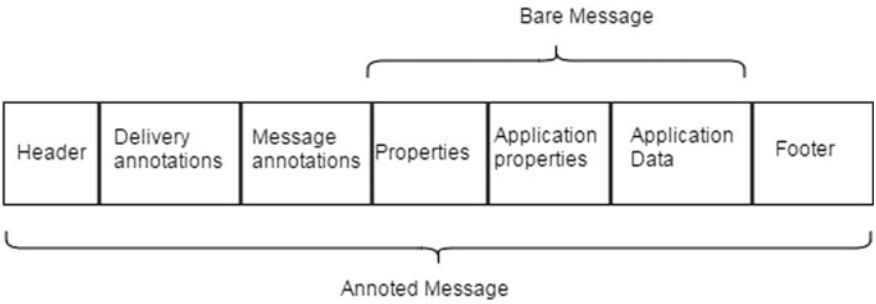

Fig. 11 AMQP message format

are Apache Qpid, IIT Software SwiftMQ messaging product, INETCO's AMQP protocol analyzer, JORAM, Kaazing's AMQP web Client, Microsoft's Windows Azure Service Bus, StormMQ, MQlight, etc. [22].

3.5 XMPP (Extensible Messaging and Presence Protocol)

XMPP is an application layer communication protocol based on XML for message-oriented Middleware and is mainly used for Instant Messaging, group chat, gaming, VoIP, video calling, file transfer, presence information, and social networking services. In 1999, Jeremie Miller of Jabber Technology developed an open-source software "jabberd server" which led to the formation of XMPP protocol. IETF standardized XMPP protocol in 2004 and used as an IETF instant messaging and presence technology.

XMPP uses decentralized Client-Server architecture, i.e., anyone can create his or her own XMPP server and can communicate with the rest of the network using DNS. A unique XMPP ID, also called JID (Jabber ID), is given to every user of XMPP in a network from their server. This user ID looks similar to an email address containing username and Domain name such as user@myexample.com. XMPP is

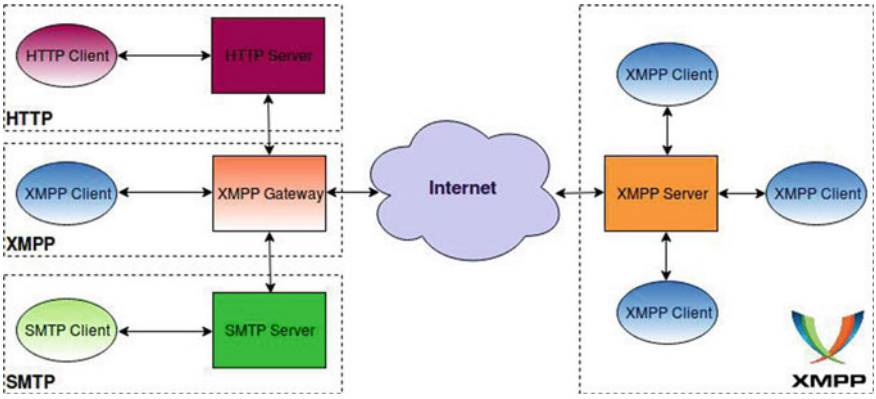

Fig. 12 XMPP communication

secure, scalable, extensible, flexible, standard which provides in-built support for authentication and encryption (Fig. 12).

The core technology of XMPP is streaming XML stanzas over the network. As the XMPP runs TCP on a transport layer, it maintains an always open TCP connection between client and server which exchanges XML snippets over the stream. XML snippets or stanzas are of three types: message¡, presence¡ and iq¡.

4 Evaluation

There are many different application layer protocols exist which can be used in different IoT scenarios. Choosing the best protocol among the pool of protocols is a hard task and depends on the requirement of the applications in which protocol has to be used. No real-time evaluation has been done till now which can compare all the existing protocols but pairwise comparisons and evaluation have been done in many surveys. Since each protocol can perform widely different in different scenarios and platforms. In a constraint environment with low power and computation, REST-based CoAP can perform efficiently while in the case where devices run on battery and require only exchange of messages MQTT is considered a good choice, and for the business applications where devices are not low constraint AMQP works more efficient. So it is not at all justifiable to give a single prescription of one protocol for all the IoT applications but a comparison based on common parameters can be given. Figure 13 shows the most common parameters for all IoT application layer protocols.

Application Protocol	CoAP	MQTT	MQTT-SN	XMPP	AMQP
REST Architecture	Yes	No	No	No	No
Publish/Subscribe	No	Yes	Yes	Yes	Yes
Request/Response	Yes	No	No	Yes	No
Transport	UDP	TCP	TCP	TCP	TCP
Security	DTLS	TLS/SSL	TLS/SSL	TLS/SSL	TLS/SSL
Header Size (Byte)	4	2	2	-	8
QoS	Yes	Yes	Yes	No	Yes

Fig. 13 Comparison between different IoT application protocols

5 Conclusion and Future Work

In this paper, we have given a brief introduction about the IoT and its applications and also the great impact of IoT in the global economy. Then we have focused on the different application layer protocols which are used most commonly nowadays. We have addressed the strengths and weaknesses of each protocol with its practical implementation in industries. In the future, we are aiming at implementing these protocols in a simulation testbed which can give an accurate comparison.

References

1. Atzori L, Iera A, Morabito G (2010) The internet of things: a survey. Comput Netw 54(15):2787–2805
2. Al-Fuqaha A, Guizani M, Mohammadi M, Aledhari M, Ayyash M (2015) Internet of things: a survey on enabling technologies, protocols, and applications. IEEE Commun Surv Tutorials 17(4):2347–2376
3. Evans D (2011) The internet of things: how the next evolution of the internet is changing everything. CISCO White Paper 1(2011):1–11
4. Manyika J, Chui M, Bughin J (2013) Disruptive technologies: advances that will transform life, business, and the global economy. McKinsey Global Institute, www.mckinsey.com/mgi

5. Islam SMR, Kwak D, Kabir MH, Hossain M, Kwak KS (2015) The Internet of things for health care: a comprehensive survey. IEEE Access 3:678–708
6. Bi Z, Xu LD, Wang C (2014) Internet of things for enterprise systems of modern manufacturing. IEEE Trans Industr Inf 10(2):1537–1546
7. Bormann C, Hartke K, Shelby Z (2014) The constrained application protocol (CoAP). RFC 7252
8. Banks A, Gupta R (2014) MQTT Version 3.1.1, OASIS Standard. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html
9. Saint-Andre P (2011) Extensible messaging and presence protocol (XMPP): core. RFC 6120
10. OASIS Advanced Message Queuing Protocol (AMQP) (2012) Version 1.0, OASIS Standard. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf
11. Stanford-Clark A, Truong HL (2013) MQTT for sensor networks (MQTT-SN) protocol specification version 1.2. http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_B ?42_E 1 2 ?3 5
12. Bormann C, Castellani AP, Shelby Z (2012) Coap: an application protocol for billions of tiny internet nodes. IEEE Internet Comput 16(2):62–67
13. Fielding RT (2000) Architectural styles and the design of network-based software architectures. AAI9980887
14. Lerche C, Hartke K, Kovatsch M (2012) Industry adoption of the internet of things: a constrained application protocol survey. In: Proceedings of 2012 IEEE 17th international conference on emerging technologies factory automation (ETFA 2012), pp 1–6, Sept 2012
15. Constrained Application Protocol—Wikipedia, The Free Encyclopedia, 2017. (Online; accessed 1 May 2017)
16. Bormann C (2016) Constrained application protocol implementations. http://coap.technology/impls.html
17. Lampkin V, Leong WT, Olivera L, Rawat S Subrahmanyam N, Xiang R (2012) Building smarter planet solutions with MQTT and IBM WebSphere MQ telemetry. IBM.com/redbooks, Sept 2012. ibm.com/redbooks
18. Amazon Web Services (2017) Message broker for AWS IoT. http://docs.aws.amazon.com/iot/latest/developerguide/protocols.html
19. Cooper J (2016) Adafruit products and adafruit IO. https://learn.adafruit.com/adafruit-io/mqtt-api
20. Govindan K, Azad AP (2015) End-to-end service assurance in iot mqtt-sn. In: 2015 12th Annual IEEE consumer communications and networking conference (CCNC), pp 290–296
21. Eugster PT, Guerraoui T, Sventek J (2000) Distributed asynchronous collections: abstractions for publish/subscribe interaction, pp 252–276. Springer Berlin Heidelberg, Berlin, Heidelberg
22. OASIS (2017) Advanced message queuing protocol: products and success stories. http://www.amqp.org/product/realworld. (Online; accessed 1 May 2017)