IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 21, 2020, accepted October 31, 2020, date of publication November 4, 2020, date of current version November 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3035849

The Use of MQTT in M2M and IoT Systems: A Survey

BISWAJEEBAN MISHRA AND ATTILA KERTESZ
Department of Software Engineering, University of Szeged, 6720 Szeged, Hungary
Corresponding author: Biswajeeban Mishra (mishra@inf.u-szeged.hu)

ABSTRACT Nowadays billions of smart devices or things are present in Internet of Things (IoT) environments, such as homes, hospitals, factories, and vehicles, all around the world. As a result, the number of interconnected devices is continuously and rapidly growing. These devices communicate with each other and with other services using various communication protocols for the transportation of sensor or event data. These protocols enable applications to collect, store, process, describe, and analyze data to solve a variety of problems. IoT also aims to provide secure, bi-directional communication between interconnected devices, such as sensors, actuators, microcontrollers or smart appliances, and corresponding cloud services. In this paper we analyze the growth of M2M protocol research (MQTT, AMQP, and CoAP) over the past 20 years, and show how the growth in MQTT research stands out from the rest. We also gather relevant application areas of MQTT, as the most widespread M2M/IoT protocol, by performing a detailed literature search in major digital research archives. Our quantitative evaluation presents some of the important MQTT-related studies published in the past five years, which we compare to discuss the main features, advantages, and limitations of the MQTT protocol. We also propose a taxonomy to compare the properties and features of various MQTT implementations, i.e. brokers and libraries currently available in the public domain to help researchers and end-users to efficiently choose a broker or client library based on their requirements. Finally, we discuss the relevant findings of our comparison and highlight open issues that need further research and attention.

INDEX TERMS IoT, IoT protocols, MQTT, MQTT brokers, survey.

I. INTRODUCTION

The Internet of Things (IoT) connects everyday devices like a fridge, oven, vehicle, washing machine, fitness band, watch, and even shoes to the internet [1]. It enables us to collect monitored data from these devices over a network without any human-to-human or human-to-computer interaction that can be used to improve our life, business or environments [3]. For example, a simple IoT application like an activity or fitness tracker can inform many things about us, such as distance we walked, ran, cycled or swam, pace, pulse rate, (swim) stroke count, calories we burned, sleep quality to help us improve our regimen. Efficient IoT solutions can help us to control these devices remotely from our phones or tablets. No matter it is agriculture, transport, sports, health, military, energy, or entertainment; today the application space of IoT is virtually endless. It is rare to find any industrial area that does not get benefits from this IoT revolution. Social networking and smart city applications can also benefit from this trend [4]. Instead of waiting for monthly or yearly reports, business and industry can get accurate consumer-data in real-time. They can analyze the data to make more informed decisions, which can add value to their business [5].

Cloud Computing [6] provides on-demand network access to a shared pool of configurable computing resources (such as networks, servers, storage, applications and services). IoT systems generate massive amounts of data, and Cloud Computing paves the way for that data to reach its destination. Clouds and IoT have a complementary relationship, and they can increase efficiency of our everyday tasks. There are different communication protocols for the transmission of data in IoT and M2M systems. MQTT was introduced by OASIS in 2013 for the abbreviation of Message Queuing Telemetry Transport protocol [7]. It became standardized with the release of version 3.1.1 in 2014, when the standardization group omitted the abbreviation for MQTT, representing an OASIS M2M communications protocol. Since then it is being referred simply as MQTT, and it is not considered an acronym

The associate editor coordinating the review of this manuscript and approving it for publication was Xingwang Li.

any more. This open standard is now a widely used communication protocol across a variety of industries [8].

The aim of this article is provide answers for the following problem statements:

1) Does the use of the MQTT protocol stand out in M2M and IoT systems?,
2) Is there a growth in MQTT-related research works since its introduction in 1999?, and
3) What kind of MQTT broker implementations are currently available for public use, and how do they differ from each other?

The novelty of this study is that it performs a quantitative evaluation of MQTT-related published works found in Google Scholar, Dimensions, and Scopus; highlights the potential and limitations of use of the MQTT protocol, identifies the annual exponential growth rate in MQTT research works since its appearance in 1999, and gathers the application areas of the MQTT protocol across various domains within IoT.

The evolution of the MQTT protocol, and specially the MQTT brokers using it, reflects the new trends of M2M and IoT systems, hence they also need to react to these changes. To reveal this evolution, this work conducts an in-depth comparative study of features of several MQTT broker implementations available in the public domain in various taxonomy categories to empower researchers or end-users to select an MQTT broker implementation based on their needs.

The remainder of this paper is structured and organized as follows: Section II introduces IoT reference model, Section III highlights messaging protocols for IoT communication, basics of publish/subscribe type messaging service and fundamentals of MQTT, Section IV discusses some important MQTT works published over past five years and presents a literature review. Section V defines the scope and method of the literature search conducted in this study and highlights its findings. Section VI introduces various MQTT implementations and compares their features through various taxonomy categories, and Section VII summarizes the revealed open issues and challenges around MQTT and, finally Section VIII concludes this work.

II. IoT REFERENCE MODEL

In an IoT system, things may refer to any sensor or devise, starting from very ordinary tiny things to large living things. Sensors in things work as input sources to produce or collect data, microcontrollers process those data, transmit over internet and actuators work as an output device to control or move something as per the instructions received from the Internet. The connections between people and things are increasing day by day, and the amount of data produced every second is stupendous. The number of smart devices has already outnumbered the number of humans on the planet. Numerous research works suggest that the world is embracing digital infrastructure five times faster than it embraced electricity and telephony. Industry estimates this pace in the number of smart devices to reach 74.44 billion by 2025 [9].

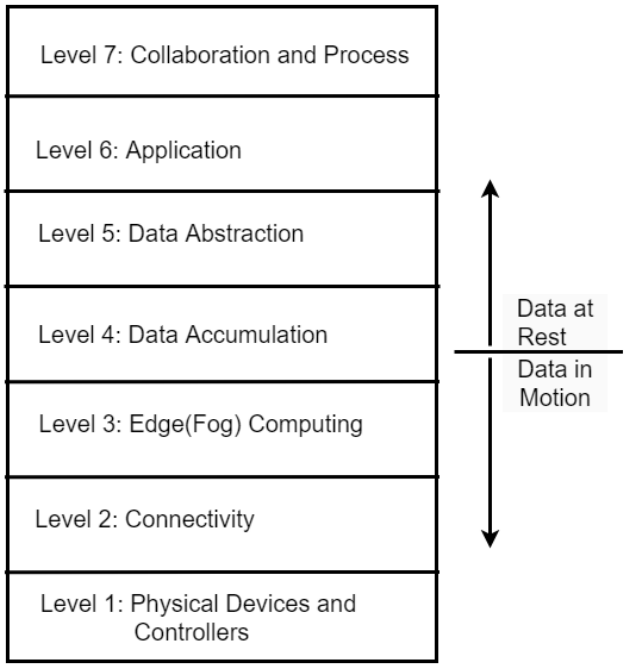

FIGURE 1. The seven layers of the IoT Reference Model [10].

Data generated by different kinds of devices can be processed in different ways and transmitted to geographically dispersed locations to be acted upon by applications. In 2014, keeping the view of creating a standardized global frame of reference, Cisco proposed a seven-layer IoT model [10] – see Figure 1. Next, we introduce this architecture.

A. LEVEL 1: PHYSICAL DEVICES AND CONTROLLERS

This layer comprises of physical devices and device controllers, which we call as things in the IoT. Being independent of shape, size, location, or origin, these things are diverse. A thing can be as small as the size of a silicon chip or as large as the size of a vehicle. These things or devices are endpoint devices that are capable of generating data, sending and receiving information, analog to digital conversion: as per the control commands they receive over the internet sent by the physical devices and controllers [2].

B. LEVEL 2: CONNECTIVITY

Traditional data communication networks incorporate several functions based on (ISO) 7-layer reference model, but even so, an IoT system contains many levels in addition to the data communications network. As the sole objective of the IoT reference model is to carry out M2M communications through the existing networks; it eliminates the need for the creation of a different network. It's designing principles make it work seamlessly on existing networks. Connectivity and reliable timely information transmission are two important roles of level 2. Connectivity includes communicating with and between the level 1 devices, reliable delivery across the networks, implementation of various protocols, switching

and routing, translation between protocols, security at the network level, and (self-earning) networking analytics [2].

C. LEVEL 3: EDGE (FOG) COMPUTING

Level 3 revolves around converting network data flows into suitable information for storage and further processing at level 4. It focuses on high volume data analysis and transformation. The fundamental principle of the IoT reference model is that intelligent systems initiate data processing as early and as close to the edge of the network as possible, instead of relying on the cloud to do all the work. It is often termed as Edge or Fog Computing. As data generated by things are usually sent to the connectivity level (level 2) networking devices, level 3 involves limited, session-less-transaction-less processing on a packet-by-packet basis. Level 3 data element functions include: data filtering, clean up, aggregation, packet content inspection, a combination of network and data level analytics, thresholding. Event generation and processing functions include: evaluating data for its norm be processed at a higher level, reformatting data for consistent higher level processing, expanding or decoding meaning adding additional information such as origin of data, distillation/reduction that refers to minimizing the impact of data and traffic on the network and higher-level processing systems, and assessment that involves evaluating threshold or alert, which process may redirect data to additional destinations [2].

D. LEVEL 4: DATA ACCUMULATION OR STORAGE

This level implements mechanisms to make network data usable by applications. It involves converting data in motion to data-at-rest, formatting network packets to relational database tables, enabling transition of event based computing to query based computing, and reducing data through filtering and selective storing. In short, level 4 captures and stores data to be used by non-real-time-applications when necessary. It bridges the gap between real-time networking world and the non-real-time application world by working as a boundary between event-based data generation and upper level query based data use [2].

E. LEVEL 5: DATA ABSTRACTION

IoT systems require to scale to the need, may it be to a corporate level or global level. To enable scaling on the large amount of data received from IoT and non-IoT systems several storage systems are required. This in turn gives birth to the necessity of information integration from various data sources. Hence data abstraction level moves around many activities. It reconciles the conflict between multiple data formats from different sources, and it assures consistent semantics of data across all sources. It also ensures that data is complete to be used by any higher-level application, and it integrates data into one place or giving access to multiple data stores using data virtualization. Furthermore, it facilitates data protection by suitable authentication and authorization and normalizing or denormalizing, and use indexing to provide quick application access [2].

F. LEVEL 6: APPLICATION

Level 6 applications interact with Level 5 and data at rest. So this level does not need to operate at network speeds. Some of the important functionalities of this level are monitoring device data. controlling devices, combining device and non-device data, interpreting information, and reporting [2].

G. LEVEL 7: COLLABORATION AND PROCESSES

This level works beyond the technical model. It includes humans and business processes. All the information created by several IoT systems is of very little value unless it produces insightful analytics. Making an insightful decision and taking appropriate action often involve people and processes. People use different applications and methods to derive useful information out of IoT data. Typically it is not a one-man job. People need to communicate and collaborate to make the most of IoT data to make the right business decisions at the right time. Hence, Level 7 represents a higher level involving humans than a single application [2], see Figure 2.

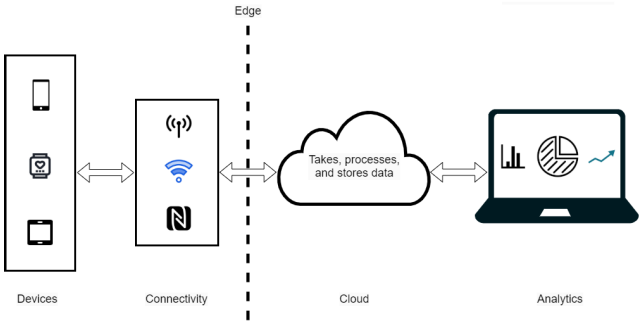

FIGURE 2. A general IoT architecture.

III. MESSAGING PROTOCOLS FOR IoT

The previous section introduced the reference model of IoT systems, and reflected the need for a reliable communication protocol, for the implementation of it. In this section, we discuss IoT protocols in general, define the basics of a publish/subscribe (pub/sub) service, and introduce the MQTT protocol in detail.

IoT networks deploy several radio technologies like RFID, WLAN (IEEE 802.11), WPAN (IEEE 802.15) and WMAN (IEEE 802.16), etc. for communications at the lower level. Lower level communication protocols may include LoRaWAN, SigFox in long-range (km)-low data rate (bps-kbps), Cellular/4G/5G in long-range(km)-high data rate (Mbps), Zigbee, Zwave in medium range (m)-medium-data rate (kbps), WiFi in medium range(m)-high-data rate (Mbps), and NFC short-range (cm)-medium data rate (kbps) category. No matter which radio technology is used to deploy an IoT network, all independent data generating end devices must make their data available to the internet for further

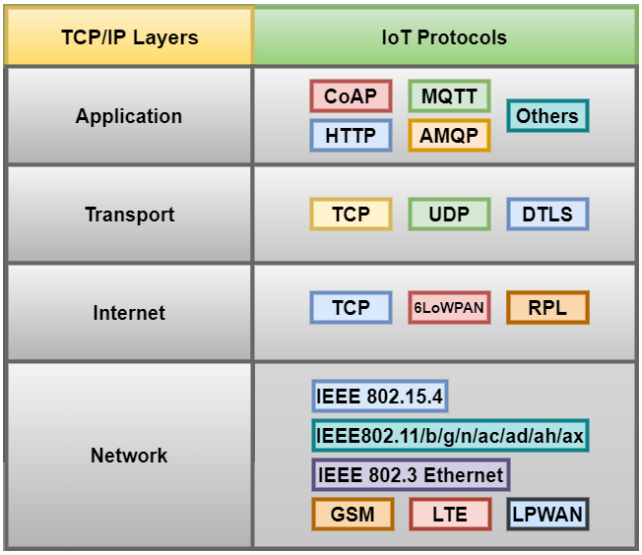

FIGURE 3. TCP/IP Model-Main IoT Protocols.

processing and send control information back [11], [12]. The performance of M2M communication heavily relies on the special messaging protocols designed for M2M communication within IoT applications [10]. The web uses a single standard messaging protocol HTTP; on the other hand, being too diverse in its characteristics IoT cannot rely on a ''one-protocol-fits-all'' philosophy. Therefore nowadays there are many messaging protocols available to select from for various needs of IoT systems. MQTT, CoAP, AMQP, and HTTP are the four widely accepted and emerging messaging protocols for IoT systems [12]. Figure 3 shows the TCP/IP protocol stack for IoT systems.

A. BASICS OF PUBLISH/SUBSCRIBE SERVICE

In a pub/sub type messaging service, ''message'' refers to the data that moves through the service, ''topic'' is a named entity that denotes a feed of messages [13], and ''subscription'' refers to an interest in receiving messages on a particular topic. The ''publisher or producer'' refers to a device or a program that creates messages and publishes them to the messaging service on a set topic, and ''subscriber or consumer'' refers to a device or program that receives messages on a specified subscription [12], [14].

B. FUNDAMENTALS OF MQTT

MQTT is an open OASIS and ISO standard (ISO/IEC PRF 20922) for client-server, publish/subscribe type messaging transport protocol [8]. It was invented by Dr. Andy Stanford-Clark of IBM, and Arlen Nipper of Arcom, in 1999. The design principles of this protocol focus on minimizing network bandwidth and device resource requirements ensuring reliable delivery. It is capable of transmitting data over low-bandwidth or unreliable networks with very low consumption of power [8], [12]. Characteristics like lightweight, open, simple, and easy deployment make MQTT an ideal communication protocol for constraint environments. The protocol runs over TCP/IP, or over other network protocols that provide ordered, lossless, bi-directional connections. It uses IANA registered port number 8883 for SSL/TSL connections, and 1883 port number for Non-TLS connections [8].

MQTT offers three qualities of service for message delivery. The first quality of service is QoS 0. It is otherwise known as ''at most once'' message delivery service. In this quality of service, messages are delivered at most once according to the operating environment. The chance of message loss remains intact. An example of the use case of QoS 0 is sending real-time pressure, humidity, and temperature sensor data to a remote reading application where it does not matter if the connection to the application reading sensor data is lost for a while. The second quality of service is QoS 1. It is otherwise known as ''at least once'' message delivery service. In this service, messages are assured to be delivered at least once, duplicity may occur. The third quality of service is QoS 2. It is otherwise referred to as ''exactly once'' message delivery service. In this service, messages are ensured to be delivered exactly once. In QoS 2, a small transport overhead is observed and protocol exchanges are minimized to reduce network traffic. This service is useful to notify interested parties when an unusual disconnection occurs. This level could be used with billing systems where redundancy of loss of messages may lead to incorrect charges being imposed [8].

MQTT has three constituent components:

1) Publisher or Producer (An MQTT Client)
2) A broker (An MQTT Server)
3) Consumer/ Subscriber (An MQTT Client)

An MQTT client is a program or device that uses the MQTT protocol. A client is responsible for opening network connection to the server, creating messages to be published, publishing application messages to the server, subscribing to request application messages that it is interested in receiving, unsubscribing to remove a request for application messages and closing network connection to the server. Application message refers to the data carried by the MQTT protocol across the network for the application. All application message transported by MQTT contains payload data, a QoS, a collection of properties, and a topic name [13], [15]. An MQTT server is a program or device based on MQTT that acts as a post office between publishers and subscribers. An MQTT broker is responsible for accepting network connections from clients, accepting application messages published by clients, processing subscribing and unsubscribing requests from clients, sending application messages to clients as per its subscriptions, and closing the network connection from the client. MQTT is a bi-directional communication protocol. This helps in both sharing data, managing, and controlling devices. It normally requires a fixed header of 2-bytes with small message payloads up to the maximum size of 256 MB [8].

TABLE 1. Some important works in MQTT.

Main topic	Reference to related study in MQTT	Publication year
MQTT advantage	Prada, Miguel A., et al. [16]	2016
	Wagle, Satyavrat [17]	2016
	Xu, Yiming, V. Mahendran, and Sridhar [18]	2016
Comparison with Other IoTprotocols	Naik, Nitin [11]	2017
	Yokotani, Tetsuya, and Yuya Sasaki [19]	2016
	Luzuriaga, Jorge E., et a [20]	2015
Limitations of MQTT	Dizdarević, Jasenka and Carpio, et al. [21]	2019
	Dan and Cheng, Xiaochun. [22]	2019
Securing MQTT communications	Niruntasukrat, Aimaschana, et al. [23]	2016
	Singh, Meena, et al. [24]	2015
	Lesjak, Christian, et al [26]	2015

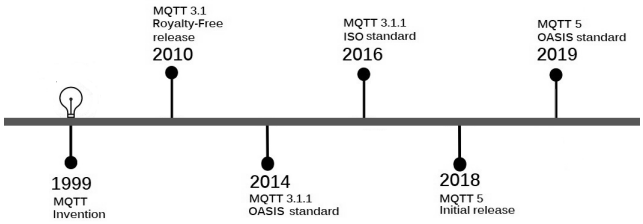

FIGURE 4. Timeline of MQTT development.

C. VERSION HISTORY OF MQTT

Since its release in 1999, MQTT has been riding high on its developmental path: see Figure 4. Its first royalty-free release occurred in 2010. In 2013, it was submitted as a standard to the OASIS standard group. The first published release was version 3.1.1. It became an OASIS standard in 2014, and in 2016 it became an ISO standard. The most recent MQTT version is MQTT v5 that was published in 2018, and got standardized in 2019. Version 5 incorporates significant feature updates to meet the requirements of modern IoT-Cloud systems. It offers better error handling for mission-critical implementation, better scalability for native cloud computing, greater flexibility, and easier integration into an existing computing infrastructure [8].

IV. LITERATURE STUDY

In this section we summarize some of the important MQTT-related studies published in the literature in the last five years. Table 1 represents an overview of important related works in MQTT. We have categorized these studies into four main topics, which we discuss next:

1) MQTT advantage,
2) Comparison of MQTT with other IoT protocols,
3) Limitations of MQTT,
4) Approaches to secure MQTT communications.

A. MQTT ADVANTAGE

Prada *et al.* [16] discussed the lightweightness of MQTT protocol for communication with resource-constrained devices. To evaluate the ability of MQTT to work with low-end devices, authors have developed a module for an EjsS (Easy Java/Javascript Simulations) based educational tool to communicate with an Arduino based device. The module developed by the authors empowers the educational tool to use MQTT protocol to communicate with the physical device powered by an Arduino microcontroller. Their experimental results show that the MQTT protocol can be successfully used to communicate between an interactive educational tool running on a web browser and a resource-limited hardware platform without adding any time or complexity burden to the educators who apply it.

Wagle [17] presented the viability of MQTT protocol for IoT centric wireless sensor networks interfacing with the Internet and implementing machine learning algorithms over the cloud. The paper implements an IoT Application involving ubiquitous sensing, M2M communication, cloud computing, and semantic data extraction and outlines the advantages, disadvantages, and suitability of MQTT towards IoT applications. The findings of the authors reinstate the retention of messages and other features of MQTT to make it fit for semantic data extraction and easy integration of new devices. The availability of topics for subscription and publishing renders data routing procedures largely redundant and eliminates the requirement for heavy mechanisms for directing data to specific buffers at the program level. QoS and Last Will and Testament enhances the reliability of MQTT and makes it suitable in constraint-bound situations.

The work by Yiming *et al.* [18] proposed an SDN-based fog computing architecture and developed its working prototype. They have implemented broker functionalities integrated at the edge-switches, that acts as a reliable message delivery platform and performs message-based analytics at the switches. They found that the fog node delivers at significantly higher throughput, as compared with the respective traditional client and end-host setup.

B. COMPARISON OF MQTT WITH OTHER IoT PROTOCOLS

Naik [11] provided an in-detail evaluation of the four widely used messaging protocols MQTT, CoAP, AMQP, and HTTP

for IoT systems. This work presents an overall comparison of the characteristics of these protocols. Furthermore, this work gives insight into the strengths and limitations of these IoT protocols by conducting an in-depth and relative analysis based on some interrelated criteria. Based on the static components and some empirical evidence from the literature the author is able to demonstrate a bigger and comparative picture of messaging protocols that give the end-user a clear understanding to select the suitable protocol for their requirements.

Yokotani and Sasaki [19] compared the performance of MQTT in the category of protocols based on ICN architecture and HTTP in the category of legacy protocols also proposes enhancements to MQTT for better performance. Their experiment confirms MQTT performs better than HTTP and concludes protocols based on ICN architecture are better suited for IoT systems.

Luzuriaga *et al.* [20] presented an experimental evaluation of AMQP and MQTT protocols over unstable and mobile networks in terms of message loss, latency, jitter, and saturation boundary values. The findings of this work are that during message bursts, the delivery follows a LIFO (last-input first-output) order, in the case of AMQP, but in MQTT packet delivery retains its order. AMQP is more security-oriented than MQTT and MQTT is more energy-efficient than AMQP. The authors suggest the use of AMQP protocol to build reliable, scalable, and clustering messaging platforms over an ideal WLAN, and the use of MQTT protocol to connect edge Nodes over-constrained environments.

C. LIMITATIONS OF MQTT

Dizdarevic and Cheng [21] showed a lightweight designing principle of MQTT to eliminate the encryption of data being transmitted. The encryption implemented as a separate feature via TLS by several MQTT brokers, in turn, increases overhead. The default plain-text data exchange by MQTT poses a big threat from a security point of view.

Dan and Cheng [22] analyzed the rising security attacks on smart devices that are capable of communicating over a network using several IoT protocols. The authors shed light on security issues surrounding MQTT, and other protocols and show how every MQTT based broker implementation has no equal abilities for entity authentication or encryption. They review some of the existing security methods used to secure a communication channel and finally demonstrate a Novel approach named 'Value-to-Keyed-Hash Message Authentication Code (Value-to-HMAC) that achieves better performance than traditional symmetric-key encryption algorithm without compromising the integrity of the information being transmitted.

D. VARIOUS APPROACHES TO SECURE MQTT COMMUNICATIONS

Niruntasukrat [23] introduced an authorization mechanism for MQTT-based IoT systems. The authorization mechanism presented in this work is based on OAuth 1.0a, an open authorization standard for web applications. Considering many aspects such as limited node resources, lack of user interface, and key/secret distribution and management, the authors have done careful modification in the OAuth 1.0a mechanism to make it suitable for the MQTT deployment. The proposed authorization mechanism requires two sets of credentials for a device to connect to the MQTT broker. Another set of credentials including Device ID and Device Secret is sent to the user through his phone or computer which can afford HTTPS. Later, it gets embedded offline into the device's memory. Authors have tested the design on a real MQTT based IoT service platform and demonstrate that it works with minimal authorization delay and message overhead without affecting user experience.

Singh *et al.* [24] experimented with the feasibility of CP/KP-ABE to enable communication security for IoT devices based on Pub-Sub architecture. They proposed a secure version of MQTT and MQTT-SN protocols with a new secure publish command called "SPublish", which publishes encrypted data based on CP/KP-ABE scheme using lightweight ECC techniques. Furthermore, they demonstrated their feasibility for various IoT requirements through simulations. The authors proved that the SMQTT-based CP/KP-ABE scheme performs better than the mechanism proposed by Wang *et al.* [25].

Lesjak [26] proposed a security design that uses the Transport Layer Security (TLS), which adds a secured communication layer beneath the MQTT protocol. Their proposed architecture incorporates a hardware security controller, that performs the TLS client authentication. With due relevant experiments, they show no significant performance overhead is imposed by the hardware security element and confirms the robustness of the proposed security mechanism. This work widens the realization of secure and privacy orientated futuristic smart service infrastructures.

V. ARTICLE SEARCH RESULTS AND STATISTICAL INFORMATION

This section focuses on the method and results surrounding our literature search. The primary objective of this study was to find MQTT-related research works published since its appearance in 1999, for identifying application areas of MQTT, and observing how MQTT stands out in the M2M protocol research race. To achieve this, relevant articles were searched in digital databases like Google Scholar, Dimensions, and Scopus. These digital databases provide efficient ways to search for various linked scholarly articles, books, abstracts, etc. Our literature search was focused on two categories:

1) Finding research works in various M2M Communication Protocols. Keywords used: "MQTT", "CoAP", "MQTT".
2) Finding MQTT Application Areas. Keywords used: "MQTT in Healthcare", "MQTT in Agriculture", "MQTT in Logistics", "MQTT in Disaster Management", and "MQTT in Smart city Services".

TABLE 2. Initial release years of some important IoT protocols.

Protocol	MQTT	AMQP	CoAP
Initial release year	1999	2003	2014

In the M2M communication protocols search category, we focused on finding relevant research articles surrounding ''MQTT'' published between the years 2000 and 2019. We also searched to get the number of published MQTT based research works every 5 years from 2000 up to 2019 ie: 2000-2004, 2005-2009, 2010-2014, and 2015-2019. The purpose was to calculate the exponential growth rate of MQTT research in a 5-year moving window period since its arrival in 1999, up to 2019. To understand how MQTT related research works stands out in comparison to other IoT protocols, we used keywords like ''CoAP'' [12], [27], [28] and ''AMQP'' [20], [29]. As the initial release of CoAP and AMQP dates to years 2014 and 2003 respectively, we took the last five years' literature search data surrounding CoAP and AMQP for a fair comparison with the last five years' literature search data surrounding MQTT.

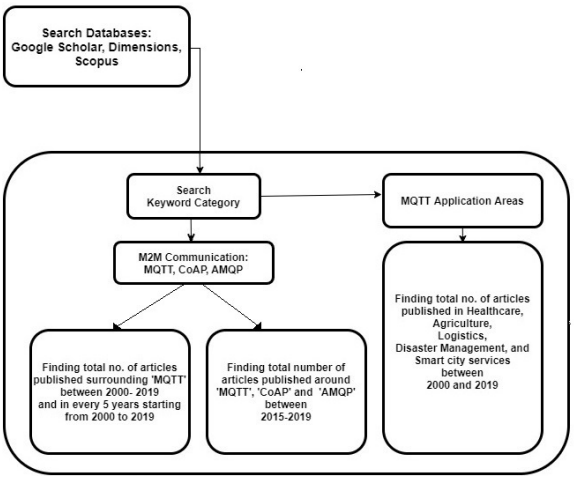

FIGURE 5. Overview of the literature search method.

The second search category was MQTT application areas. In this category following keywords, were used: ''MQTT in Healthcare'', ''MQTT in Agriculture'', ''MQTT in Logistics'', ''MQTT in Disaster Management'', and ''MQTT in Smart city Services''. Period filter was applied to find published scholarly articles between 2000 and 2019 (20 years' data). This independent literature search was conducted on 7th November 2019. All the articles that explicitly matched the search criteria were included in this study. Figure 5 shows an overview of the used method.

The total number of MQTT based research works found between 2000 to 2019 on Google Scholar, Dimensions database, and Scopus are 16,461, 7,078, and 1048 respectively. See Table 3 and 4.

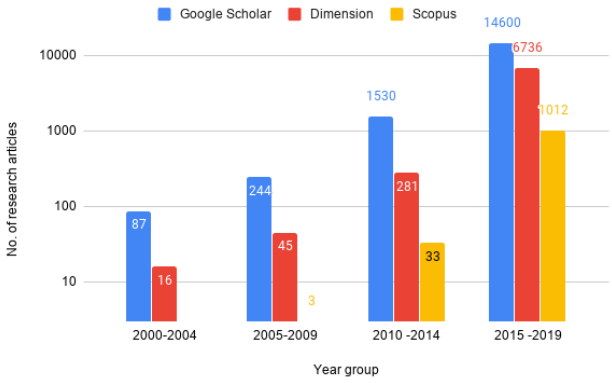

FIGURE 6. Number of research works in MQTT between 2000 to 2019.

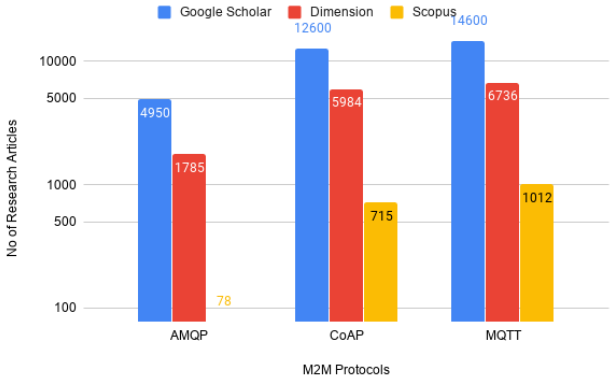

FIGURE 7. Number of research works in MQTT, CoAP and MQTT between 2015 to 2019.

To analyze the progress in MQTT-based research works in every five-year interval starting from 2000 to 2019, we sampled search data in a five-year moving window between the mentioned period. On Google Scholar, the total number of MQTT based research works found between 2000-2004, 2005-2009, 2010-2014, and 2015-2019 are 87, 244, 1,530, and 14,600 respectively. On Dimensions, the total number of MQTT based research works found between 2000-2004, 2005-2009, 2010-2014, and 2015-2019 are 16, 45, 281, and 6,736 respectively. On Scopus, the total number of MQTT based research works found between 2000-2004, 2005-2009, 2010-2014, and 2015-2019 are 0, 3, 33, and 1,024 respectively. See Figure 6.

Hence, in a 5-year window between 2004 and 2019, MQTT-based research works show an average annual exponential growth rate percent (see Equation 1)

$$\ln\left(\frac{\frac{Present}{Past}}{No.\ of\ years}\right) * 100 \tag{1}$$

of 32.02 on Google Scholar search, on Dimensions, it is 33.77%, and in Scopus, MQTT based research works register an average annual exponential growth rate of 36.39 percent. See Table 5.

We also investigated ongoing research around other M2M IoT protocols, namely AMQP and CoAP. As the initial

TABLE 3. IoT protocols related literature search data, 7th Nov 19.

Google Scholar	No. of related documents			Dimensions	No. of related documents			Scopus	No. of related documents		
Year/Keywords	MQTT	CoAP	AMQP	Year/Keywords	MQTT	CoAP	AMQP	Year/Keywords	MQTT	CoAP	AMQP
2000-2004	87	-	-	2000-2004	16	-	-	2000-2004	0	-	-
2005-2009	244	-	-	2005-2009	45	-	-	2005-2009	3	-	-
2010-2014	1.530	-	-	2010-2014	281	-	-	2010-2014	33	-	-
2015-2019	14,600	12,600	4950	2015-2019	6,736	5,984	1,785	2015-2019	1,012	715	78

TABLE 4. MQTT application areas related literature search data, 7th Nov 19.

	Keywords	No. of related documents		Keywords	No. of related documents		Keywords	No. of related documents
Google Scholar: 2000-2019 (20 years'data)	MQTT in healthcare	3,870	Dimensions: 2000-2019 (20 years'data)	MQTT in healthcare	2,050	Scopus: 2000-2019 (20 years'data)	MQTT in healthcare	38
	MQTT in agriculture	1,850		MQTT in agriculture	1,264		MQTT in agriculture	26
	MQTT in logistics	1,560		MQTT in logistics	1,474		MQTT in logistics	19
	MQTT in disaster management	1,180		MQTT in disaster management	1,000		MQTT in disaster management	1
	MQTT protocol in smart city services	5,110		MQTT protocol in smart city services	3,161		MQTT protocol in smart city services	70

TABLE 5. Average annual exponential growth in MQTT based research work, 2000-2019.

Time period (year-group)	No of Research Articles	5 Year Growth			
		Absolute (Present-Past)	Percentage ((Absolute/past)*100)	Avg. annual exponential growth rate percent (ln(Present/past))/ No. of years)*100	Avg. annual exponential growth rate percent from 2004 to 2019
Google Scholar					
2000-2004	87				
2005-2009	244	157	180.46	20.63	
2010 -2014	1,530	1,286	527.05	36.72	
2015 -2019	14,600	13,070	854.25	45.12	
					32.02
Dimensions					
2000-2004	16				
2005-2009	45	29	181.25	20.69	
2010 -2014	281	236	524.45	36.64	
2015 -2019	6,736	6,455	2297.16	63.54	
					37.77
Scopus					
2000-2004	0				
2005-2009	3	3			
2010 -2014	33	30	1000	47.96	
2015 -2019	1,012	979	2966.67	68.47	
					36.39

release years of CoAP and AMQP are 2014 and 2015 respectively 2, we decided to gather the last five years' research works surrounding those protocols. Between 2015 and 2019: 12,600 CoAP related articles were found on Google Scholar, 5,984 CoAP based articles were found on Dimensions, and "715" CoAP related articles were found in Scopus. Similarly, the total number of AMQP related articles between 2015 and 2019 found on Google Scholar, Dimensions and Scopus are 4,950, 1,785, and 78 respectively. In terms of the last five years' total number of research publications, MQTT stands tall among other M2M protocols (CoAP, AMQP) with 14,600 number of published articles. See Figure 7.

Considering MQTT application areas, we aggregated the last 20 years (2000-2019) MQTT related publication data. On Google scholar, out of 13,570 published articles in numerous domains, 3,870 articles were found to be related to the healthcare domain, 1,850 articles were related to the agriculture domain, 1,560 articles were in the logistics domain, 1,180 articles were in the disaster management domain, and 5,110 articles were in smart city services domain. On Dimensions, out of 8949 published articles 2,050 articles were found to be related to the healthcare domain, 1,264 articles were related to the agriculture domain, 1,474 articles fell into the logistics domain, 1,000 articles were in the disaster

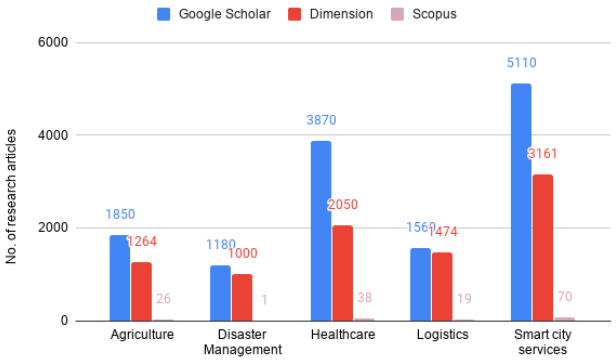

FIGURE 8. MQTT application areas: 2000-2019.

management domain, and 3,161 articles in smart city services domain. On Scopus listing, out 154 published articles, 38 were related to healthcare, 26 were related to agriculture, 19 were related to logistics, only one article was related to disaster management, and 70 articles were related to smart city services – see Figure 8.

VI. INTRODUCTION OF MQTT IMPLEMENTATIONS AND TAXONOMY OF FEATURES

In this section, we present an introduction and detailed comparison of some of the currently available MQTT implementations (brokers and client libraries) for IoT communication. We compare brokers and client libraries that implement versions 5.0 and/or 3.1.1 and/or 3.1 of the MQTT Protocol. All data collected for this comparison are from the respective official documentation/website/ GitHub links of the MQTT implementations. The comparison is limited to features of MQTT protocol and uses the stable version of each implementation.

A. INTRODUCTION OF MQTT IMPLEMENTATIONS

The MQTT protocol provides a lightweight way of using a publish/subscribe model for messaging [8]. In this subsection, we will introduce some of the widely used MQTT brokers and client libraries.

Mosquitto [30] is an EPL/EDL licensed open-source message broker. It is developed by Eclipse Foundation and supports 3.1, 3.1.1.1, and 5.0 versions of the MQTT protocol. It is a single-threaded non-scalable implementation. It is written in C. It also offers a C library for implementing MQTT clients.

Bevywise MQTT Route [31] is a commercially licensed, closed-source MQTT-based message broker developed by Bevywise Networks. It is built on MQTT 3.1 and 3.1.1.1 specifications and implements all QoS levels of MQTT. It is a fixed-threaded (2threads) non-scalable implementation. It is written in C and Python. It also provides MQTT client implementation.

EMQ X [32] is developed by EMQ Inc. It is an Erlang-based open-source message broker/client built on MQTT 5.0 specifications. It is highly scalable supports clustering. It can run virtually anywhere starting from edge to cloud. It uses Apache License Version 2.0.

HiveMQ CE (Community Edition) [33] is a Java-based open-source scalable MQTT broker. It supports MQTT 3.x and MQTT 5.0. It is developed by HiveMQ GmbH. It is distributed with Apache License Version 2.0.

HiveMQ [34] is an MQTT based scalable, commercially licensed messaging platform built on the specifications of 3.x and 5.0 versions of MQTT protocol. It is written in Java and developed by HiveMQ GmbH. HiveMQ also has an MQTT client implementation.

IBM Watson IoT Platform Message Gateway [35] is a commercially licensed, scalable messaging service based on MQTT 3.x and 5.0. It offers 40+ MQTT client libraries.

JoramMQ [36] is an open-source, scalable message broker based on MQTT 3.x specifications. It is developed by ScalAgent. It has both commercial and free distributions distributed in commercial license and GNU Lesser General Public License respectively.

flespi [37] is a commercially licensed, scalable IoT platform developed by Gurtam. It is written in C and implements 3.1, 3.1.1, and 5.0 versions of MQTT specification.

PubSub+ [38] is an event broker that implements MQTT 3.1.1 specification. It is developed by Solace and available in both free and commercial versions.

Thingstream [39] is a scalable IoT communications-as-a-service platform offered by u-blox. It implements MQTT 5.0 specification and offers both MQTT broker and client library. It is commercially distributed.

VernemQ [40] is an open-source, highly scalable MQTT broker developed by Octavo Labs AG. It is written in Erlang and implements MQTT 3.x and 5.0 versions. It uses Apache License version 2.0.

RabbitMQ [41] is an erlang based message broker that supports MQTT 3.1.1 version. It is developed by Pivotal Software. It uses MPL 1.1 license.

Apache ActiveMQ [42] is an open-source, scalable, multi-protocol message server written in Java and developed by Apache Software Foundation. It is being distributed in two flavors- "Classic" and "Artemis". It uses Apache License version 2.0. It implements the MQTT 3.1.1 specification.

Adafruit IO [43] is developed by Adafruit. It provides open-source MQTT client libraries for Python, Ruby, and Arduino. It implements MQTT 3.1.1 protocol and uses MIT license for software distribution. net-mqtt is an open-source MQTT implementation for Haskell language and developed by Dustin Sallings. It supports 3.x, 5.0 versions of the protocol. It uses BSD 3 license.

Eclipse Paho MQTT [44] offers open-source client libraries in Java, Python, JacaScript, GoLang, C, C++, Rust, and .Net(C#) languages. Paho C library implements all the versions of MQTT while libraries in all other languages only which implement versions 3.1 and 3.1.1 of the MQTT protocol. It uses Eclipse Public License 1.0, and Eclipse Distribution License 1.0 (BSD).

TABLE 6. Comparison of features of MQTT brokers with source availability model and design related properties. Key: "–" means unknown.

Features	Source code availability model		Design and Implementation			
MQTT Brokers	Open/Close	Software License	Written in	Supported OS	Latest stable release, release date	Developed by
Mosquitto	Open	Eclipse Public License 1.0, Eclipse Distribution License 1.0 (BSD)	C	Linux, Mac, Windows	1.6.9, 2020-02-27	Eclipse Foundation
Bevywise MQTT Route	Close	Commercial License	C, Python	Linux, Unix, MacOS, Windows, Raspbian	2.0, 2019-12-03	Bevywise Networks
EMQ X	Open	Apache License version 2.0	Erlang	Linux, Mac, Windows, BSD	3.0, 2019-04-03	EMQ Enterprise Inc.
HiveMQ CE	Open	Apache License version 2.0	Java	Linux, Mac, Windows	2020.3, 2020-07-06	HiveMQ GmbH
HiveMQ	Close	Apache License version 2.0	Java	Linux, Mac, Windows	4.3.5, 2020-07-17	HiveMQ GmbH
IBM WIoTP Message Gateway	Close	Commercial License	C	Linux	5.0.0.1, 2019-02-29	IBM
JoramMQ	Close	Commercial License, LGPL	Java	Linux, Unix, MacOS, Windows, Raspbian	1.13, 2019-04-29	ScalAgent
flespi	Close	Commercial License	C	-	-, 2018-04-05	Gurtam
PubSub+	Close	Commercial License, Free Version	C, C++	Linux, Mac, Windows	8.13, 2018-09-28	Solace
Thingstream	Close	Commercial License	C, C++, Java, JavaScript, Python, Go	-	3.3.0, 2019-03-14	u-box
VerneMQ	Open	Apache License version 2.0	Erlang	Linux, Mac	1.9.1, 2019-08-12	Octavo Labs AG
RabbitMQ	Open	MPL 1.1	Erlang	Linux, Unix, Mac, Windows, BSD	3.8.1, 2019-10-31	Pivotal Software
ActiveMQ	Open	Apache License version 2.0	Java	Windows, Unix, Linux, Cygwin	5.15.10, 2018-09-01	Apache Software Foundation
ActiveMQ Artemis	Open	Apache License version 2.0	Java	Windows, Unix, Linux, Cygwin	2.10.1, 2019-09-26	Apache Software Foundation

wolfMQT [45] is an open-source C MQTT client library for embedded use. It is developed by WolfSSL and available in GPL v2.0 and Commercial licenses.

Eclipse M2Mqtt [46] is an open-source, C# client library for .Net and WinRT platforms. It used Eclipse Public License 1.0.

Machine Head [47] is an open-source clojure based MQTT library distributed with Creative Commons Attribution 3.0 Unported License. It is developed by ClojureWerkz.

MQTT-C [48] is an MQTT based, open-source C language library developed by Liam Bindle. It is distributed with MIT license.

B. TAXONOMY OF FEATURES OF MQTT IMPLEMENTATIONS

To compare and analyze features of MQTT protocol implementations we introduced in the previous subsection, we define seven taxonomy categories. Tables 6, 7, 10 and 11 use these categories to compare the source code availability, the design and implementation, the protocol features, security, and finally data visualization support of the overviewed solutions. In the following paragraphs we define our taxonomy elements in detail, then provide a discussion for the comparison.

- **Source code availability model:** There are two types of source code availability model: closed-source and open-source. Under the open-source model source code of a released software product can be viewed and modified. Under the closed-source model source code is not made available in the public domain [49]. Various licenses are available for commercial and non-commercial software distributions and redistributions. A software license is a legal document that provides abiding guidelines for the use and distribution of software [50].
- **Source code metrics:** In this taxonomy element we chose source code metrics to present and compare various quality features of the source code of the examined MQTT brokers. Poor software quality makes the code difficult to read and understand, harder to guess the functionality or check for undefined symbols, and reuse the code. It adversely affects the overall productivity of the employees and the profitability of the company. In this work, we used a source code analyzer called CLOC [51] to calculate certain quality metrics. CLOC is a command line tool that takes file, directory, and/or archive names as inputs and recognizes applied programming languages to develop the software, counts the number of files, blank lines, comment lines, and lines of code. CLOC works with only open-source applications.

TABLE 7. Comparison of the examined open-source MQTT brokers with software metrics.

MQTT Broker	Prime programming language	No. of files	No of blank lines	No. of comment lines	Lines of code
VerneMQ	Erlang	286	6701	8649	46014
RabbitMQ	Erlang	340	14436	13361	95078
Mosquitto	C	540	12027	7628	58519
HiveMQ CE	Java	1349	34768	36414	127000
EMQX	Erlang	181	5005	4664	22370
ActiveMQ	Java	5058	113811	162830	474415

So in this comparison, we have only listed open-source MQTT brokers.

- **Design and Implementation:** This category focuses on software development related artifacts such as the programming language used to develop the application, supported operating systems or platforms, latest stable release, and date of release, cross-compilation ability, and the entity or company or individual behind the development of the application.
- **Protocol Features:** In this category, we compare various MQTT implementations according to the supported MQTT features such as QoS, Retain Flag, Persistent Session, Shared Subscriptions, Last Will and testament, Error Log, Built-in Gateway, MQTT Version(3.x/5.0), and availability of MQTT-SN Support, etc. ''Retain Flag'' when enabled (set to true) the broker stores the last retained message and the corresponding QoS for that topic. ''Persistent Session'' presents an ongoing connection to an MQTT message broker). ''Shared subscriptions'' is an MQTT V5 feature that allows MQTT clients to share a single subscription on the broker. ''last Will and testament'' is used to notify subscribers of an unexpected disconnection of the publisher. MQTT-SN refers to MQTT for sensor networks [52]. ''Built-in gateway'' refers to an MQTT gateway that acts as an intermediary between sensors/devices and any IoT platform.
- **Security:** Under this category, we list various security-related features implemented by various MQTT brokers and client libraries such as Authentication, MQTT Over TSL/SSL, TCP, WS/WSS, thread security, etc. For MQTT over TSL/SSL communications the port 8883 is reserved. MQTT over Websockets allows receiving MQTT data directly through a web browser.
- **Cloud offerings:** In this category, various MQTT implementations are compared in terms of Bare Metal availability, Cloud Hosting availability and Docker Support.
- **Data visualization support:** The following features form this category: UI and dashboard, Custom UI, ElasticSearch Integration, Tableau Support, and ElasticSearch integration. ''Tableau'' is a data visualization service and ElasticSearch integration'. It is a distributed RESTful search engine built for the cloud.

C. DISCUSSION

In this subsection, we provide further discussions on the comparison of the analyzed MQTT brokers and client libraries. In the previous subsections, we provided short introductions of the main properties of the overviewed works, defined the taxonomy elements used to categorize them, and provided classifications with comparison tables. Though the tables provide detailed information on the properties of MQTT implementations, we summarise some of the most valuable findings from these comparisons.

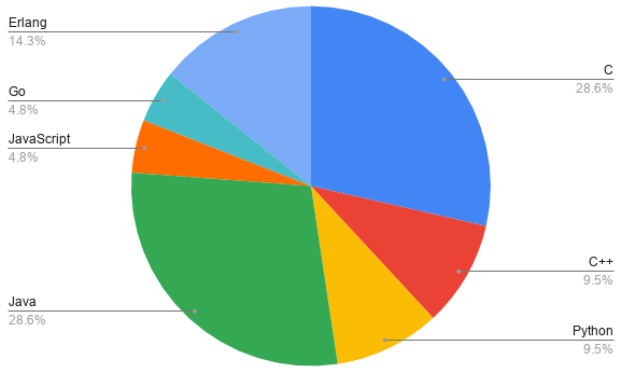

FIGURE 9. The ratio of the programming languages used to implement MQTT brokers.

Concerning the compared MQTT brokers, Table 6 shows that almost 50% of investigated solutions are open-source, and 50% of the tools are closed-source in nature. Concerning the implementation of the reviewed brokers, Figure 9 shows that almost 57.2 % of the solutions are written in C and Java where the contribution of each language amounts to 28.60%, 14.30% of solutions are written in Erlang, C++, and Python: each language shares of 9.50% of solutions, almost 4.8% of reviewed solutions are written in JavaScript and the percentage share of Go-based solutions also amounts to the same 4.8% of the solutions. Table 7 details the measured source code metrics of considered open-source MQTT brokers. We can see that ActiveMQ has the highest number of lines of code with a good number of comments. In terms of number of comment lines to lines of code ratio, ActiveMQ is followed by HiveMQ CE, EMQX, VerneMQ, RabbitMQ, and Mosquitto, respectively. Table 8 shows that almost all

TABLE 8. Comparison of features of MQTT brokers with protocol specific properties. Keys: "Yes" means supported, "No" means not supported, "–" means unknown.

Implemented protocol specific features									
MQTT Brokers	QoS Support	Retain Flag	Persistent Session	Shared Subscriptions	Last Will and Testament	Error Log	Built-in Gateway	MQTT Version	MQTT-SN Support
Mosquitto	0, 1, 2	Yes	Yes	No	Yes	No	No	3.1.1, 5.0	No
Bevywise MQTT Route	0, 1, 2	Yes	Yes	Yes	Yes	Yes, user can view in UI	Yes	3.x, 5.0	Yes
EMQ X	0, 1, 2	Yes	Yes	Yes	Yes	No	No	3.1.1	Yes
HiveMQ CE	0, 1, 2	Yes	Yes	Yes	Yes	No	No	3.x, 5.0	No
HiveMQ	0, 1, 2	Yes	Yes	Yes	Yes	No	No	3.x, 5.0	No
IBM WIoTP Message Gateway	0, 1, 2	Yes	Yes	-	-	Yes	Yes	3.x, 5.0	-
JoramMQ	0, 1, 2	Yes	Yes	Yes	-	Yes	No	3.x	Yes
flespi	0, 1, 2	Yes	Yes	Yes	Yes	Yes	No	3.1,3.1.1,5.0	No
PubSub+	0, 1, 2	Yes	Yes	Yes	Yes	Yes	Yes	3.1.1	No
Thingstream	0, 1, 2	-	-	No	-	-	No	5.0	Yes
VerneMQ	0, 1, 2	Yes	Yes	Yes	Yes	Yes	No	3.x, 5.0	No
RabbitMQ	0, 1	Partial	Yes	No	Yes	Yes	Yes	3.1.1	No
ActiveMQ	0, 1, 2	Yes	Yes	No	Yes	-	No	3.1.1	No
ActiveMQ Artemis	0, 1, 2	Yes	Yes	No	Yes	-	No	3.1.1	No

TABLE 9. Comparison of features of MQTT brokers with security related properties.

Security features			
MQTT Brokers	**Authentication**	**MQTT over TSL/SSL**	**WS/WSS**
Mosquitto	Yes	Yes	Yes
Bevywise MQTT Route	Yes	Yes	Yes
EMQ X	Yes	Yes	Yes
HiveMQ CE	Yes	Yes	Yes
HiveMQ	Yes	Yes	Yes
IBM WIoTP Message Gateway	Yes	Yes	Yes
JoramMQ	Yes	Yes	Yes
flespi	Yes	Yes	Yes
PubSub+	Yes	Yes	Yes
Thingstream	Yes	Yes	Yes
VerneMQ	Yes	Yes	Yes
RabbitMQ	Yes	Yes	Yes
ActiveMQ	Yes	Yes	Yes
ActiveMQ Artemis	Yes	Yes	Yes

solutions support most QoS categories, retain flag, LWT, and persistent connection features. RabbitMQ supports only QoS 0, and 1. Of all the reviewed broker solutions, up to 61.53% support the shared subscription feature, 57.14% of the solutions have already adopted MQTT 5.0, and 69.23% of the solutions have not implemented MQTT-SN, yet. Table 9 compares features of MQTT brokers with security-related properties. It shows that all the brokers have enabled security features through authentication, MQTT over TSL/SSL, and WS/WSS. Table 10 compares features of MQTT brokers with cloud and data visualization related properties. Of all reviewed solutions, around 64.28% of the tools support cloud hosting, 85.71 % of the solutions have Docker container availability, 92.3% have UI and dashboard. Only a very few number of solutions support bare metal, ElasticSearch and Tableau services integration.

For the compared MQTT client libraries, Table 11 shows that almost 83.33% of solutions are open-source in nature. Concerning the implementation of the reviewed brokers, Figure 10 shows that almost 22.9% of the solutions are written in C, 17.1% solutions are in Java. C and Java-based solutions are followed by C++(11.4%), Python(11.4%), Erlang (8.6%), C#(5.7%), PHP(5.7%), Perl(5.7%), Ruby (5.7%), Go (2.9%), JavaScript (2.9%) based solutions. Paho MQTT and Thingstream support the highest number of programming languages, i.e C, C++, Java, JavaScript, Python, and Go. Among all the overviewed library solutions, HiveMQ MQTT client, net-mqtt, MQTT-C, Mosquitto, and EMQ X support thread-safety feature. MQTT client libraries, like net-mqtt, Paho-MQTT, wolfMQTT, MQTT-C, Mosquitto, and EMQ X have support cross-compilation. Almost all the library solutions support most QoS categories.

From this comparison, we found that 'Docker container', 'MQTT over WS/ WSS Support', 'authentication', 'REST API integration', 'MQTT Over TSL/SSL', 'TCP Support', 'Retain Flag', 'persistent session', 'UI and Dashboard', 'Last Will and testament' and 'QoS' are the most supported features by all the brokers. We also found that RabbitMQ does Not support QoS2 subscriptions. RabbitMQ automatically downgrades QoS 2 to QoS1 [41]. Some of the least supported features are, 'MQTT-SN Support', 'built-in gateway','localization support', 'bare-metal', 'Tableau support', 'rule engine', and 'Error Log', etc.

TABLE 10. Comparison of features of MQTT brokers with cloud and data visualization related properties, Keys: "Yes" means supported, "No" means not supported, "–" means unknown.

Features	Cloud offerings			Data visualization support		
MQTT Brokers	Bare Metal	Cloud Hosting	Docker Container Availability	UI and dashboard	ElasticSearch Integration	Tableau Support
Mosquitto	-	Yes	Yes	No	No	No
Bevywise MQTT Route	Yes	Yes	Yes	Yes	Yes	Yes
EMQ X	Yes	Yes	Yes	Yes	No	No
HiveMQ CE	-	Yes	Yes	Yes	No	No
HiveMQ	-	Yes	Yes	Yes	No	No
IBM WIoTP Message Gateway	Yes	Yes	Yes	Yes	No	No
JoramMQ	-	No	No	-	Yes	No
flespi	-	Yes	Yes	Yes	No	Yes
PubSub+	Yes	Yes	Yes	Yes	No	No
Thingstream	-	Yes	No	Yes	No	No
VerneMQ	-	No	Yes	Yes	No	No
RabbitMQ	-	No	Yes	Yes	No	No
ActiveMQ	-	No	Yes	Yes	No	No
ActiveMQ Artemis	-	No	Yes	Yes	No	No

TABLE 11. Comparison of features of MQTT client libraries. Key: "Yes" means supported, "No" means not supported, "–" means unknown.

Features	Source code availability model		Design			Implemented Protocol features			Security		
MQTT Client Libraries	Open/Close	Software License	Written in	Supported OS	Ability to cross compile	MQTT Version	QoS	MQTT-SN	TSL/SSL	WS/WSS	Thread Safety
Adafruit IO	Open	MIT License	Ruby on Rails, Node.js, Python	CentOS, Debian, Docker, Mac OS X, Linux, Raspbian	-	-	0, 1	-	-	-	-
HiveMQ MQTT Client	Open	Apache License Version 2.0	Java	Linux, Mac, Windows	-	3.x, 5.0	0, 1, 2	-	Yes	Yes	Yes
net-mqtt	Open	BSD 3	Haskell	-	Yes	3.x, 5.0	0, 1, 2	Yes	Yes	-	Yes
Paho MQTT	Open	Eclipse Public License 1.0, Eclipse Distribution License	C, C++, Java, JavaScript, Python, Go	Varies upon language. Details Here: https://www.eclipse.org/paho/downloads.php	Yes (For C, C++ clients)	3.x,5.0 (only C and Java client library)	0, 1, 2	Yes	Yes	Yes	-
Thingstream	Close	Commercial License	C, C++, Java, JavaScript, Python, Go	-	-	5.0	0, 1, 2	Yes	Yes	-	-
wolfMQTT	Open	GNU Public License Version 2, Commercial License	C	Win32/64, Linux, Mac OS X, FreeRTOS Microchip, Harmony, Nucleus	-	3.1.1, 5.0	0, 1, 2	Yes	Yes	-	-
M2Mqtt	Open	Eclipse Public License 1.0	C#	Windows		3.x	0, 1, 2	-	Yes	-	-
Machine Head	Open	Creative Commons Attribution 3.0 Unported License	Clojure	Mac OS X, Linux	-	3.x	-	-	-	-	-
MQTT-C	Open	MIT License	C	-	Yes	3.x	0, 1, 2	-	Yes	-	Yes
Mosquitto	Open	Eclipse Public License 1.0, Eclipse Distribution License 1.0 (BSD)	C	Windows, BSD, Linux macOS, QNX	Yes	3.x, 5.0	0, 1, 2	No	Yes	Yes	Yes
EMQ X	Open	Apache License version 2.0	Erlang	Linux, Unix, MacOS, Windows,Raspberry P	Yes	3.x, 5.0	0, 1, 2	Yes	Yes	Yes	Yes
Bevywise MQTT Route	Close	Commercial license	C, Python	Linux, Unix, MacOS, Windows, Raspberry P	-	3.x, 5.0	0, 1, 2	Yes	Yes	Yes	-

VII. SUMMARY OF RESEARCH TRENDS, OPEN ISSUES AND CHALLENGES

From our survey, it is quite evident that MQTT is one of the most widely used IoT protocol solutions. The lightweight designing principle of MQTT enables data-exchange in a plain-text format that represents a security threat. Hence, several MQTT broker implementations enable encryption as a separate feature on the top of TLS. It results in some performance overhead as discussed in [22]. Currently, many MQTT brokers use the CONNECT control type message packet to enable authentication. Brokers require clients to send usernames and passwords with the CONNECT message for the validation of the connection failing which connection is refused. We found that enhancing security for MQTT is an

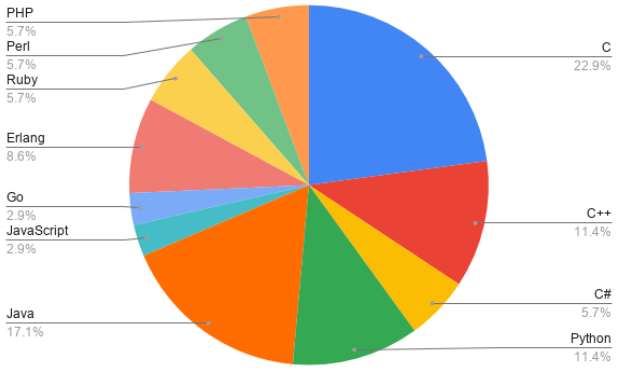

FIGURE 10. The ratio of the programming languages used to implement MQTT client libraries.

ongoing effort. Various authorization methods are continually being developed, experimented, and tried out to make the security aspects of MQTT better.

Confidentiality is paramount for securing a system [53], and this can be achieved by encrypting the messages to be published at the application layer. The encryption can either be implemented as a client-to-broker model or end-to-end model. In a client-to-broker type model of encryption, brokers decrypt the messages being transmitted on a topic head, encrypt them and send to the clients that are subscribed to that specific topic. This process requires more computing power and energy. Whereas in an end-to-end encryption model, a broker does not require to decrypt the information being transmitted on various topic heads. The broker just functions as a post office to send the message to its appropriate recipients. This process involves the consumption of less computing power and energy [21]. The commercial and technical interest in Wireless Sensor Networks (WSNs) is on a rise [54]. A typical Wireless Sensor Network consists of a large number of battery-operated sensors and actuators having limited computational resources. These devices need to communicate with each other.

Based on our survey the following open issues are found:

- It is evident that MQTT-SN, which is an important MQTT v.5 feature, needs to be largely adopted by more broker implementations;
- nevertheless, since MQTT-SN is a successor of MQTT, it also inherits the previous security issues of authentication and encryption, so a little compromise in an MQTT-SN communication can have an impact on the whole IoT infrastructure in terms of confidentiality and accessibility. Solving the security problem in MQTT would bring to it a great advantage over other available protocol solutions.
- Since big amount of data is being generated by IoT systems, various brokers may expand their support for various database programs, AI-enabled search, and MQTT data analytics.
- The MQTT standard architecture defines only one broker in a system, hence, it is not suitable for edge-based IoT applications and cannot harness the best out of a multi-core environment. As the world is rapidly moving forward in distributed and edge-based computing direction more research on multi-threaded, scalable MQTT implementations, edge-based MQTT prototypes and MQTT-SN security are required to bridge the gap [55].
- Finally, we would like to bring attention to privacy issues of data flows happening within an IoT System. Although there are some privacy-related works (e.g. [56] and [57]) in IoT in general, there is still a large body of research needed on protocol-specific privacy solutions.

VIII. CONCLUSION

Things, as basic components for data source, are paramount in the Internet of Things paradigm. No matter which communication technology is used to deploy and operate an IoT/M2M network, all of the participating, independent data generating devices heavily rely on the special messaging protocols designed for M2M communication within IoT applications.

In this paper, we introduced the set of various M2M communication protocols appeared in the past 20 years, and surveyed the evolution and usage of the MQTT protocol, which is the most widespread M2M/IoT protocol. We have comprehensively analyzed some of the important research works of the current literature to highlight the main features, advantages, and limitations of the MQTT protocol and its broker implementations over other IoT protocols. We presented our findings around the current use of MQTT and its application areas using various graphs and comparison tables. We arrived to an in-depth comparative study of the features of several MQTT brokers and client libraries in various taxonomy categories, which can be used to empower researchers and users to select an MQTT implementation based on their requirements and suitability. We also highlighted future research directions that need to address security and scalability issues to further improve the effectiveness of these solutions.

Concerning our future works, we plan to perform detailed performance measurements and usage evaluation of concrete MQTT broker implementations as an extension to this study.

ACKNOWLEDGMENT

This research was supported by the Hungarian Scientific Research Fund under the grant number OTKA FK 131793, and by the University of Szeged Open Access Fund under the grant number 5053.

REFERENCES

[1] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A survey on application layer protocols for the Internet of Things," *Trans. IoT Cloud Comput.*, vol. 3, no. 1, pp. 11–17, 2015.

[2] Cisco. *The Internet of Things Reference Model*. Accessed: Aug. 23, 2020. [Online]. Available: http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf

[3] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013, doi: 10.1016/j.future.2013.01.010.

[4] Z. Ning, F. Xia, X. Hu, Z. Chen, and M. S. Obaidat, "Social-oriented adaptive transmission in opportunistic Internet of smartphones," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 810–820, Apr. 2017, doi: 10.1109/TII.2016.2635081.

[5] Curtin University. *Introduction to the Internet of Things*. Accessed: Aug. 23, 2020. [Online]. Available: https://study.curtin.edu.au/offering/mooc-introduction-to-the-internet-of-things–iot1x

[6] R. Buyya, J. Broberg, and A. M. Goscinski, *Cloud Computing: Principles and Paradigms*. Hoboken, NJ, USA: Wiley, 2011.

[7] *OASIS Website*. Accessed: Oct. 10, 2020. [Online]. Available: https://www.oasis-open.org/org

[8] (Oct. 29, 2014). *MQTT Version 3.1.1. Edited by Andrew Banks and Rahul Gupta*. OASIS Standard. Accessed: Oct. 10, 2020. [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html and [Online]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[9] Statista. *IoT: Number of Connected Devices Worldwide 2012–2025*. Accessed: Aug. 23, 2020. [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[10] Y. Ai, M. Peng, and K. Zhang, "Edge cloud computing technologies for Internet of Things: A primer," *Digit. Commun. Netw.*, vol. 4, no. 2, pp. 77–86, 2018.

[11] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in *Proc. IEEE Int. Syst. Eng. Symp. (ISSE)*, Oct. 2017, pp. 1–7, doi: 10.1109/syseng.2017.8088251.

[12] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight Internet protocols for Web enablement of sensors using constrained gateway devices," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Jan. 2013, pp. 334–340, doi: 10.1109/iccnc.2013.6504105.

[13] T. Jaffey. *MQTT and CoAP, IoT Protocols*. Accessed: Aug. 23, 2020. [Online]. Available: https://eclipse.org/615community/eclipsenewsletter/2014/febru-ary/article2.php

[14] A. Foster, "Messaging technologies for the industrial Internet and the Internet of Things, version 1.7—July 2014," PrismTech, Mumbai, India, White Paper, 2015, p. 21.

[15] N. S. Han, "Semantic service provisioning for 6LoWPAN: Powering Internet of Things applications on Web," Ph.D. dissertation, Distrib. Syst. Group ETH Zurich, Zürich, Switzerland, 2015.

[16] M. A. Prada, P. Reguera, S. Alonso, A. Morán, J. J. Fuertes, and M. Domínguez, "Communication with resource-constrained devices through MQTT for control education," *IFAC-PapersOnLine*, vol. 49, no. 6, pp. 150–155, 2016, doi: 10.1016/j.ifacol.2016.07.169.

[17] S. Wagle, "Semantic data extraction over MQTT for IoTcentric wireless sensor networks," in *Proc. Int. Conf. Internet Things Appl. (IOTA)*, Jan. 2016, pp. 227–232, doi: 10.1109/iota.2016.7562727.

[18] Y. Xu, V. Mahendran, and S. Radhakrishnan, "Towards SDN-based fog computing: MQTT broker virtualization for effective and reliable delivery," in *Proc. 8th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2016, pp. 1–6, doi: 10.1109/comsnets.2016.7439974.

[19] T. Yokotani and Y. Sasaki, "Comparison with HTTP and MQTT on required network resources for IoT," in *Proc. Int. Conf. Control, Electron., Renew. Energy Commun. (ICCEREC)*, Sep. 2016, pp. 1–6, doi: 10.1109/iccerec.2016.7814989.

[20] J. E. Luzuriaga, M. Perez, P. Boronat, J. C. Cano, C. Calafate, and P. Manzoni, "A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks," in *Proc. 12th Annu. IEEE Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2015, pp. 931–936, doi: 10.1109/ccnc.2015.7158101.

[21] D. Dinculeană and X. Cheng, "Vulnerabilities and limitations of MQTT protocol used between IoT devices," *Appl. Sci.*, vol. 9, no. 5, p. 848, Feb. 2019, doi: 10.3390/app9050848.

[22] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for Internet of Things and related challenges of fog and cloud computing integration," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–29, Feb. 2019, doi: 10.1145/3292674.

[23] A. Niruntasukrat, C. Issariyapat, P. Pongpaibool, K. Meesublak, P. Aiumsupucgul, and A. Panya, "Authorization mechanism for MQTT-based Internet of Things," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC)*, May 2016, pp. 290–295, doi: 10.1109/iccw.2016.7503802.

[24] M. Singh, M. A. Rajan, V. L. Shivraj, and P. Balamuralidhar, "Secure MQTT for Internet of Things (IoT)," in *Proc. 5th Int. Conf. Commun. Syst. Netw. Technol.*, Apr. 2015, pp. 746–751, doi: 10.1109/csnt.2015.16.

[25] X. Wang, J. Zhang, E. M. Schooler, and M. Ion, "Performance evaluation of attribute-based encryption: Toward data privacy in the IoT," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 725–730, doi: 10.1109/icc.2014.6883405.

[26] C. Lesjak, D. Hein, M. Hofmann, M. Maritsch, A. Aldrian, P. Priller, T. Ebner, T. Ruprechter, and G. Pregartner, "Securing smart maintenance services: Hardware-security and TLS for MQTT," in *Proc. IEEE 13th Int. Conf. Ind. Informat. (INDIN)*, Jul. 2015, pp. 1243–1250, doi: 10.1109/indin.2015.7281913.

[27] N. De Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali, "Comparison of two lightweight protocols for smartphone-based sensing," in *Proc. IEEE 20th Symp. Commun. Veh. Technol. Benelux (SCVT)*, Nov. 2013, pp. 1–6, doi: 10.1109/scvt.2013.6735994.

[28] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," in *Proc. IEEE 9th Int. Conf. Intell. Sensors, Sensor Netw. Inf. Process. (ISSNIP)*, Apr. 2014, pp. 1–6, doi: 10.1109/issnip.2014.6827678.

[29] H. Subramoni, G. Marsh, S. Narravula, P. Lai, and D. K. Panda, "Design and evaluation of benchmarks for financial applications using advanced message queuing protocol (AMQP) over InfiniBand," in *Proc. Workshop High Perform. Comput. Finance*, Nov. 2008, pp. 1–8, doi: 10.1109/whpcf.2008.4745404.

[30] *Mosquittopub Man Page*. Accessed: Aug. 23, 2020. [Online]. Available: https://mosquitto.org/man/mosquitto_pub-1.html

[31] *Bevywise MQTT Route Developer Document*. Accessed: Aug. 23, 2020. [Online]. Available: https://www.bevywise.com/675mqtt-broker/developer-guide/

[32] *Emq X Broker—High Performance MQTT Message Broker Documentation*. Accessed: Aug. 23, 2020. [Online]. Available: https://docs.emqx.io/broker/latest/en/

[33] *HiveMQ Community Edition*. Accessed: Aug. 23, 2020. [Online]. Available: https://github.com/680hivemq/hivemq-community-edition

[34] *HiveMQ Documentation*. Accessed: Aug. 23, 2020. [Online]. Available: https://www.hivemq.com/docs/hivemq/4.3/

[35] *IBM Knowledge Center*. Accessed: Aug. 24, 2020. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSWMAJ_5.0.0.1/com.ibm.ism.doc/welcome.html

[36] *MQTT Protocol*. Accessed: Aug. 24, 2020. [Online]. Available: http://www.scalagent.com/en/jorammq-33/technology-36/mqtt-protocol

[37] *flespi MQTT Broker—MQTT 5.0 Compliant, Secure, Fast, and Free*. Accessed: Aug. 24, 2020. Online]. Available: https://flespi.com/mqtt-broker

[38] *Solace Pubsub+*. Accessed: Aug. 24, 2020. [Online]. Available: https://www.solace.com

[39] *Homepage—Thingstream by u-blox IoT Communication-as-a-Service*. Accessed: Aug. 24, 2020. [Online]. Available: https://thingstream.io/

[40] *VerneMQ Documentation*. Accessed: Aug. 24, 2020. . [Online]. Available: https://docs.vernemq.com/

[41] *Rabbitmq Tutorials*. Accessed: Aug. 24, 2020. [Online]. Available: https://www.rabbitmq.com/getstarted.html

[42] *Apache ActiveMQ*. Accessed: Aug. 24, 2020. [Online]. Available: http://activemq.apache.org/

[43] *Welcome to Adafruit IO*. Accessed: Aug. 24, 2020. [Online]. Available: https://io.adafruit.com/

[44] *Eclipse Paho*. Accessed: Aug. 24, 2020. [Online]. Available: https://www.eclipse.org/paho/

[45] *wolfMQTT Client Library*. Accessed: Aug. 24, 2020. [Online]. Available: https://www.wolfssl.com/products/wolfmqtt/

[46] *M2mqtt Releases*. Accessed: Aug. 24, 2020. [Online]. Available: https://github.com/eclipse/paho.mqtt.m2mqtt/releases

[47] *Clojure Mqtt Client*. Accessed: Aug. 24, 2020. [Online]. Available: https://github.com/clojurewerkz/720machine_head

[48] *Mqtt-C*. Accessed: Aug. 24, 2020. [Online]. Available: https://github.com/LiamBindle/MQTT-C

[49] J. W. Paulson, G. Succi, and A. Eberlein, "An empirical study of open-source and closed-source software products," *IEEE Trans. Softw. Eng.*, vol. 30, no. 4, pp. 246–256, Apr. 2004.

[50] J.-A. Ligeti, "Software license compliance system and method," U.S. Patent 20 040 143 746 A1, Jul. 22, 2004.

[51] *AlDanial/cloc*. Accessed: Aug. 24, 2020. [Online]. Available: https://github.com/AlDanial/cloc

[52] K. Govindan and A. P. Azad, "End-to-end service assurance in IoT MQTT-SN," in *Proc. 12th Annu. IEEE Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2015, pp. 290–296, doi: 10.1109/ccnc.2015.7157991.

[53] S. Katsikeas, K. Fysarakis, A. Miaoudakis, A. Van Bemten, I. Askoxylakis, I. Papaefstathiou, and A. Plemenos, "Lightweight & secure industrial IoT communications via the MQ telemetry transport protocol," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2017, pp. 1193–1200.

[54] *MQTT For Sensor Networks (MQTT-SN)*. Accessed: Aug. 24, 2020. [Online]. Available: https://www.mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf
[55] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.
[56] N. J. Al Fardan and K. G. Paterson, "Lucky thirteen: Breaking the TLS and DTLS record protocols," in *Proc. IEEE Symp. Secur. Privacy*, May 2013, pp. 526–540, doi: 10.1109/sp.2013.42.
[57] X. Li, J. Xu, Z. Zhang, D. Feng, and H. Hu, "Multiple handshakes security of TLS 1.3 candidates," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 486–505, doi: 10.1109/sp.2016.36.

BISWAJEEBAN MISHRA received the M.Sc. degree in computer sciences from Ravenshaw University, India, in 2010. He is currently a member of the IoT Cloud Research Group, Department of Software Engineering, University of Szeged, Hungary, and a pre-doctoral student with the Doctoral School of Computer Science, University of Szeged. Prior to joining the doctoral programme in 2015, he had served in the IT industry in India for five years. His research interests include the IoT Systems, communication efficiency of the IoT systems, and cloud computing.

ATTILA KERTESZ is currently an Associate Professor with the Software Engineering Department, University of Szeged, Hungary, leading the IoT Cloud Research Group of the Department. His research interests include the federative management of the IoT, fog and cloud systems, and data management issues of distributed systems in general. He is currently the Leader of a national project OTKA FK 131793 financed by the Hungarian Scientific Research Fund, and a work package leader in the GINOP IoLT project, financed by the Hungarian Government and the European Regional Development Fund. He is also a Management Committee member of the INDAIRPOLLNET and CERCIRAS COST actions. He has also participated in several successful European projects, including ENTICE EU H2020, COST IC1304, COST IC0805, SHIWA, S-Cube EU FP7, and the CoreGRID EU FP6 Network of Excellence projects. He was a member of numerous program committees for European conferences and workshops, and has published over 100 scientific papers, having more than 1000 independent citations.