future internet

MDPI

Article

Communication Protocols of an Industrial Internet of Things Environment: A Comparative Study

Samer Jaloudi

Department of Information and Communication Technology, Al Quds Open University, Nablus 00407, West Bank, Palestine; sgalodi@qou.edu or samer.jaloudi@ieee.org; Tel.: +970-0599-376-492

Received: 4 February 2019; Accepted: 4 March 2019; Published: 7 March 2019

Abstract: Most industrial and SCADA-like (supervisory control and data acquisition) systems use proprietary communication protocols, and hence interoperability is not fulfilled. However, the MODBUS TCP is an open de facto standard, and is used for some automation and telecontrol systems. It is based on a polling mechanism and follows the synchronous request–response pattern, as opposed to the asynchronous publish–subscribe pattern. In this study, polling-based and event-based protocols are investigated to realize an open and interoperable Industrial Internet of Things (IIoT) environment. Many Internet of Things (IoT) protocols are introduced and compared, and the message queuing telemetry transport (MQTT) is chosen as the event-based, publish–subscribe protocol. The study shows that MODBUS defines an optimized message structure in the application layer, which is dedicated to industrial applications. In addition, it shows that an event-oriented IoT protocol complements the MODBUS TCP but cannot replace it. Therefore, two scenarios are proposed to build the IIoT environment. The first scenario is to consider the MODBUS TCP as an IoT protocol, and build the environment using the MODBUS TCP on a standalone basis. The second scenario is to use MQTT in conjunction with the MODBUS TCP. The first scenario is efficient and complies with most industrial applications where the request–response pattern is needed only. If the publish–subscribe pattern is needed, the MQTT in the second scenario complements the MODBUS TCP and eliminates the need for a gateway; however, MQTT lacks interoperability. To maintain a homogeneous message structure for the entire environment, industrial data are organized using the structure of MODBUS messages, formatted in the UTF-8, and then transferred in the payload of an MQTT publish message. The open and interoperable environment can be used for Internet SCADA, Internet-based monitoring, and industrial control systems.

Keywords: automation; IIoT; MQTT; MODBUS; publish–subscribe; request–response; SCADA

1. Introduction

The Internet of Things (IoT) is an emerging technology that represents a cost-effective, scalable, and reliable ecosystem, proposed for many applications, including smart city sectors [1,2], consumer devices [3,4], industrial environments [5,6], Internet of vehicles [7,8], multimedia [9,10], and 5G systems [11,12]. The IoT platform, from the communications perspective, consists of a TCP/IP network and standard protocols [13]. Standard protocols primarily include advanced message queuing protocol version 1.0 (AMQP 1.0) [14–16], message queuing and telemetry transport (MQTT) [17], constrained application protocol (CoAP) [18], extensible messaging and presence protocol (XMPP) [19], and JavaScript object notation (JSON) [20]. The TCP/IP networks include Wi-Fi [21], Internet, Intranet, and modern mobile networks. The employment of a communication infrastructure and protocol depends on the field of application, the timing requirements, and the data transmission rates [13]. Therefore, the requirements of consumer electronics are different from those of smart city sectors and industrial applications.

In the industry context, Industrial Internet of Things (IIoT) [22,23], Industry 4.0 [24,25], Smart Manufacturing [26,27], and Smart Factory [28,29] are all terminologies for the emerging industrial environments that employ information and communication technologies (ICTs), including IoT platforms, while maintaining industry requirements. The term IIoT refers to the use of IoT technologies in many fields of application such as manufacturing, factories, transportation, gas, oil, and electric grids. However, most industrial and SCADA-like (supervisory control and data acquisition) systems of these fields employ proprietary communication protocols and ICTs, which lead to closed industrial systems. Hence, customers are stuck to a single vendor, costs are high, and interoperability is not fulfilled.

In this paper, standard IoT protocols are introduced and compared. Then, MQTT is chosen for machine-to-machine (M2M) communications to complement the MODBUS TCP [30–32] operations in an IIoT environment. This environment integrates an event-based message-oriented protocol, i.e., MQTT, with a polling-based request–response protocol, intended for industrial applications, i.e., MODBUS TCP. The study shows that the MODBUS TCP and MQTT can coexist together, and in parallel, within the same IIoT environment. While industrial requirements of control and monitoring can be met via the MODBUS TCP using the request–response model, the MQTT protocol complements its operation by fulfilling the IoT requirements, using the publish–subscribe pattern for M2M communications. The IoT protocol works in parallel with the MODBUS TCP and relays industrial data to an Internet-based server for remote monitoring, analysis, and archiving. To solve the interoperability problem, industrial data are formatted using the MODBUS message format and transferred in the payload of the MQTT publish messages. In fact, the IoT protocols cannot replace the MODBUS protocol in most industrial applications, especially when an optimized message structure is required in the application layer. A second scenario is proposed here, which is to consider the MODBUS TCP as an IoT protocol, and hence build the IIoT environment with the MODBUS TCP only. This solution is efficient and complies with most industrial applications; however, it only depends on the request–response model. The choice of solution is totally dependent on the requirements of the industrial application.

The developed environment can be employed in many industrial applications, including Internet SCADA, Internet-based monitoring, and industrial control systems. Hence, customers are not stuck to a single vendor, costs are reduced, and interoperability is maintained.

The remainder of the paper is organized as follows. Section 2 presents a related literature survey, Section 3 presents a theoretical background of the MODBUS protocol, Section 4 compares IoT protocols, and Section 5 compares MQTT and the MODBUS TCP. Section 6 compares the latencies and resource usage of both protocols. Section 7 presents a discussion and Section 8 concludes the paper.

2. Related Work

In the context of industry, researchers are proposing and examining protocols, networks, and middleware architectures for industrial ICT infrastructure and integration.

For integration purposes, the authors of [33] proposed a data-oriented machine-to-machine (M2M) communication middleware based on the ZeroMQ platform for IIoT applications. The researchers of [34] presented a case study for controlling industrial robots and monitoring energy consumption remotely based on Ebbits middleware, which transforms data into web services. Another service-oriented IIoT middleware is proposed in [35] for balancing task allocation between a mobile terminal and a utility cloud service. In [36], a platform based on SystemJ for IIoT is proposed using an FPGA (field programmable gate array), and then tested in an automation system. Reference [37] describes a collaboration-oriented M2M (CoM2M) messaging mechanism for IIoT, based on the platform PicknPack food packaging line. In [38], legacy flexible manufacturing systems are integrated with the SCADA system of an Industry 4.0 framework via Ethernet. However, these research papers investigated middleware architectures for integration purposes.

The infrastructure of an IIoT was examined in [39], where the IoT vision in industrial wireless sensor networks was implemented using the IPv6 over low-power wireless personal area network

(6LoWPAN) and CoAP. Furthermore, the author of [40] proposed a software-defined network for an IIoT based on new networking technologies. However, these research papers investigated network solutions and architectures.

Infrastructure protocols were examined in many studies. For example, the authors of [41] developed an edge IoT gateway to extend the connectivity of MODBUS devices to IoT by storing the scanned data from MODBUS devices locally, and then transferring the changes via an MQTT publisher to MQTT clients via a broker. The researchers of [42] designed and implemented a web-based real-time data monitoring system that uses MODBUS TCP communications, following which all data are displayed in a real-time chart in an Internet browser, which is refreshed at regular intervals using hypertext transfer protocol (HTTP) polling communications. In [43], a MODBUS serial protocol was reported that collects data serially via a RS-232 protocol and transfers the collected data over the ZigBee protocol. In [44], measurements of field devices collected by the MODBUS serial protocol were transferred over HTTP using a Wi-Fi network. However, these papers proposed a gateway as a bridge between MODBUS and the Internet, Intranet, or wireless network.

In this study, the MODBUS TCP is proposed to be implemented in two scenarios—either alone, or in parallel with an MQTT publisher without a gateway. The first solution proposes the MODBUS TCP as an IoT protocol to build the industrial environment on a standalone basis. Using the second arrangement, the MODBUS TCP executes its operations while maintaining the requirements of industrial applications and MQTT achieves the M2M communications for IoT functions.

3. MODBUS Theory

The MODBUS TCP is a byte-oriented, industrial communication protocol, open de facto standard, used for data exchange between embedded systems, devices, and industrial applications. Devices, reacting as clients, may benefit from the inexpensive implementation of such a lightweight protocol for polling industrial devices that react as servers. Polling communications follow the request–response mechanism, where a client queries the server for specific data or executes commands in the server using a frame of bytes arranged in a specific way, called a frame format. The server replies to the client queries via a frame of bytes either holding measurement data from sensors or confirming the execution of commands. Sixteen-bit data registers store measurement values, and coils hold the status of ON and OFF switches. Therefore, MODBUS TCP uses the polling mechanism, as opposed to the event-based mechanism, explained in the next section.

As listed in Table 1, the protocol specifications [30] define three categories of function codes for the access of data in remote devices. These data are stored in coils or registers as status values for measurements or transferred as setpoints for control. Coils perform one-bit read and write operations for switching the attached devices ON and OFF or reading and writing one-bit internal configuration values. Discrete inputs perform one-bit read operations for reading the status of the attached devices, whether they are switched ON or OFF. The 16-bit input registers are responsible for measurements from physical devices, and the 16-bit holding registers perform read and write operations related to internal reconfigurable values.

Table 1. Function codes for data access in MODBUS.

Data Access	Type	Function Code	Meaning
1 bit	physical discrete input	0x02	read discrete inputs
1 bit	internal bits, physical coils	0x01	read coils
1 bit	internal bits, physical coils	0x05	write single coil
1 bit	internal bits, physical coils	0x0F	write multiple coils
16 bit	physical input registers	0x04	read input registers
16 bit	internal and physical output registers	0x03	read holding registers
16 bit	internal and physical output registers	0x06	write single register
16 bit	internal and physical output registers	0x10	write multiple registers
16 bit	internal and physical output registers	0x17	read/write registers
16 bit	internal and physical output registers	0x16	mask write register
16 bit	internal and physical output registers	0x18	read first in first out (FIFO) queue

A message structure of a MODBUS TCP client query for reading input registers is shown in Figure 1. The slave replies to the master query in the same format with the read registers using the function code (FC) "read input registers" (FC = 0x04), or as a confirmation to executing commands in case of other function codes such as "write single coil" (FC = 0x05).

The header of the MODBUS frame consists of four fields: a two-byte transaction identifier (ID); a two-byte protocol type (MODBUS over TCP); a two-byte length, which counts the number of bytes for the rest fields; and a one-byte unit identifier (Unit). However, the protocol data unit (PDU) consists of a one-byte function code (FC), which is, here, a code to read the registers and a data field that may contain other fields depending on the FC itself. Both the header and the PDU form an application data unit (ADU), which is the complete frame of the query.

The following illustrative example explains the principle of the MODBUS frame format that uses a function code (0x04) to read three continuous input registers in a remote device. The function is able to read from 1 to 125 contiguous input registers. Here, a client query asks a server to read the values of three continuous input registers—register address "14" (0x000E), register address "15" (0x000F), and register address "16" (0x0010). Therefore, the client sends a single message "000100000060104000E0003" and the server replies by sending one frame "000100000009010406FE20 6666A63F" that contains three values of continuous registers. The first register contains the hexadecimal value "0xFE20," which corresponds to the sixteen-bit signed short integer value "11111110 00100000" or the decimal value "–480". The last two registers hold the IEEE 754 short floating-point [45] representation "0x3FA66666" or the decimal value "1.3".

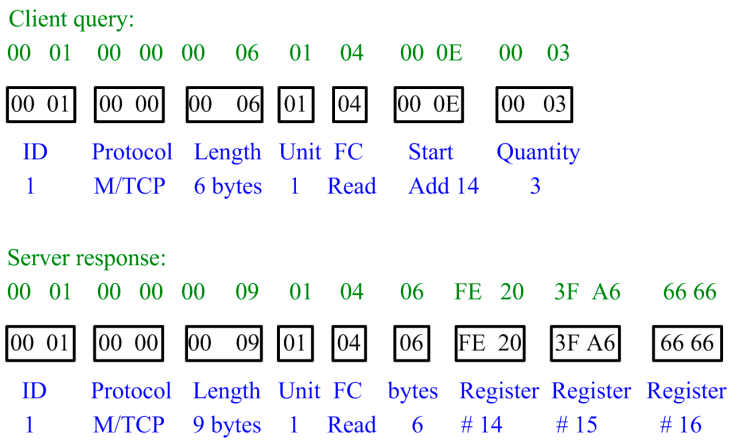


Figure 1. MODBUS query and response, an illustrative example.

Using the MODBUS specifications [30], the information from Table 1, and the MODBUS frame of Figure 1, the total consumed bytes (size) can be calculated via Equation (1). The formula is derived for the function code of "read input registers" (0x04) and is plotted in Figure 2, where N refers to the number of registers, which is doubled because each register contains two bytes:

$$\begin{aligned} Size &= \text{request\_bytes} + \text{response\_bytes} \\ &= (7+5) + (9+2\times N) \\ &= 2\times N + 21 \end{aligned} \quad . \tag{1}$$

For the request case, the header occupies 7 bytes, the function code occupies 1 byte, the start register-address occupies 2 bytes, and the quantity occupies 2 bytes. For the response case, the header occupies 7 bytes, the function code 1 byte, and the length 1 byte.

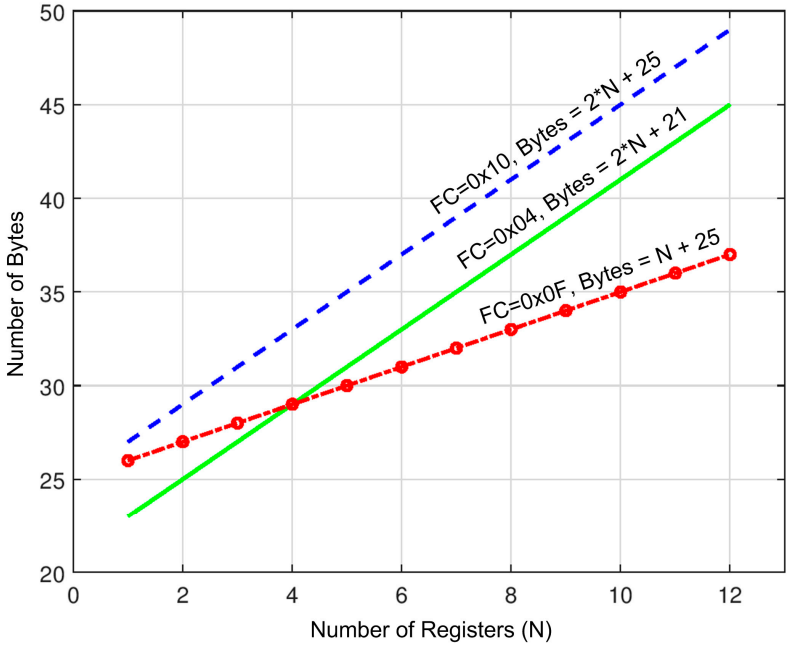


Figure 2. Relationship between the number of bytes and the number of registers of function codes (FC) "read input registers" (0x04), "write multiple coils" (0x0F), and "write multiple registers" (0x10).

The same principle is applied to other function codes, such as "read holding registers" (0x03), which has the same linear equation; "write multiple registers" (0x10), which can be represented by the linear equation ($2*N + 25$); and "write multiple coils" (0x0F), which can be represented by the linear equation ($N + 25$). As an example, if the number of registers (N) is 4, the size is 29 for the "read input registers" (0x04) and 33 for the "write multiple registers" (0x10) function codes.

If the client application requires the execution of both operations—read and write—within the same message for a remote device, the function code "read/write multiple registers" (0x17) shows higher performance than both the "read input registers" (0x04) and the "write multiple registers" (0x10) function codes if used separately. Equation (2) illustrates two linear formulae for the function code "read/write multiple registers" (0x17), which sends both commands within the same message:

$$\begin{aligned} &\text{Bytes\_request} = 17 + 2 \times N_{\text{Read}} \\ &\text{Bytes\_response} = 9 + 2 \times N_{\text{Write}} \\ &\therefore \text{Total\_size} = 26 + 2 \times (N_{\text{Read}} + N_{\text{Write}}) \end{aligned} \quad . \tag{2}$$

Both equations of the function code "read/write multiple registers" (0x17) request and response are plotted in Figure 3. Both equations are linear and depend on the number of registers to be written (N_{Write}) and read (N_{Read}).

As a result, the MODBUS TCP has an optimized frame structure suitable for SCADA-like systems and has a communication mechanism that fulfills the industrial requirements. In addition, it has a communication model and pattern that are compatible with industrial applications. As shown in Figures 1–3, the protocol is lightweight. Moreover, it has an open specification, and uses TCP/IP networks. Accordingly, it can be considered as an IoT protocol.

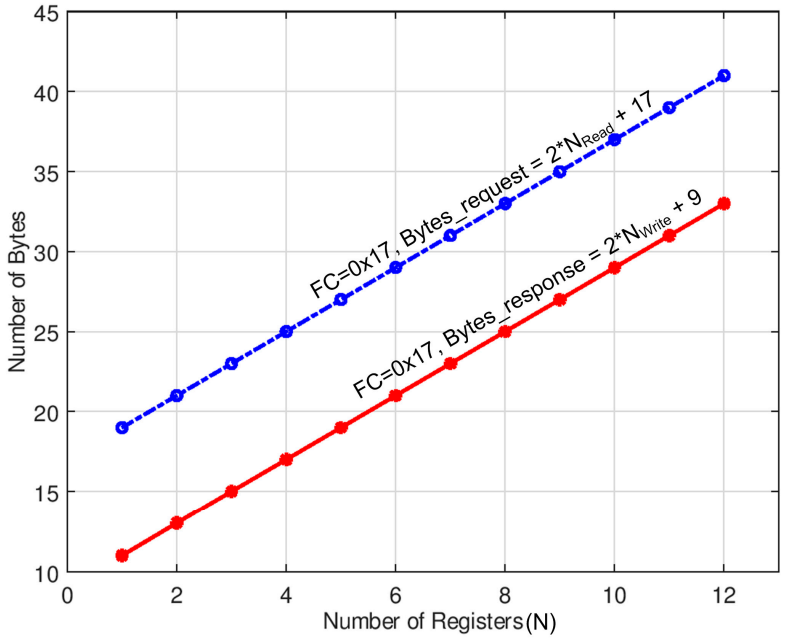


Figure 3. Relationship between the number of bytes and the number of registers of the MODBUS function code (FC) "read/write multiple registers" (0x17).

4. Comparison between IoT Protocols

A performance comparison between HTTP and MQTT is conducted in [46] on required network resources for IoT, while the payload was fixed to zero bytes and the topic names were varied. In addition, a performance analysis of MQTT, HTTP, and CoAP was performed in [47] for IoT-based monitoring of a smart home. The authors of [48] discussed and analyzed the efficiency, usage, and requirements of MQTT and CoAP. Moreover, the authors of [49] compared AMQP and MQTT over mobile networks, and the authors of [50] emulated a quantitative performance assessment of CoAP in comparison with HTTP.

In this section, the main differences between HTTP, CoAP, MQTT, AMQP, XMPP, and MODBUS TCP protocols are discussed from various telecommunication aspects. Then, a protocol is chosen that fulfills the requirements of the IIoT environment.

Accordingly, Table 2 summarizes these differences from different communication aspects including infrastructure, architecture, mechanism, model, messaging pattern, methodology, and transmission paradigm. These protocols use the client–server communication architecture. HTTP uses the request–response model and is a document-oriented protocol, whereas MQTT uses the publish–subscribe model and is message-oriented. Thus, MQTT is one-to-many, and HTTP is one-to-one (peer-to-peer). CoAP uses a specific infrastructure—namely, 6LoWPAN (IEEE 802.15.4)—which employs IPv6 in the network layer. Both MQTT and HTTP use an inexpensive and available communication infrastructure, which is Internet or Intranet in wire mode (Ethernet—IEEE 802.3) or wireless mode (Wi-Fi—IEEE 802.11)—which may employ either IPv4 or IPv6 in the network layer. In the transport layer, MQTT and HTTP protocols use TCP port numbers 1883 and 80, respectively. However, CoAP uses UDP port number 5683. Given that MQTT is event-based, it is a message-oriented protocol. Thus, CoAP mimics HTTP in using polling-based messaging, but in a shorter time and smaller frame-size.

Table 2. Comparison of Internet of Things (IoT) protocols.

Feature	HTTP	CoAP	MQTT	MODBUS TCP
infrastructure	Ethernet, Wi-Fi	6LoWPAN	Ethernet, Wi-Fi	Ethernet, Wi-Fi
network layer	IPv4 or IPv6	IPv6	IPv4 or IPv6	IPv4 or IPv6
transport layer	TCP	UDP	TCP	TCP
transport port	80, 443	5683	1883, 8883	502, 802
model	synchronous	asynchronous	asynchronous	synchronous
pattern	request—response	both	publish—subscribe	request—response
mechanism	one-to-one	one-to-one	one-to-many	one-to-one
methodology	document-oriented	document-oriented	message-oriented	byte-oriented
paradigm	long polling-based	polling-based	event-based	polling-based
quality level	one level	two: CON or NON	three: QoS 0, 1, 2	one level
standard	IETF (RFC7230)	IETF (RFC7252)	ISO/IEC, OASIS	modbus.org
encoding	ASCII text	RESTful (Binary)	UTF-8 (Binary)	Binary
security	SSL, TLS	DTLS	SSL, TLS	TLS

CoAP is an application layer protocol, dedicated to communication with constrained devices in IPv6-based IoT infrastructures. Two communication patterns are used by CoAP, i.e., publish–subscribe and request–response [51]. The CoAP messaging pattern is based on the exchange of messages between endpoints and uses a short fixed-length binary header that may be followed by a compact binary option and a payload. Compared to HTTP, as shown in Figure 4, CoAP runs over the connectionless UDP in the transport layer, whereas in the network layer, CoAP uses either IPv6 or 6LoWPAN. When CoAP uses IPv6, it is necessary for it to use Ethernet or Wi-Fi for the data link and physical layers, respectively. When CoAP uses 6LoWPAN, it employs IEEE 802.15.4e for the data link and physical layers.

The content (payload) of HTTP may vary according to the type of transferred data, called content-type, which could be plain text, HTML, XML, GIF image, PDF application, or audio. For the exchange of data using HTTP, XML is used, which handles verbose plain text for solving interoperability issues. However, for CoAP, the efficient XML interchange (EXI) [52] is used, which encodes verbose XML documents in binary format, if interoperability is considered. This is normally used for constrained devices to increase the performance and decrease the consumed power. Hence, CoAP is suitable for constrained devices in IoT-based wireless sensor networks that employ IPv6-based infrastructure. However, it needs a gateway to exchange data over the Internet.

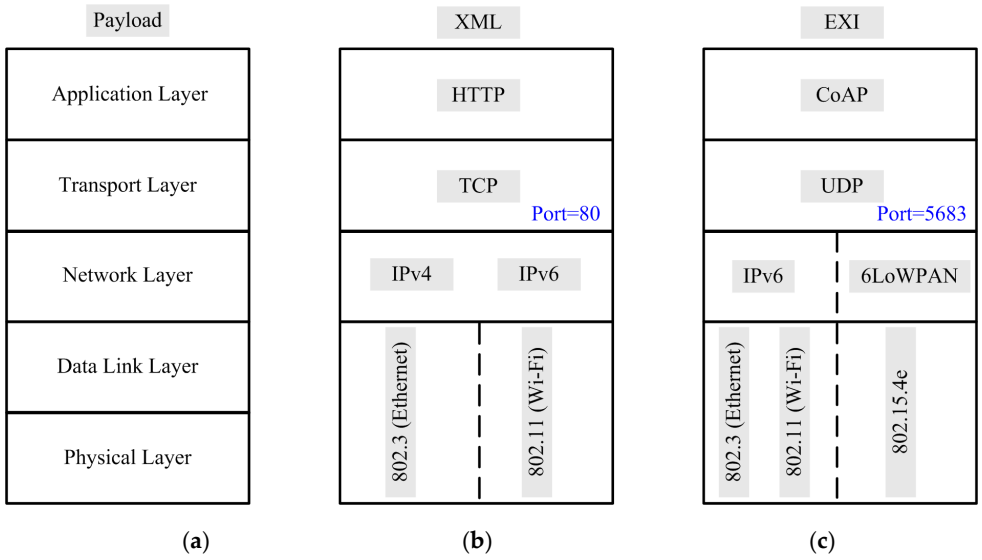


Figure 4. Communication protocols in the IEEE model (**a**); the HTTP (**b**); the CoAP (**c**).

There are two common features and two main differences between MQTT and CoAP. Both target constrained devices and networks, and both have low data overhead. Importantly, MQTT is one-to-many and TCP-based message-oriented, whereas CoAP is one-to-one and UDP-based. Since TCP is connection-oriented, and UDP is connectionless, MQTT is more reliable. Moreover, CoAP needs a dedicated communication infrastructure based on IPv6 networks in addition to a dedicated gateway to pass content over the Internet. Hence, CoAP was not used for this IIoT environment. In the same manner, HTTP, which is polling-based, was not considered for this study because of its philosophy that uses the synchronous communication model for peer-to-peer and request–response exchange of data.

XMPP is an XML-based messaging protocol that is able to transfer verbose messages, audio and video signals in chat conversations. In addition, it supports request–response as well as an event-based communication pattern. However, the XML-based verbose messages of XMPP increase the message size of SCADA-like applications, which have byte-oriented messages, and hence, cannot be used for IIoT efficiently. As a comparison, MODBUS has small-sized data units, with a maximum of 255 bytes, which are suitable for automation, telecontrol, and monitoring, whereas XMPP messages need more than 400 bytes of overhead.

To summarize, many IoT protocols exist, and event-based protocols are of considerable interest for transferring data as notifications to complement the MODBUS TCP. This MODBUS protocol is polling-based, synchronous, request–response, and optimized for control and monitoring in industrial applications. It can establish an IIoT environment, either on a standalone basis or in conjunction with an event-based protocol to cover the publish–subscribe mechanism if needed. MQTT is able to complement the MODBUS TCP via its asynchronous model, event-based paradigm, and publish–subscribe pattern. Alternatively, HTTP uses a request–response mechanism and, hence, was not considered for this study. CoAP was also found to be not suitable for this scenario because it needs a specific infrastructure, and hence, a gateway to pass data over the Internet, which adds more costs and causes complications to the environment. In addition, XMPP was not considered here, because it is XML-based verbose protocol that requires a large overhead for small-sized industrial packets. Moreover, AMQP was eliminated because it is dedicated to the exchange of business messages between two entities. This protocol is used normally for application-to-application integration at the enterprise level, which is higher than the level of both MODBUS TCP and MQTT.

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), have been designed to provide security over a TCP/IP network. If HTTP uses SSL or TLS, it employs port 443. This is also applicable to MODBUS TCP and MQTT, which employ ports 802 and 8883, respectively.

5. Comparison between MODBUS TCP and MQTT

In this section, a comparison between the MODBUS TCP and the MQTT protocol is conducted from three aspects. These aspects are the communication model of the protocol in the original IEEE model, the message exchange philosophy, and the required number of bytes for some message types that demonstrate the overhead of each type. However, the other protocols are not compared for the reasons listed above.

As shown in Figure 5, the MQTT and the MODBUS TCP protocols are both in the same level in the IEEE model. While the MQTT protocol encodes the user data in UTF-8, the MODBUS TCP uses a byte-encoded frame format for the user data, which is intended for industrial applications.

Figure 6 illustrates a comparison between MQTT philosophy and MODBUS philosophy, from the perspective of the exchange of messages. The request of the MODBUS query uses a TCP-based connection and employs a frame-format based on an application-layer message structure, which is optimized and dedicated for telecontrol and monitoring. The case is different with MQTT; while the first client (publisher) produces an event using four messages, the second client (subscriber) consumes that event in six messages.

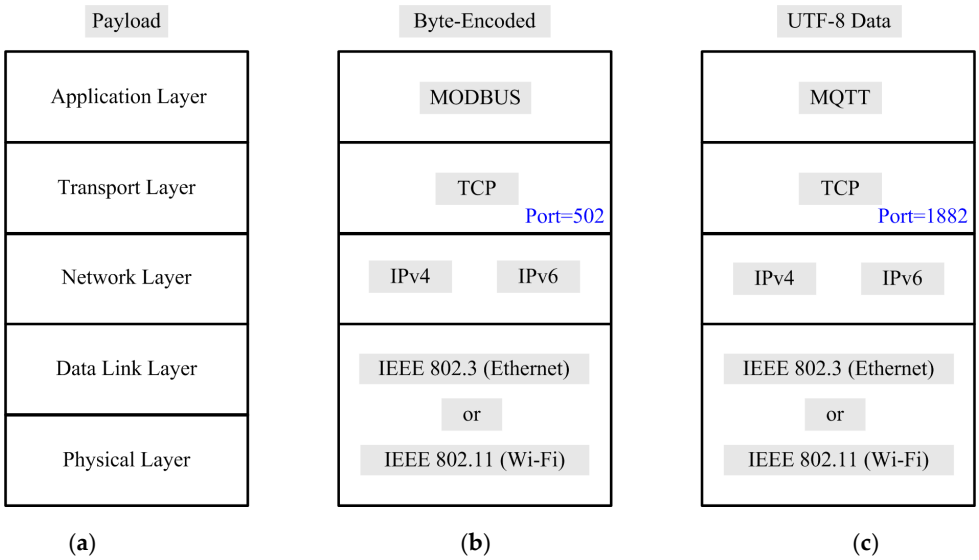


Figure 5. The IEEE model (**a**); compared to the MODBUS TCP (**b**); and the MQTT (**c**).

The publisher sends a connect packet (CONNECT), with username (un) and password (pwd) if required, to the server (broker) trying to establish a TCP connection. The server acknowledges the attempt with a CONNACK packet, telling the client (publisher) whether the connection is successfully established. Then, the client publishes the temperature value, for example, via a PUBLISH packet, with temp topic and a value of 22.7. The client ends the publish event with the server by sending a DISCONNECT packet.

Meanwhile, and in addition to the aforementioned steps, the subscriber must SUBSCRIBE to the same topic (temp) to receive the published messages of interest. The subscription packet is acknowledged with the SUBACK packet. When the broker receives the message that handles the temperature value, via the same PUBLISH packet, it forwards it to the subscriber, which may then terminate the connection using the DISCONNECT message.

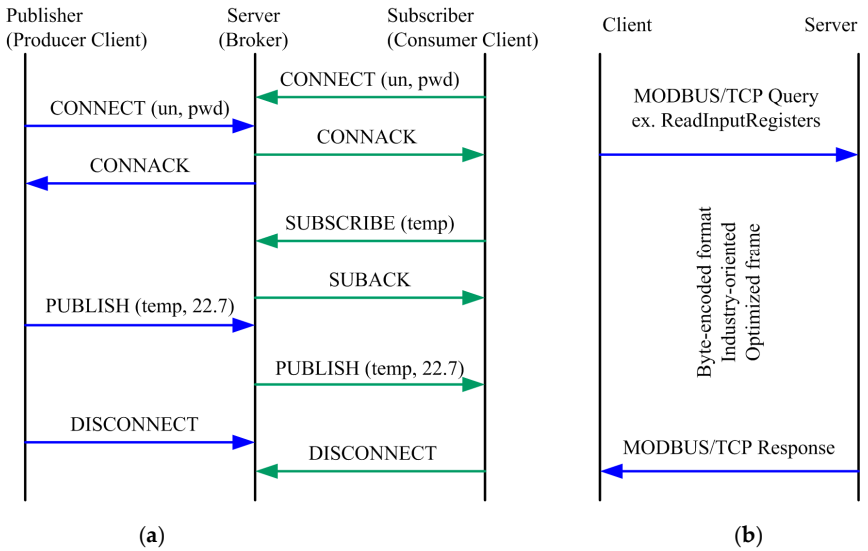


Figure 6. Comparison of protocols for the exchange of messages: (**a**) MQTT; (**b**) MODBUS TCP.

As a comparison between MODBUS TCP and MQTT messages, Figure 7 shows the required number of bytes of the MODBUS messages, compared to those of MQTT. The message of the MODBUS TCP requires 27 bytes to read the values of three registers using the function code "read input registers" (FC = 0x04), as shown in Figures 1 and 2. However, the message of the MODBUS TCP requires 29 bytes

to write the values of two registers using the function code "write multiple registers" (FC = 0x10) in a remote device. In both cases, the MODBUS TCP issues a query from the client side and a response from the server side. Meanwhile, if the function code "read/write multiple registers" (0x17) is used for both operations, the total consumed bytes for reading three registers and writing two registers within the same connection is 36 bytes. For MQTT, in this example, publishing a 10-byte payload by the producer and consuming it by the subscriber requires 115 bytes. It is important to mention that the total consumed bytes of MQTT publish and subscribe messages depends on the topic length and the length of the user data.

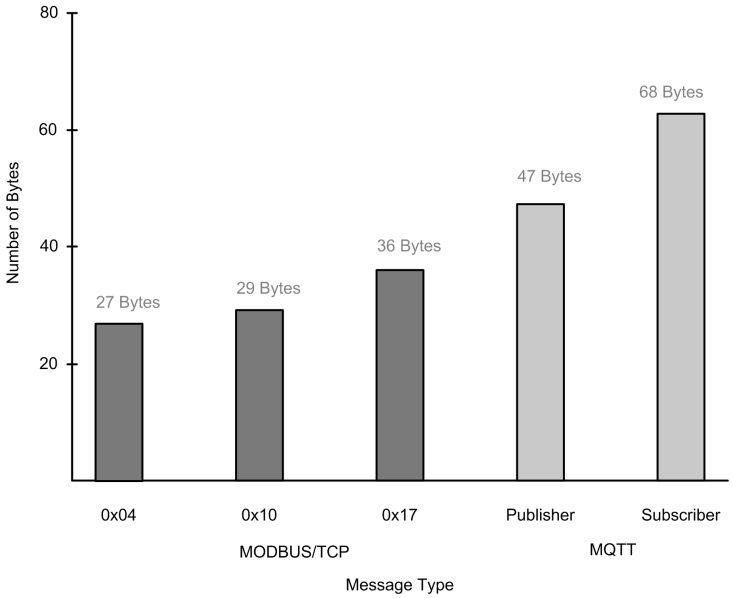


Figure 7. Required bytes of MQTT messages compared to those of MODBUS.

Figure 8 shows the overhead portion of the three messages for a constant payload of 10 bytes. The function code "read/write multiple registers" (0x17) has an overhead of 26 bytes out of 36 bytes. The function codes "read input registers" (0x04) and "write multiple registers" (0x10) have a total overhead of 46 bytes out of 56-bytes (the total size). In comparison, the MQTT publish–subscribe message has an overhead of 105 bytes out of 115 bytes.

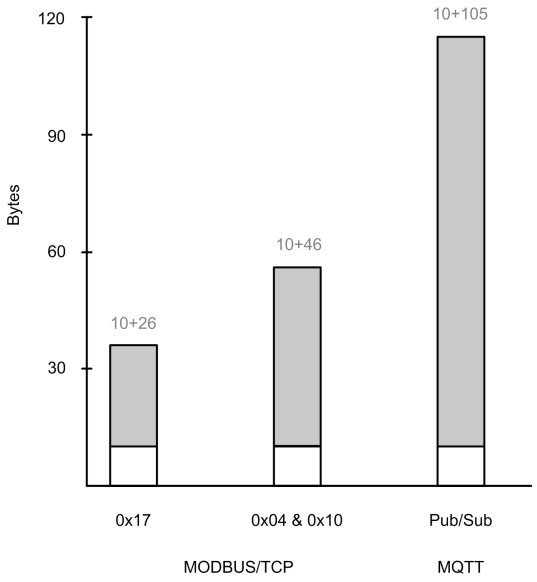


Figure 8. MQTT and MODBUS overheads for a payload of 10 bytes.

The result of Figure 8 is illustrated in Figure 9 for values between 2 bytes and 106 bytes. The figure demonstrates the overhead percentage with respect to the total message size according to Equation (3):

$$\text{Overhead}(\%) = \frac{\text{Overhead}}{\text{Overhead} + \text{Payload}} \times 100\%. \quad (3)$$

The percentage overheads of the aforementioned function codes are calculated and plotted in Figure 9. The figure shows values for fewer than 106 bytes because the message size in industrial applications and SCADA-like systems is small. MQTT has the highest overhead; the function code "read input registers" (0x04) has the least overhead (21 bytes), and the function code "write multiple registers" (0x10), which is covered by the curve of the function code "read/write multiple registers (0x17), has 25 bytes of overhead. However, if the application client reads and writes registers in the same message, the function code "read/write multiple registers" (0x17) shows lower overhead than the function codes "read input registers" (0x04) and "write multiple registers" (0x10) if both are used separately for the same purpose.

Accordingly, the MQTT protocol is suitable for IoT-based publish–subscribe applications, but it is unable to replace the MODBUS TCP, which fulfills the industrial requirements and uses an optimized frame-format for request–response communications between industrial clients and servers. These properties make it suitable for SCADA-like systems, automation, monitoring, and control. In Section 6, performance of MQTT and the MODBUS TCP are tested and compared based on the Round-Trip time (RTT) measurements and the central processing unit (CPU) usage.

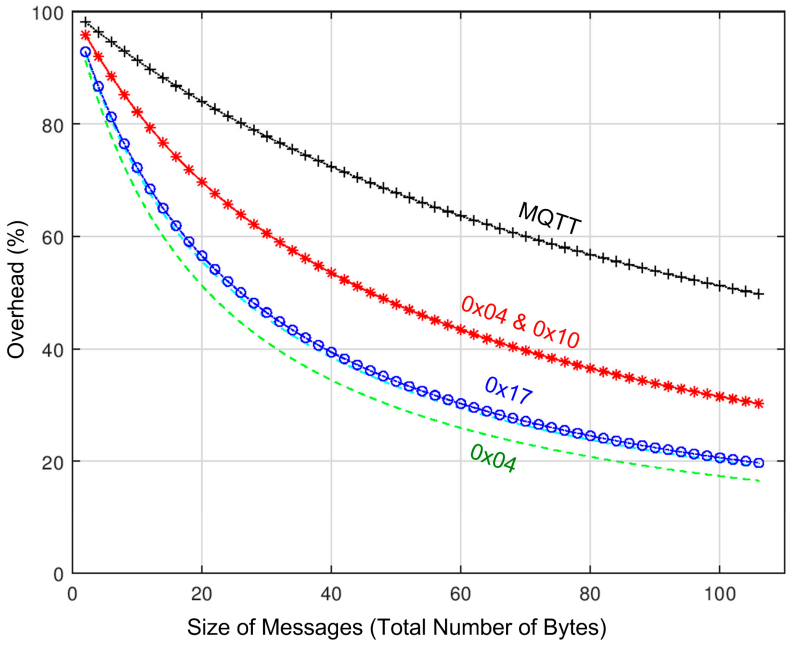
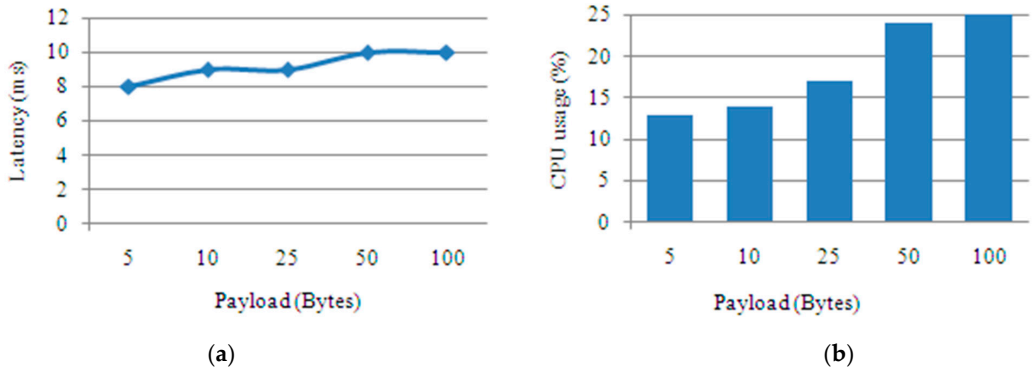


Figure 9. The overhead of an MQTT message compared to that of MODBUS.

6. Latency and CPU Usage

In order to support the results of Section 5, the performance of MQTT and the MODBUS TCP are compared in this section. This performance is based on the Round-Trip Time (RTT) measurements and the CPU usage. A Java-based MQTT publisher was developed and installed on a laptop that runs the operating system Linux Ubuntu 16.04. The specifications of the laptop were: Dell-Inspiron 3537, Intel® CoreTM i5-4200U CPU at 1.6 GHz × 4, 5.85 GiB RAM. The server (broker) was the Apache ActiveMQ software, which was installed on a desktop PC that runs the operating system Microsoft Windows XP. The desktop PC featured Pentium Dual Core CPU at 3.2 GHz and 2 GB RAM. The consumer was the MQTT.FX software, which was installed on the desktop PC as well. The RTT values measured from the

publisher and the CPU usage measured from the desktop PC are shown in Figure 10. The publisher that connected to the server (broker) started counting the RTT from the time of creating the socket to closing it, which includes the time needed for formatting the message in a publish message, connecting to the server without a logon process, publishing the message, and disconnecting from the server, in addition to the network latency. The CPU usage values are the maximum registered values, and the tests were conducted for payload sizes of 5, 10, 25, 50, and 100 bytes.



(**a**) (**b**)

Figure 10. MQTT-related measurements: (**a**) Round-Trip Time (RTT); (**b**) CPU usage.

The RTT measurements of the MODBUS TCP were also conducted using the same network. A MODBUS TCP client was developed and installed on the laptop, and a MODBUS TCP server was developed and installed on the desktop PC. Many tests of function codes were carried out for "read holding registers" (FC = 0x03), "read input registers" (FC = 0x04), and "write multiple registers" (FC = 0x10). The total transmitted bytes in each transaction were 27 bytes, 33 bytes, and 35 bytes respectively; the total was 95 bytes when all messages were transmitted within one transaction. The RTT values, which were measured on the client side, were between 6 ms and 8 ms, and the CPU usage, which was measured on the server side, had a maximum value of 3%.

These results confirm the theoretical outcomes of Section 5, which show that the MODBUS TCP is a lightweight protocol suitable for SCADA-like systems and industrial applications. The MODBUS TCP client connects to the MODBUS TCP server using polling communications (direct one-to-one communications). However, the MQTT client connects to another client via a server, called a broker. This fact explains the high CPU consumption while the MQTT server was busy in receiving and redirecting the publish messages.

7. Discussion

Based on the comparisons of Sections 4–6 as well as the theoretical study of the MODBUS TCP in Section 3, two scenarios are discussed here to build the industrial IoT environment.

The first scenario is to employ only the MODBUS TCP to build the IIoT environment. The study of Section 3 proved that MODBUS TCP is able to react as an IoT protocol. Security is a fundamental issue in IoT, which was solved in [32] for MODBUS TCP using TLS. As mentioned above in Section 5, the MODBUS TCP is able to build the IIoT environment using the synchronous request–response communication pattern on a standalone basis, which is a scenario that complies with most industrial applications. However, this solution totally relies on the polling-based mechanism.

In the second scenario, the event-based mechanism is fulfilled by MQTT, where M2M communications are required, using the asynchronous publish–subscribe communication pattern. For industrial functions, the MODBUS TCP is employed, which fulfills the synchronous request–response communication pattern. In this scenario, MQTT works in parallel with the MODBUS TCP within the same platform, as shown in Figure 11. This figure illustrates the simulation environment of the second scenario. The environment consisted of a desktop PC running Microsoft Windows XP, a laptop running Linux Ubuntu 16.04, and an LAN/Internet network. The desktop PC is equipped with a

java-based MODBUS TCP client, the ActiveMQ MQTT server, and the MQTT.FX consumer. The laptop was equipped with a multithreaded Java program; one thread ran the MQTT publisher, and another thread ran the MODBUS TCP server. In addition, an online MQTT broker and consumer, provided by HiveMQ for testing, were employed here as an Internet-based server.

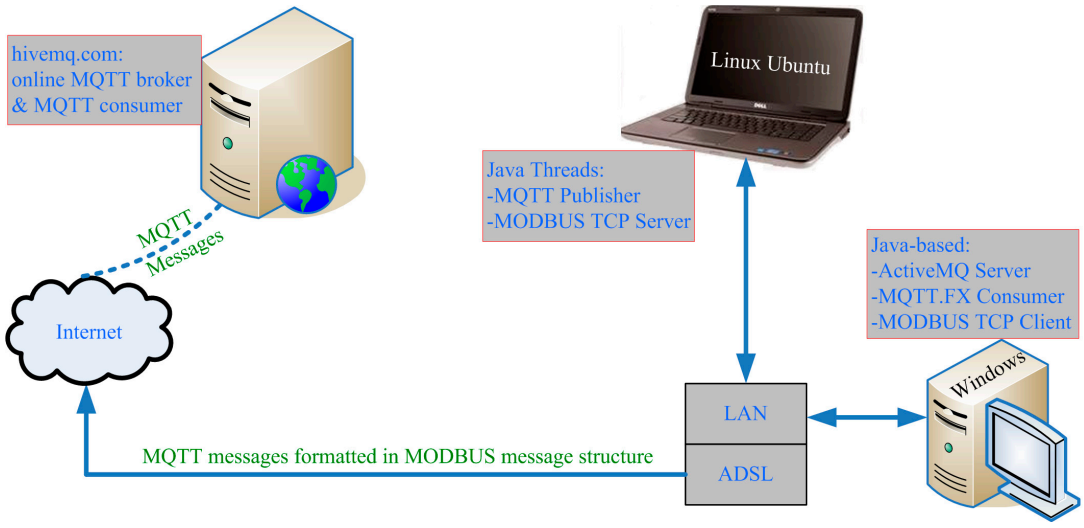


Figure 11. Simulation environment of the second scenario.

The RTT measurements and the CPU usage of this environment were measured for the MODBUS TCP and then compared to the previous results of Section 6. The RTT values of the MODBUS TCP, which were measured on the desktop PC, had a maximum value of 9 ms. This represents an increase of maximum 12.5% from the previous RTT values presented in Section 6. As illustrated in Figure 12, the CPU history of the laptop, which contains an MQTT publisher and a MODBUS TCP server, showed that the CPU usage is always less than 20%. These results indicate that the concurrent execution of MQTT in parallel with the MODBUS TCP within the same platform does not severely influence the performance of the MODBUS TCP. Nevertheless, while MQTT fulfills the event-based paradigm, it lacks interoperability.

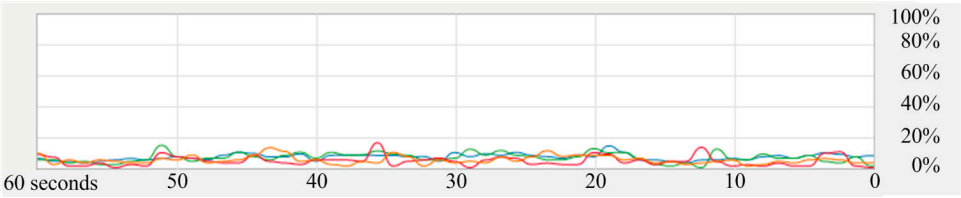
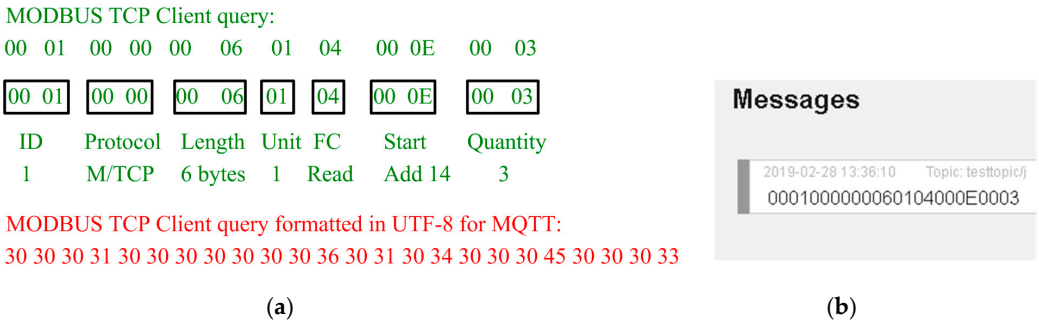


Figure 12. CPU history of the laptop that ran an MQTT publisher and MODBUS TCP server.

The MQTT protocol formats the data using the UTF-8 standard, and hence interoperability is not fulfilled. To maintain a homogeneous message for the entire environment, industrial data are organized using the structure of MODBUS messages, formatted in the UTF-8 and then transferred in the payload of an MQTT publish message, as shown in Figure 13a. The hexadecimal representations of Figure 13a were obtained based on the information of Table 3. Digit numbers zero to nine are formatted in UTF-8 using the hexadecimal representation. Digit number "0" is represented by "30" and digit number "9" is represented by "39". Additionally, letters "A" to "F" are represented by "41" to "46", respectively. Each byte of the MODBUS message is represented by two bytes, as shown in Figure 13. For example, the ID part of the MODBUS message is "00 01", which leads to "30 30 30 31" representation, and doubles the payload of the MQTT publish message. The MQTT publish

message was subsequently transmitted to the online hivemq.com server, and the result is illustrated in Figure 13b.

Using the arrangement of Figure 13a, the message structure, which is the MODBUS TCP, was maintained throughout the entire IIoT environment for both the control part and the monitoring part. This led to a unified message structure, unified data processing, and hence the interoperability problem was solved.



(**a**) (**b**)

Figure 13. (**a**) MODBUS message formatted via UTF-8 in the MQTT payload to solve the interoperability problem in the second scenario; (**b**) message as it appeared on hivemq.com consumer.

Table 3. UTF-8 encoding characters.

Character	UTF-8 (Decimal)	UTF-8 (Hex)	Meaning
0	48	0x30	Digit Zero
1	49	0x31	Digit One
2	50	0x32	Digit Two
3	51	0x33	Digit Three
4	52	0x34	Digit Four
5	53	0x35	Digit Five
6	54	0x36	Digit Six
7	55	0x37	Digit Seven
8	56	0x38	Digit Eight
9	57	0x39	Digit Nine
A	65	0x41	Capital Letter A
B	66	0x42	Capital Letter B
C	67	0x43	Capital Letter C
D	68	0x44	Capital Letter D
E	69	0x45	Capital Letter E
F	70	0x46	Capital Letter F

In conclusion, the choice of solution is totally dependent on the requirements of the industrial application. The MODBUS TCP builds the IIoT environment if publish–subscribe communications are not required, representing a secured solution that fulfills the requirements of most IIoT applications. If M2M communications are required, MQTT can be used, although it doubles the payload of the publish message in order to solve the interoperability problem.

8. Conclusions

Two scenarios are introduced to build the industrial IoT environment. The first scenario employs the MODBUS TCP only for synchronous polling communications; this solution complies with most industrial control systems and SCADA-like applications. However, if asynchronous event-based communications are required, MQTT complements MODBUS TCP operations. In this particular case, an industrial IoT environment requires the employment of at least two protocols—one for the IoT functions, mainly for M2M communications, and another for the industrial functions. MODBUS fulfills the industrial requirements, mainly the telecontrol, monitoring, and automation functions. MQTT works in parallel with the MODBUS TCP and complements its functions, but cannot replace MODBUS.

Since MQTT lacks interoperability, the industrial data are formatted using the structure of MODBUS messages, and then transferred in the payload of the MQTT publish message, which uses the format of the UTF-8. Thus, the message structure of the MODBUS TCP is maintained throughout the entire environment. This solution provides interoperability, but doubles the payload of the industrial data.

The simulation results and measurements, presented in Sections 6 and 7, show that the concurrent execution of MQTT in parallel with the MODBUS TCP within the same platform does not severely influence the performance of the MODBUS TCP.

Security is a fundamental issue in IoT, which was solved in [32] for MODBUS TCP using TLS via TCP port 802. MQTT may also employ TLS for encryption via port 8883. More information on the security of industrial IoT systems can be found in [53–55], on intrusion detection in [56–58], on encryption in [59], and on black-hole detection in [60].

Funding: This research received no external funding.

Conflicts of Interest: The author declares no conflict of interest.

Abbreviations

6LoWPAN	IPv6 over low-power wireless personal area network
AMQP	Advanced Message Queuing Protocol
CoAP	Constrained Application Protocol
EXI	Efficient XML Interchange
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Taskforce
IoT	Internet of Things
IIoT	Industrial Internet of Things
JSON	JavaScript Object Notation
M2M	Machine-to-Machine
MQTT	Message Queuing Telemetry Transport
OASIS	Organization for the Advancement of Structured Information Standards
SCADA	Supervisory Control and Data Acquisition
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UTF-8	Unicode Transformation Format—8-bit
XMPP	eXtensible Message and Presence Protocol

References

1. Zanella, A.; Bui, N.; Castellani, A.; Vangelista, L.; Zorzi, M. Internet of Things for Smart Cities. *IEEE Internet Things J.* **2014**, *1*, 22–32. [CrossRef]
2. Ahlgren, B.; Hidell, M.; Ngai, E.C.-H. Internet of Things for Smart Cities: Interoperability and Open Data. *IEEE Internet Comput.* **2016**, *20*, 2–56. [CrossRef]
3. Jaloudi, S. Software-Defined Radio for Modular Audio Mixers: Making Use of Market-Available Audio Consoles and Software-Defined Radio to Build Multiparty Audio-Mixing Systems. *IEEE Consum. Electron. Mag.* **2017**, *6*, 97–104. [CrossRef]
4. Watthanawisuth, N.; Maturos, T.; Sappat, A.; Tuantranont, A. The IoT wearable stretch sensor using 3D-Graphene foam. In Proceedings of the IEEE Conference on SENSORS, Busan, Korea, 1–4 November 2015. [CrossRef]
5. Chi, Q.; Yan, H.; Zhang, C.; Pang, Z.; Da Xu, L. A Reconfigurable Smart Sensor Interface for Industrial WSN in IoT Environment. *IEEE Trans. Ind. Inform.* **2014**, *10*, 1417–1425. [CrossRef]
6. El Kaed, C.; Khan, I.; Berg, A.V.D.; Hossayni, H.; Saint-Marcel, C. SRE: Semantic Rules Engine for the Industrial Internet-Of-Things Gateways. *IEEE Trans. Ind. Inform.* **2018**, *14*, 715–724. [CrossRef]
7. Iqbal, R.; Butt, T.; Shafiq, O.; Talib, M.; Umer, T. Context-Aware Data-Driven Intelligent Framework for Internet of Vehicles. *IEEE Access* **2018**, *6*, 58182–58194. [CrossRef]

8. Silva, R.; Iqbal, R. Ethical Implications of Social Internet of Vehicle Systems. *IEEE Internet Things J.* **2018**. [CrossRef]
9. Long, C.; Cao, Y.; Jiang, T.; Zhang, Q. Edge Computing Framework for Cooperative Video Processing in Multimedia IoT Systems. *IEEE Trans. Multimed.* **2018**, *20*, 1126–1139. [CrossRef]
10. Ja'afreh, M.A.; Aloqaily, M.; Ridhawi, I.A.; Mostafa, N. A hybrid-based 3D streaming framework for mobile devices over IoT environments. In Proceedings of the 3rd International Conference on Fog and Mobile Edge Computing (FMEC), Barcelona, Spain, 23–26 April 2018. [CrossRef]
11. Al Ridhawi, I.; Aloqaily, M.; Kotb, Y.; Al Ridhawi, Y.; Jararweh, Y. A collaborative mobile edge computing and user solution for service composition in 5G systems. *Wiley Trans. Emerg. Telecommun. Technol.* **2018**, *29*, e3446. [CrossRef]
12. Balasubramanian, V.; Aloqaily, M.; Zaman, F.; Jararweh, Y. Exploring Computing at the Edge: A Multi-Interface System Architecture Enabled Mobile Device Cloud. In Proceedings of the 7th International Conference on Cloud Networking (CloudNet), Tokyo, Japan, 22–24 October 2018. [CrossRef]
13. Jaloudi, S. Open source software of smart city protocols current status and challenges. In Proceedings of the International Conference on Open Source Software Computing (OSSCOM), Amman, Jordan, 10–13 September 2015. [CrossRef]
14. Standard 19464. *Advanced Message Queuing Protocol 1.0 (AMQP 1.0)*; ISO/IEC: Geneva, Switzerland, 2016.
15. O'Hara, J. ISO 19464 Connecting Business for Value. 2014. Available online: http://www.amqp.org/sites/amqp.org/files/2014.05.01%20ISO%2019464%20AMQP-ORG_0.pdf (accessed on 4 February 2019).
16. Godfrey, R.; Ingham, D.; Schloming, R. OASIS Standard Advanced Message Queuing Protocol (AMQP) Version 1.0. 2012. Available online: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf (accessed on 4 February 2019).
17. Standard PRF 20922. *Message Queuing and Telemetry Transport (MQTT) Version 3.1.1*; ISO/IEC: Geneva, Switzerland, 2016.
18. Standard RFC 7252. *Constrained Application Protocol (CoAP)*; IETF: Fremont, CA, USA, 2014.
19. Standard RFC 6120. *Extensible Message and Presence Protocol (XMPP)*; IETF: Fremont, CA, USA, 2011.
20. Standard RFC 7159. *The JavaScript Object Notation (JSON) Data Interchange Format*; IETF: Fremont, CA, USA, 2014.
21. Standard IEEE 802.11. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY)*; IEEE: New York, NY, USA, 2012.
22. Tao, F.; Cheng, J.; Qi, Q. IIHub: An Industrial Internet-of-Things Hub Toward Smart Manufacturing Based on Cyber-Physical System. *IEEE Trans. Ind. Inform.* **2018**, *14*, 2271–2280. [CrossRef]
23. Ferrari, P.; Flammini, A.; Rinaldi, S.; Sisinni, E.; Maffei, D.; Malara, M. Impact of Quality of Service on Cloud Based Industrial IoT Applications with OPC UA. *Electronics* **2018**, *7*, 109. [CrossRef]
24. Angrisani, L.; Cesaro, U.; D'Arco, M.; Grillo, D.; Tocchi, A. IOT Enabling Measurement Applications in Industry 4.0: Platform for Remote Programming ATES. In Proceedings of the IEEE Workshop on Metrology for Industry 4.0 and IoT, Brescia, Italy, 16–18 April 2018. [CrossRef]
25. Müller, J.M.; Kiel, D.; Voigt, K.-I. What Drives the Implementation of Industry 4.0? The Role of Opportunities and Challenges in the Context of Sustainability. *Sustainability* **2018**, *10*, 247. [CrossRef]
26. Sangkeun, Y.; Kim, Y.W.; Choi, H. An assessment framework for smart manufacturing. In Proceedings of the IEEE 20th International Conference on Advanced Communication Technology, Chuncheon-si Gangwon-do, Korea, 11–14 February 2018. [CrossRef]
27. Moyne, J.; Iskandar, J. Big Data Analytics for Smart Manufacturing: Case Studies in Semiconductor Manufacturing. *Processes* **2017**, *5*, 39. [CrossRef]
28. Chekired, D.A.; Khoukhi, L.; Mouftah, H.T. Industrial IoT Data Scheduling Based on Hierarchical Fog Computing: A Key for Enabling Smart Factory. *IEEE Trans. Ind. Inform.* **2018**, *14*, 4590–4602. [CrossRef]
29. Mabkhot, M.M.; Al-Ahmari, A.M.; Salah, B.; Alkhalefah, H. Requirements of the Smart Factory System: A Survey and Perspective. *Machines* **2018**, *6*, 23. [CrossRef]
30. Modbus Application Protocol Specification V1.1b3. 2012. Available online: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf (accessed on 3 February 2019).
31. Modbus Messaging on TCP/IP Implementation Guide V1.0b. 2006. Available online: http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf (accessed on 3 February 2019).

32. MODBUS/TCP Security. 2018. Available online: http://www.modbus.org/docs/MB-TCP-Security-v21_2018-07-24.pdf (accessed on 4 February 2019).
33. Meng, Z.; Wu, Z.; Muvianto, C.; Gray, J. A Data-Oriented M2M Messaging Mechanism for Industrial IoT Applications. *IEEE Internet Things J.* **2017**, *4*, 236–246. [CrossRef]
34. Brizzi, P.; Conzon, D.; Khaleel, H.; Tomasi, R.; Pastrone, C.; Spirito, A.M.; Knechtel, M.; Pramudianto, F.; Cultrona, P. Bringing the Internet of Things along the manufacturing line: A case study in controlling industrial robot and monitoring energy consumption remotely. In Proceedings of the IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), Cagliari, Italy, 10–13 September 2013. [CrossRef]
35. Chang, C.; Srirama, S.N.; Mass, J. A middleware for discovering proximity-based service-oriented Industrial Internet of Things. In Proceedings of the IEEE International Conference on Services Computing, New Your, NY, USA, 27 June–2 July 2015. [CrossRef]
36. Packwood, D.; Sharma, M.; Ding, D.; Park, H.; Salcic, Z.; Malik, A.; Kevin, I.; Wang, K. FPGA-based Mixed-Criticality Execution Platform for SystemJ and the Internet of Industrial Things. In Proceedings of the IEEE 18th International Symposium on Real-Time Distributed Computing, Auckland, New Zealand, 13–17 April 2015. [CrossRef]
37. Meng, Z.; Wu, Z.; Gray, J. A Collaboration-Oriented M2M Messaging Mechanism for the Collaborative Automation between Machines in Future Industrial Networks. *Sensors* **2017**, *17*, 2694. [CrossRef] [PubMed]
38. Calderón Godoy, A.J.; González Pérez, I. Integration of Sensor and Actuator Networks and the SCADA System to Promote the Migration of the Legacy Flexible Manufacturing System towards the Industry 4.0 Concept. *J. Sens. Actuator Netw.* **2018**, *7*, 23. [CrossRef]
39. Kruger, C.P.; Hancke, G.P. Implementing the Internet of Things vision in industrial wireless sensor networks. In Proceedings of the 12th IEEE International Conference on Industrial Informatics, Porto Alegre, Brazil, 27–30 July 2014. [CrossRef]
40. Hu, P. A System Architecture for Software-Defined Industrial Internet of Things. In Proceedings of the IEEE International Conference on Ubiquitous Wireless Broadband, Montreal, QC, Canada, 4–7 October 2015. [CrossRef]
41. Corotinschi, G.; Gitan, V.G. Enabling IoT connectivity for Modbus networks by using IoT edge gateways. In Proceedings of the IEEE International Conference on Development and Application Systems, Suceava, Romania, 24–26 May 2018. [CrossRef]
42. Joshi, R.; Jadav, H.M.; Mali, A.; Kulkarni, S.V. IOT application for real-time monitor of PLC data using EPICS. In Proceedings of the IEEE International Conference on Internet of Things and Applications, Pune, India, 22–24 January 2016. [CrossRef]
43. Trancă, D.-C.; Pălăcean, A.V.; Mihu, A.C.; Rosner, D. ZigBee based wireless modbus aggregator for intelligent industrial facilities. In Proceedings of the IEEE 25th Telecommunication Forum, Belgrade, Serbia, 21–22 November 2017. [CrossRef]
44. Shinde, K.S.; Bhagat, P.H. Industrial process monitoring using loT. In Proceedings of the IEEE International conference on IoT in Social, Mobile, Analytics and Cloud, Palladam, India, 10–11 February 2017. [CrossRef]
45. Standard IEEE 754. *Binary Floating-Point Arithmetic*; IEEE: New York, NY, USA, 2008.
46. Yokotani, T.; Sasaki, Y. Comparison with HTTP and MQTT on required network resources for IoT. In Proceedings of the IEEE International Conference on Control, Electronics, Renewable Energy and Communications, Bandung, Indonesia, 13–15 September 2016. [CrossRef]
47. Joshi, J.; Rajapriya, V.; Rahul, S.R.; Kumar, P.; Polepally, S.; Samineni, R.; Tej, D.K. Performance enhancement and IoT based monitoring for smart home. In Proceedings of the IEEE International Conference on Information Networking, Da Nang, Vietnam, 11–13 January 2017. [CrossRef]
48. Thota, P.; Kim, Y. Implementation and Comparison of M2M Protocols for Internet of Things. In Proceedings of the IEEE International Conference on ACIT-CSII-BCD, Las Vegas, NV, USA, 12–14 December 2016. [CrossRef]
49. Luzuriaga, J.E.; Perezy, M.; Boronaty, P.; Cano, J.C.; Calafate, C.; Manzoni, P. A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks. In Proceedings of the 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 9–12 January 2015. [CrossRef]
50. Gao, W.; Nguyeny, J.; Yu, W.; Lu, C.; Kuy, D.; Hatcher, W.G. Towards Emulation-Based Performance Assessment of Constrained Application Protocol (CoAP) in Dynamic Networks. *IEEE Internet Things J.* **2017**, *4*, 1597–1610. [CrossRef]

51. Koster, M.; Keranen, A.; Jimene, J. IETF Draft Standard Publish-Subscribe Broker for the Constrained Application Protocol (CoAP). 2019. Available online: https://tools.ietf.org/html/draft-ietf-core-coap-pubsub-06 (accessed on 4 February 2019).
52. Käbisch, S.; Peintner, D. W3C Recommendation Canonical EXI. 2018. Available online: https://www.w3.org/TR/exi-c14n/ (accessed on 4 February 2019).
53. Carías, J.F.; Labaka, L.; Sarriegi, J.M.; Hernantes, J. Defining a Cyber Resilience Investment Strategy in an Industrial Internet of Things Context. *Sensors* **2019**, *19*, 138. [CrossRef] [PubMed]
54. Kwon, S.; Jeong, J.; Shon, T. Toward Security Enhanced Provisioning in Industrial IoT Systems. *Sensors* **2018**, *18*, 4372. [CrossRef] [PubMed]
55. Xun, P.; Zhu, P.-D.; Hu, Y.-F.; Cui, P.-S.; Zhang, Y. Command Disaggregation Attack and Mitigation in Industrial Internet of Things. *Sensors* **2017**, *17*, 2408. [CrossRef] [PubMed]
56. Aloqaily, M.; Otoum, S.; Ridhawi, I.A.; Jararweh, Y. An Intrusion Detection System for Connected Vehicles in Smart Cities. *J. Ad Hoc Netw.* **2019**, in press. [CrossRef]
57. Otoum, S.; Kantarci, B.; Mouftah, H. Adaptively Supervised and Intrusion-Aware Data Aggregation for Wireless Sensor Clusters in Critical Infrastructures. In Proceedings of the IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018. [CrossRef]
58. Otoum, S.; Kantarci, B.; Mouftah, H.T. Detection of Known and Unknown Intrusive Sensor Behavior in Critical Applications. *IEEE Sens. Lett.* **2017**, *1*, 1–4. [CrossRef]
59. Wang, C.; Shen, J.; Liu, Q.; Ren, Y.; Li, T. A Novel Security Scheme Based on Instant Encrypted Transmission for Internet of Things. *Secur. Commun. Netw.* **2018**, *2018*, 3680851. [CrossRef]
60. Otoum, S.; Kantarci, B.; Mouftah, H.T. Hierarchical trust-based black-hole detection in WSN-based smart grid monitoring. In Proceedings of the IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017. [CrossRef]