mathematics

MDPI

Article

Enhancing the Modbus Communication Protocol to Minimize Acquisition Times Based on an STM32-Embedded Device

Ionel Zagan [1,2,*] and Vasile Gheorghiță Găitan [1,2]

[1] Faculty of Electrical Engineering and Computer Science, Stefan Cel Mare University, 720229 Suceava, Romania
[2] Integrated Center for Research, Development and Innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for Fabrication and Control (MANSiD), Stefan Cel Mare University, 720229 Suceava, Romania
* Correspondence: zagan@usm.ro

Abstract: The primary function of a distributed bus is to connect sensors, actuators, and control units that are used for an acquisition process. Application domains, such as industrial monitoring and control systems, manufacturing processes, or building automation, present different requirements that are not exactly invariable and coherent. Updating data from Modbus-type devices involves updating data through a technique called polling, which involves repeatedly scanning the registers from each device. This paper highlights the performance of Modbus communication, considering scenarios in which distributed devices are integrated and accessed registers are or are not at consecutive addresses. The Modbus protocol allows reading one or more holding-type data registers. If the registers are not at consecutive addresses, multiple requests are required, with implications for the real-time characteristics of the data acquisition system. We studied the data update times within the SMARTConvert application when variable numbers of registers are accessed, and we designed an extension for the Modbus protocol. The major reason Modbus is used in current research is that no assumptions are required about application semantics, and the performance/resource ratio for generic services is excellent.

Keywords: Modbus; remote terminal unit (RTU); acquisition cycle; industrial Internet of Things; communication

MSC: 68M12

Citation: Zagan, I.; Găitan, V.G. Enhancing the Modbus Communication Protocol to Minimize Acquisition Times Based on an STM32-Embedded Device. *Mathematics* **2022**, *10*, 4686. https://doi.org/10.3390/math10244686

Academic Editors: Mihai-Victor Micea, Alex Doboli, Daniel-Ioan Curiac and Cristina Sorina Stângaciu

Received: 25 October 2022
Accepted: 7 December 2022
Published: 10 December 2022

1. Introduction

In industrial information systems and automated manufacturing systems based on hierarchical models, one or more networks may be used at each control level. All devices on the same hierarchical level are connected to one or more networks with similar capabilities. As a result, the following levels of an automation process can be distinguished: wide area networks at the factory level, shop floor level, cell controller level, process level (programmable logic controller (PLC) and computer numerical control (CNC)), and field level (sensors and actuators). The latter type of network communicates between smart devices at the bottom (lowest) level of the hierarchy to control automated manufacturing systems [1]. Because these devices are located "in the field", i.e., close to the equipment they command, the network that interconnects them is often referred to as the field network (fieldbus). The networks in the last three levels presented above are called local industrial networks (LINs).

In the top-down path of the communication protocol layers, the application process data are augmented by the layer-specific data needed to execute those protocols. These data usually address and control information, which is mostly combined in a protocol header. Thus, the number of bits actually transmitted can be considerably higher than the user data provided by the application process, and the communication over-control can be substantial.

A classification of local networks at the enterprise level is presented below, with the first four types of networks being LINs [2]:

- The sensorbus is the network at the lowest level, generally used to connect simple, low-cost sensors, such as on/off switches. They transmit small amounts of data, requiring little computing power.
- The devicebus is the largest category of networks providing communication services for smart devices, which can perform multiple functions.
- The fieldbus generally is an advancement of the devicebus in that it supports the transmission of large amounts of data, but at a generally low speed, and requires more CPU power at the device level. Some fieldbus networks directly support the distribution of control functions to the end devices.
- The control bus primarily performs peer-to-peer communications between devices at high levels of control such as PLC or distributed control system (DCS) controllers.
- The enterprise bus is the basic network of the factory where administration is performed based on Ethernet Transmission Control Protocol/Internet Protocol (TCP/IP).

One reason to promote Ethernet at the field level is that, being the same network technology as used in the office world, direct integration is feasible, i.e., both the automation and office domains can generally be interconnected to a single network. However, an overview of the reality in the current automation industry shows that the requirements are different. Ethernet is a solution for the two lower open systems interconnection (OSI) layers and, as fieldbus history has shown, this is not enough. Even if the OSI stack commonly used with TCP-UDP-IP is considered, only the four lower layers are covered (Figure 1). Consequently, Ethernet or Internet technologies can be obtained in the field currently occupied by fieldbus systems, although they are actually used in practice [1]:

- Tuning a fieldbus protocol on UDP-TCP-IP;
- TCP-IP tuning on an existing bus;
- Define new real-time protocols;
- Limiting free average access in the Ethernet standard;
- Ethernet hardware changes to achieve better real-time capabilities.

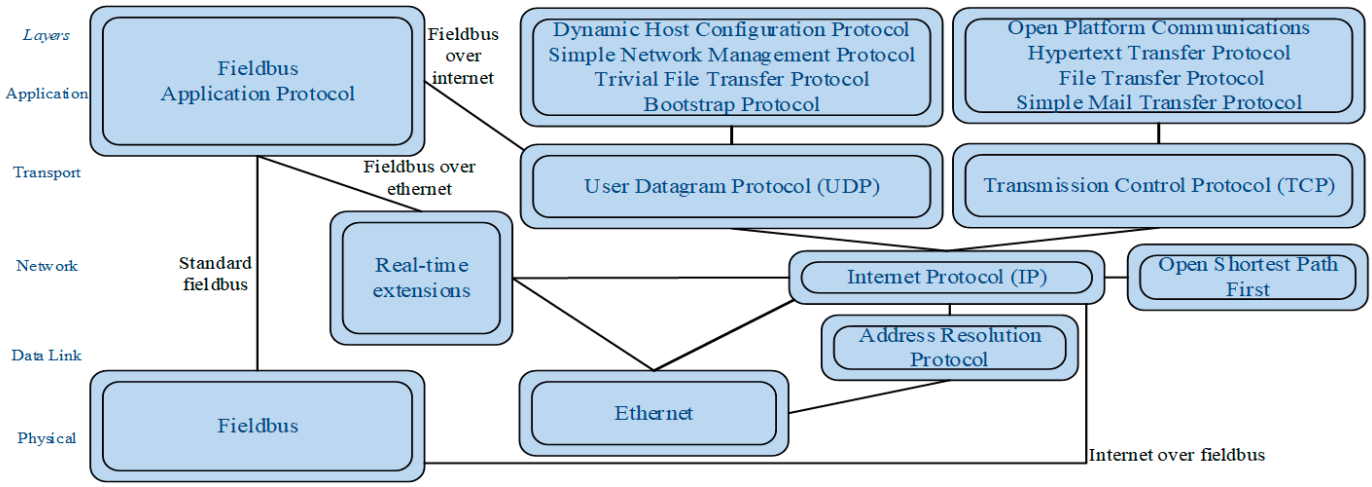


Figure 1. Structure of Ethernet and fieldbus in relation to protocol stack based on TCP/IP model.

The Modbus network protocol was designed in 1978 by Modicon Inc. It provided an efficient method to transmit control data between controllers and sensors using mainly an RS232 port. This protocol was then successfully introduced into automation installations, quickly becoming a de facto standard in the industrial segment. Thus, the Modbus protocol is today a unique protocol, being one of the main protocols among automation systems [2]. Schneider Electric has transferred Modbus and Modbus/TCP (Modbus over TCP/IP)

technical specifications [3,4] to Modbus.org. In terms of client/server communication between connected devices on different types of buses or networks, Modbus implements application-level messaging, a protocol placed at level 7 of the OSI stack. Modbus is currently transported using any of the following layers:

- RS485, RS422, RS232;
- TCP/IP;
- Modbus Plus peer-to-peer protocol, a token transmission network;
- A wide variety of physical transmission media (optical fiber and GSM).

Its main difference from the Distributed Network Protocol (DNP3), also known as IEEE Std 1815, is that Modbus is an application layer protocol, whereas DNP3 implements data link and application layers. Even though DNP3 and Modbus are polled protocols, to reduce bandwidth overhead, the DNP3 protocol is event-based. In terms of the physical layer, serial line Modbus systems can use different physical media such as RS485 or RS232. The two-wire interface is the most common in practice, implementing half duplex transmission. As an option, in full duplex mode, the four-wire RS485 interface is used. In the case of standard Modbus, all systems are interconnected in parallel on a three-conductor bus. Two conductors form a twisted pair over which data are bidirectionally transmitted, usually at a rate of 9600 bps.

The main contributions of our study related to mathematical sciences and embedded systems domain consist of implementing the structure of an acquisition cycle (AC) designed for incompletely defined networks by adding a time stamp and achieving broad temporal coherence. The derived contributions are based on studies related to data update times within the Modbus protocol and the definition of mathematical equations for the acquisition cycle analysis in the context of embedded systems mathematical models.

The remainder of this paper is organized as follows: Section 2 reviews previous Modbus-based projects. Section 3 describes the mathematical model for analyzing the performance of the Modbus communications acquisition cycle, and Section 4 validates the experimental results. Sections 5 and 6 present the discussion and conclusions, respectively.

2. Related Work

The Modbus industrial automation protocol is analyzed and implemented using Protege [5], a specific language for this field. Using Protege considerably facilitates protocol stack implementation and maintenance, and increases code sharing and reuse. The implementation successfully works together with a real integration device, fully complying with the Modbus standard. An important aspect of the proposed implementation is the memory footprint resulting from the implementation of the protocol package. Using the modular implementation concept, producing custom Modbus functionality subsets to achieve a minimum memory footprint is possible. A Modbus covert channel that can be used for remote command and control was introduced. The communication channel uses the least significant bits to store and transfer information between the client and server modules. Using this implementation, we aimed to test the bandwidth and stealth of some configurations [6].

In one study [7], the communication channel was based on a client–server architecture. Thus, information transfer was bidirectional, allowing users to upload commands and parameters to receive responses. This sender-oriented approach, which is a constraint of the Modbus architecture, implies a responsive command line for the user. Thus, the Modbus implementation presented in this study did not cover all possible variants of supervisory control and data acquisition (SCADA) communication channels.

One proposed solution [8] was successfully implemented and adopted for real equipment in the food industry. The researchers designed an innovative Modbus extension to overcome deficiencies such as address space size, bandwidth, and shared transfers. The choice of execution of communication protocols provided an efficient method to avoid unexpected synchronization interference with highly critical activities due to network inaccessibility. Many tasks having truly asynchronous access to the network would be

inappropriate. These devices have so far accumulated over 500,000 operating hours, without any protocol-related failures, confirming the correct behavior of the protocol. The characteristics of malicious traffic are defined in the Snort rules database, in which the tool automatically extracts the traffic features, places them in packages, and generates a Modbus malware package using Scapy [9]. To test the Modbus traffic generation tool, three Modbus Snort rules were added to the input file used by the tool to extract traffic characteristics. Accordingly, the tool reads the rules and generates malicious Modbus packages that trigger these rules in Snort. To confirm that the triggered rules are the same rules that were used by the tool, the alert SID and message can be compared in Snort with the SID and message of the sender-side rules. Authors [9] described in detail a new tool that can be useful in assessing the security of SCADA systems. The proposed instrument has the ability to generate malicious SCADA traffic of the Modbus protocol type, with the Modbus protocol being the most popular industrial protocol currently used. Malicious traffic is generated based on Snort rules that are used to trigger malicious packages. From the Snort rules, the proposed tool automatically extracts traffic functions, stores them in packages, and generates a Modbus malware package using Scapy.

The real-time performance of the Modbus TCP was experimentally evaluated, including both over-control and communication jitter [10]. To evaluate the additional introduced over-control and jitter, the authors performed two sets of experiments. In the first, based on Modbus TCP communication, they measured and analyzed round trip delay time. In the second set of experiments, they considered the direct TCP configuration that was used as the reference and presented several summary statistics. The authors performed all experiments by transferring randomly distributed, uniformly allocated recorded content between the client and server systems [10]. Both systems used a typical low-cost built-in microcontroller that was suitable for industrial applications, namely NXP LPC2468 [11], based on a 72 MHz ARM7 processor managed by the FreeRTOS real-time operating system. The results confirmed the high overall quality of Modbus TCP communication evaluated in experiments. However, its critical elements, such as the event system, did not introduce any significant jitter despite their non-negligible complexity. In another study [10], the real-time performance of Modbus TCP communication was experimentally evaluated, considering a direct streamed TCP connection. The detailed analysis showed that the mechanism for the recognition of the TCP segment substantially affected the general communication jitter, which compromised the application of the Modbus TCP protocol in real-time systems. Furthermore, the effects of the recognition mechanism on communication performance are not straightforward: they depend on many factors, such as system architecture, priority assignment, traffic initiator (e.g., client or server), and so on. For this reason, further analysis is required. However, a method to adjust the recognition mechanism by configuring the protocol stack to reduce the jitter introduced in the Modbus TCP communication needs to be designed so that real-time performance can be further improved.

Other authors [12] described the implementation of an experimental network and presented their practical results. From the various methods of error detection and correction, they chose to use the Reed–Solomon codes, which are systematic codes capable of correcting errors in isolated bits as well as impulse errors. The experimental time measurements showed that the maximum amount of redundant information that could be added to the network was four parity characters. Thus, the authors implemented an RS code (255, 251). Coding and decoding operations were performed by software, adapting the REED-SOLOMON encoding algorithms developed by Karn and Ortega, to execute them on a microcontroller in the dsPIC30F series. However, to reduce processing time and latency, future deployments can use the VHDL hardware description language.

Researchers [13] introduced and presented a test model for Modbus TCP protocol vulnerability based on an antisample algorithm. The reception rate was improved for test cases, successfully and time-efficiently detecting vulnerabilities in industrial control protocols. Experimental results [14] demonstrated that multichannel data acquisition could be used to accurately retrieve the corresponding data from a process; the embedded system

also implemented the communication task with a PC. Moreover, the proposed data acquisition system was feasible and stable in a lighting system, with notable energy savings. Security analysis involves protocol verification considering different types of attacks and basic cybersecurity services, such as integrity and confidentiality [15]. Current research has focused on how Modbus-RTU can be improved considering various aspects of communication performance [16]. The results obtained from simulations and experimental tests validated the effectiveness of the proposed method to increase the data rate while maintaining the reliability of the protocol. Researchers performed experimental validations demonstrating the effectiveness of the proposed method in isolating the segments susceptible to disturbances in a communication bus [17]. This maintained the compatibility with commercial devices and improved the performance of the entire network.

3. Modbus Protocol and Acquisition Cycle Analysis

Modbus has the advantage of wide acceptability between tool manufacturers and users, with many systems in operation. It is therefore a de facto industrial standard with proven capabilities. Certain features of the Modbus protocol are fixed, such as frame format, frame sequences, communication error handling, exception conditions, and performed functions. Other features can be selected, such as the transmission medium, characteristics of the communication channel, and transmission mode (RTU or ASCII). User features are set on each device and cannot be changed when the system is running. The two transmission modes in which data are exchanged are as follows [18]:

- ASCII: Easy to read, for example, in the testing stage (ASCII format);
- RTU: Compact and fast, being used for normal operation (hexadecimal format).

Traditional Modbus messages can be placed in RTU or ASCII frames and can be packaged with a TCP/IP interface, so they can be sent over the internet. Modbus RTU is the most common implementation available for Modbus. Modbus RTU is simply Modbus over serial communication media such as RS485, RS422, and RS232. Modbus TCP/IP is basically the Modbus RTU protocol with a TCP interface that runs on Ethernet. It does not require a checksum calculation, as lower layers already provide checksum protection. It defines the rules for organizing and interpreting the data independent of the data transmission medium. Modbus TCP/IP has not been extensively addressed because it was not included in this project. Modbus protocol messaging services based on the client–server model can be successfully used for real-time information exchange [2]:

- Between two devices;
- For data exchange between device applications and other devices;
- Between human–machine interface (HMI)/SCADA applications and devices;
- Between a PC and a device program that provides online services.

The Modbus header is 7 bytes long, with the detailed meaning of the fields being as follows. The Modbus Application Protocol (MBAP) header contains the following fields:

- Transaction identifier: used to pair transactions. The Modbus server copies the transaction identifier from the request message to the reply message;
- Protocol identifier: used for the intrasystem multiplexing mechanism. The Modbus protocol identifier is zero;
- Length: The length field is a byte counter of the following fields, including the unit identifier and data fields;
- Unit identifier: This field is used within a system and serves as routing. It is typically used to communicate with server stations on the Modbus and Modbus + serial lines through an access gate (gateway) between TCP-IP Ethernet networks and the Modbus serial line. The field is set by the Modbus client in the request message and is required to be returned with the same value in the response message from the server.

3.1. Laboratory Model for Analysis of Communication Performance Based on Modbus Protocol

The Modbus protocol performs a broadcast communication (unconfirmed, valid only for the address zero, and a unicast one (confirmed) for the other user addresses, 1,..., 247). A schematic diagram of the experimental laboratory system including the Modbus protocol is shown in Figure 2. For tests, we implemented an emulation application on an STM32F7 32 bit ARM microcontroller system (for a large number of stations). This allowed the use of time stamps and markers that could be viewed on an oscilloscope, resulting in a much finer measure of the times involved in the values update periods on the device. The requirements assume that the tests are performed for a maximum of 50 addresses of holding-type registers. Figure 2 shows the experimental architecture of the Modbus system. For the practical laboratory tests the STM32F7 microcontroller was used to emulate the server devices communicating via Modbus protocol. In this paper, the delays corresponding to the Modbus communication protocol were measured in a PC (client)–STM32F7 (servers) configuration by emulating several configurations of server stations on the microcontroller, and at the same time sending different messages each with a different number of registers on Modbus. At the server level, where we connected Modbus RTU workstations to the bus, three components were defined:

1. The application, specific to each workstation;
2. Real-time operating system (RTOS) or sequencer (in the case of tests, we used an RTOS);
3. Serial communication driver.

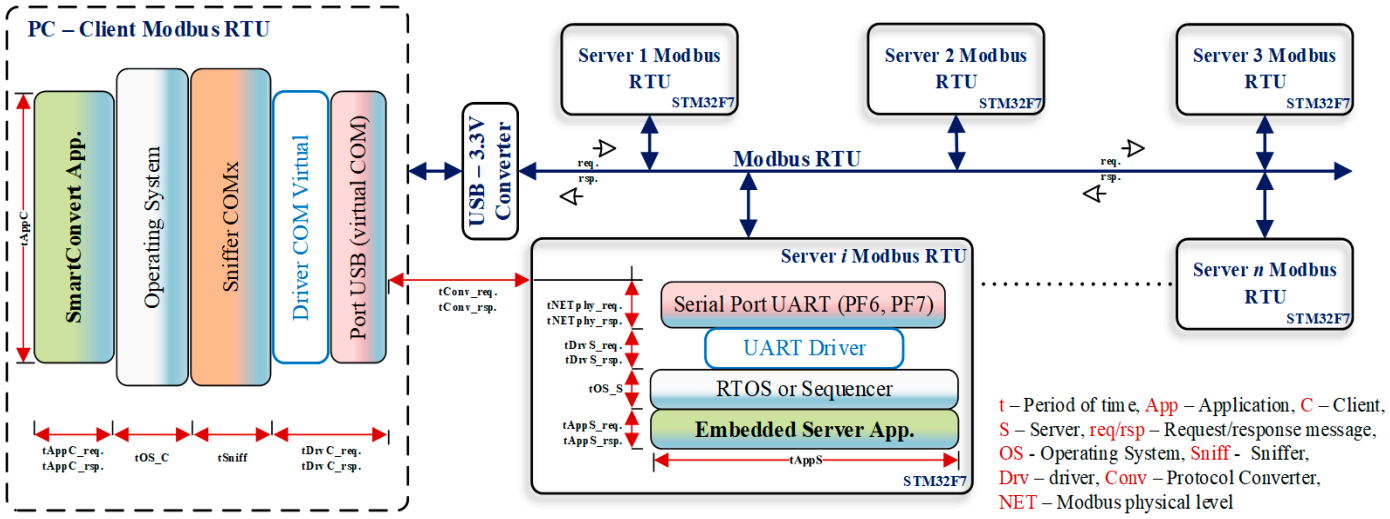


Figure 2. Experimental architecture of the Modbus client–server system.

TCP/IP uses a dedicated header to identify Modbus application data units (ADUs). This is called the Modbus application Protocol Header (MBAP). The header includes control information (addresses, message length, message type, etc.) that is placed in front of the message transmitted over the network [19]. This header is different from the Modbus RTU–ADU, which is used on serial communication lines, in the following respects [20]:

- The field containing server address used by the serial Modbus is replaced by a single byte called the unit identifier, which is found in the MBAP. The unit identifier octet is used to communicate through devices such as bridges, routers, and gateways, which use a single IP address to support multiple Modbus endpoints.
- All Modbus requests and responses are designed so that the receiver can verify the end of the message. For function codes where the Modbus protocol data unit (PDU) length is fixed, the function code is sufficient. For function codes carrying a variable amount of data, in the request or response, the data field includes a byte counter.

- When Modbus is transmitted on TCP/IP, the MBAP header contains additional length information to allow the receiver to recognize the message boundaries, even if they have been divided into multiple packets for transmission. The existence of implicit or explicit rules on length and use of cyclic redundancy check (CRC-32) by Ethernet leads to a low chance of undetected corruption of demand or response messages.

The control functions used in laboratory tests are FC03 (0x03) and FC16 (0x10). Modbus function FC03 reads the holding register and FC16 writes multiple registers. Other commonly used Modbus functions are FC06 (write single register), FC22 (mask write register), and FC23 (read/write multiple registers).

Memory is allocated at the byte level using the area fc03_register (Figure 3). The Modbus client provides an interface that allows the user application to build requests for various Modbus services, including access to the application objects. The Modbus client application programming interface (API) is not part of the Modbus messaging on the TCP/IP Implementation Guide V1.0b specification, although it contains an example described in the implementation model [21].

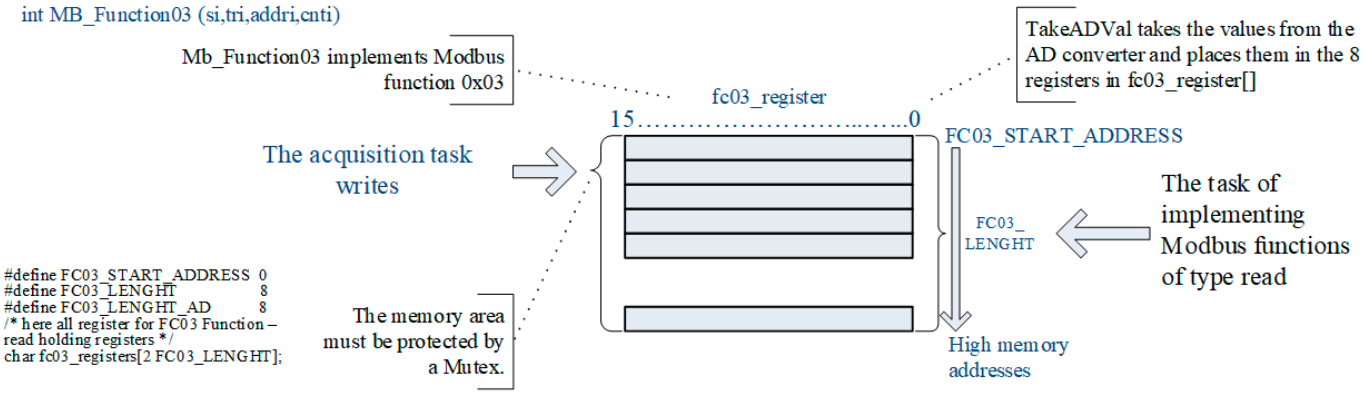


Figure 3. Registers attached to Modbus function MB_Function03.

With regard to the Modbus server, upon receipt of a Modbus request, this module activates a local function to read, write, or perform other actions. The function processing is performed in full transparency toward the application program. The main function of the Modbus server is to wait for Modbus requests at TCP port 502, treat the request, and then build a Modbus response according to the device context.

The example presented in Table 1 is a reading request using FC03, where the contents of registers 80, 81 and 82 are 0x01FF, 0x55EF, and 0x00DF, respectively.

Table 1. Example for reading function FC03.

Request		Response	
Field Name	**(Hex)**	**Field Name**	**(Hex)**
Function	03	Function	03
Upper start address (Hi)	00	Byte counter	06
Lower starting address (Lo)	4A	Register value Hi (92)	01
Registers number Hi	00	Register value Lo (92)	FF
Registers number Lo	03	Register value Hi (93)	55
		Register value Lo (93)	EF
		Register value Hi (94)	00
		Register value Lo (94)	DF

3.2. Mathematical Equations for Modbus Acquisition Cycle Analysis

Modbus performance is influenced by the limitations imposed by serial links EIA-232/485 [22]. Designers share these data with multiple operator workstations, control systems, and all other potential users of the data, generating complex hierarchies of client–server groups. Thus, even expensive DCSs waste valuable time mixing data for the benefit of higher hierarchical levels. The following equations represent the times required for the FC16 function in cases where the message is transmitted correctly or with an error. The parameter α can take values between 0 and 1, depending on the successive character transmission rate on Modbus. If characters are sent one after the other, α is 0. If a delay occurs between the transmission of two successive characters, α is different from 0, and becomes 1 if the distance between two successive characters is 1.5 characters, as specified in the Modbus standard. Parameters $\alpha 1$ and $\alpha 2$ are proposed for FC03 and FC16 because, for Modbus functions, message averages may differ.

$$\begin{aligned} \text{TreqF16} = {} & \text{h} + \text{data} + \text{CRC} \\ & = (1\ \text{addr} + 1\ \text{function code} + 2\ \text{starting address} \\ & + 2\ \text{amount of registers} + 1\ \text{byte counter}) + 2 \times \text{nreg} + 2 \\ & = 9 + 2 \times \text{nreg}\ (\text{frames}) \end{aligned} \tag{1}$$

$$\begin{aligned} \text{TrspF16} = {} & \text{TrspOkF16} \\ & = (1\ \text{addr} + 1\ \text{function code} + 2\ \text{starting address} \\ & + 2\ \text{amount of registers}) + \text{CRC} = 8\ (\text{frames}) \end{aligned} \tag{2}$$

$$\text{TF16OK} = (17 + \text{nreg} \times 2) \times \text{lframe}\ (\text{bits}) \tag{3}$$

If the response is an error then:

$$\begin{aligned} \text{TrspF16} = {} & \text{TrspErrF03} \\ & = (1\ \text{addr} + 1\ \text{error code} + 1\ \text{exception code}) + 0 \\ & + \text{CRC} = 5 \end{aligned} \tag{4}$$

$$\text{tTF03OK} = (20 + \text{nreg} \times (2 + 3 \times \alpha) + 18 \times \alpha) \times \text{lframe} \times \text{tbit} \tag{5}$$

$$\text{tTF03ERR} = (20 + 18 \times \alpha) \times \text{lframe} \times \text{tbit} \tag{6}$$

$$\text{tTF16OK} = (24 + \text{nreg} \times (2 + 3 \times \alpha) + 24 \times \alpha) \times \text{lframe} \times \text{tbit} \tag{7}$$

$$\text{tTF16ERR} = (21 + \text{nreg} \times (2 + 3 \times \alpha) + 19.5 \times \alpha) \times \text{lframe} \times \text{tbit} \tag{8}$$

If α = 0 then the transaction is shorter.

$$\text{tTF03OK} = (38 + 5 \times \text{nreg}) \times \text{lframe} \times \text{tbit} \tag{9}$$

$$\text{tTF03ERR} = 38 \times \text{lframe} \times \text{tbit} \tag{10}$$

$$\text{tTF16OK} = (48 + 5 \times \text{nreg}) \times \text{lframe} \times \text{tbit} \tag{11}$$

$$\text{tTF16ERR} = (40.5 + 5 \times \text{nreg}) \times \text{lframe} \times \text{tbit} \tag{12}$$

tHWnet hardware delays (line drivers, signal propagation on long lines, etc.) can also be added to the given transaction time. However, these delays are considered at high communication speeds, but can be insignificant at 9600 bps. The acquisition cycle time mainly depends on the Modbus control function and the number of characters, which can be higher or lower on reception or emission depending on the control function (Figure 2). The time can also depend on the size of the message expressed by nreg (the number of registers).

3.3. Customizing Mathematical Relationships

Industrial processes are often classified according to different paradigms, namely continuous systems and discrete event systems. In this context, an example of a continuous process is a controller that controls the temperature in an environment within set limits. Such a system must read the temperature from a sensor, compare it with the reference value, calculate the required correction using a control algorithm, and finally control a heating actuator. These operations occur at periodic intervals, mainly determined by the dynamics of the controlled process. An important feature of discrete event systems is that although the action triggered by an event is known, the time at which this event occurs is unknown. However, the order in which two or more events occur is, in principle, important.

We considered a case with a single read for all 10 registers on the server (thermostat) with address 1 and a single write for the 10 registers on the server with address 2 (the most favorable case in terms of the address of the fan coil). For this case, a set of acquisition slots S_AC (adr, FCxx, adrReg, and nreg) for reading and writing can be expressed by {(1,3,7,10), (2,16,7,10)}.

The Modbus backend interface is the interface from the Modbus server to the user application where the application objects are defined. Regarding the management of the TCP layer, the solution adopted for the rest of the system components is transparent to the user application, so that Berkeley Standard Distribution (BSD) sockets are used. Despite being less flexible, using BSD sockets is more productive. The access control module plays the role, when necessary, of ensuring security elements regarding access to the internal data of the devices. For resource management and data flow control, to balance the data flow of incoming and outgoing messages between the Modbus client and server, flow control mechanisms are required at all levels of the Modbus message stack. This module is primarily based on transport layer (TCP) flow control, to which a data link layer and application layer flow control are added. The address and register's address values do not affect the S_AC in terms of acquisition cycle time (tAC). The acquisition cycle time for S_AC defined can thus be calculated as follows:

$$\begin{aligned} \text{tAC} = \text{tClient1}(\text{adr1}, \text{FC03}, \text{nreg1}) &+ \text{tServer1}(\text{adr1}, \text{FC03}, \text{nreg1}) \\ &+ \text{tTF031}(\text{FC03}, \alpha 1, \text{err1}, \text{tout}, \text{nreg1}) + \text{tHWnet1} \\ &+ \text{tadjust1} + \text{tClient2}(\text{adr2}, \text{FC16}, \text{nreg2}) \\ &+ \text{tServer2}(\text{adr2}, \text{FC16}, \text{nreg2}) \\ &+ \text{tTF162}(\text{FC16}, \alpha 2, \text{err2}, \text{tout}, \text{nreg2}) + \text{tHWnet2} \\ &+ \text{tadjust2} \end{aligned} \tag{13}$$

If adr1 = 1, adr2 = 2, nreg1 = nreg2 = 10; then the time for the acquisition cycle is:

$$\begin{aligned} \text{tAC} = \text{tClient1}(1, \text{FC03}, 10) &+ \text{tServer1}(1, \text{FC03}, 10) \\ &+ \text{tTF031}(\text{FC03}, \alpha 1, \text{err1}, \text{tout}, 10) + \text{tHWnet1} + \text{tadjust1} \\ &+ \text{tClient2}(2, \text{FC16}, 10) + \text{tServer2}(2, \text{FC16}, 10) \\ &+ \text{tTF162}(\text{FC16}, \alpha 2, \text{err2}, \text{tout}, 10) + \text{tHWnet2} + \text{tadjust2} \end{aligned} \tag{14}$$

$$\begin{aligned} \text{tAC} = \text{tAll1}(1, \text{FC03}, 10) &+ \text{tTF031}(\text{FC03}, \alpha 1, \text{err1}, \text{tout}, 10) \\ &+ \text{tAll2}(2, \text{FC16}, 10) + \text{tTF16\_2}(\text{FC16}, \alpha 2, \text{err2}, \text{tout}, 10) \end{aligned} \tag{15}$$

where

$$\begin{aligned} \text{tAll1}((1, \text{FC03}, 10) &= \text{tClient1}(\text{adr1}, \text{FC03}, \text{nreg1}) \\ &+ \text{tServer1}(\text{adr1}, \text{FC03}, \text{nreg1}) + \text{tHWnet1} + \text{tadjust1} \end{aligned} \tag{16}$$

$$\begin{aligned} \text{tAll2}((2, \text{FC16}, 10) \\ = \text{tClient2}(\text{adr2}, \text{FC16}, \text{nreg2}) \\ + \text{tServer2}(\text{adr2}, \text{FC16}, \text{nreg2}) + \text{tHWnet2} + \text{tadjust2} \end{aligned} \tag{17}$$

Considering lframe = 10, tbit = 1/9600 = 1.041(6) e−4 = 104.17 us from the function definition, it follows that there are the following three cases:

(a) Modbus message is transmitted correctly:

$$\begin{aligned} \text{tTF031}(\text{FC03}, \alpha 1, 0, 0, 10) \\ = (20 + \text{nreg1} \times (2 + 3 \times \alpha 1) \\ + 18 \times \alpha 1) \times \text{lframe} \times \text{tbit} \\ = 41.668 + 50.0016 \times \alpha 1 \text{ (ms)} \end{aligned} \tag{18}$$

$$\begin{aligned} \text{tTF162}(\text{FC16}, \alpha 2, 0, 0, 10) \\ = (24 + \text{nreg2} \times (2 + 3 \times \alpha 2)2 \\ + 24 \times \alpha 2) \times \text{lframe} \times \text{tbit} \\ = 45.8348 + 56.2518 \times \alpha 2 \text{ (ms)} \end{aligned} \tag{19}$$

If $\alpha 1 = 1$, $\alpha 2 = 1$, err1 = 0, err2 = 0 and tout = 0 then tTF031(FC03.1,0,0,10) = 91.669 ms and tTF162(FC16,1,0,0,10) = 102.086 ms.

$$\text{tAC} = \text{tAll1}((1, \text{FC03}, 10) + \text{tAll2}((2, \text{FC16}, 10) + 193.775 \text{ ms} \tag{20}$$

If $\alpha 1 = 0$, $\alpha 2 = 0$, err1 = 0, err2 = 0 and tout = 0 then tTF031(FC03.0,0,0,10) = 41.668 ms and tTF162(FC16,0,0,0,10) = 4545.834 ms.

$$\text{tAC} = \text{tAll1}(1, \text{FC03}, 10) + \text{tAll2}(2, \text{FC16}, 10) + 87.502 \text{ ms} \tag{21}$$

(b) Modbus message is sent with error:

$$\begin{aligned} \text{tTF031}(\text{FC03}, \alpha 1, 1, 0, x) = \text{tTF03ERR} = (20 + 18 \times \alpha 1) \times \text{lframe} \times \text{tbit} \\ = 20834 + 18750.6 \times \alpha 1 \end{aligned} \tag{22}$$

$$\begin{aligned} \text{tTF162}(\text{FC16}, \alpha 2, 1, 0, x) = \text{tTF16ERR} \\ = (21 + \text{nreg2} \times (2 + 3 \times \alpha 2) \\ + 19,5 \times \alpha 2) \times \text{lframe} \times \text{tbit} \\ = 42709.7 + 51564.15 \times \alpha 2 \end{aligned} \tag{23}$$

If $\alpha 1 = 1$, $\alpha 2 = 1$, err1 = 1, err2 = 1 and tout = 0 then tTF031(FC03.1,1,0,x) = 39.5846 ms and tTF162(FC16.1,1,0,x) = 94.27385 ms.

$$\text{tAC} = \text{tAll1}(1, \text{FC03}, 10) + \text{tAll2}(2, \text{FC16}, 10) + 133.85845 \text{ ms} \tag{24}$$

If $\alpha 1 = 0$, $\alpha 2 = 0$, err1 = 1, err2 = 1 and tout = 0 then tTF031(FC03.0,1,0,x) = 20.834 ms and tTF162(FC16,0,1,0,x) = 42.7097 ms.

$$\text{tAC} = \text{tAll1}(1, \text{FC03}, 10) + \text{tAll2}(2, \text{FC16}, 10) + 63.5437 \text{ ms} \tag{25}$$

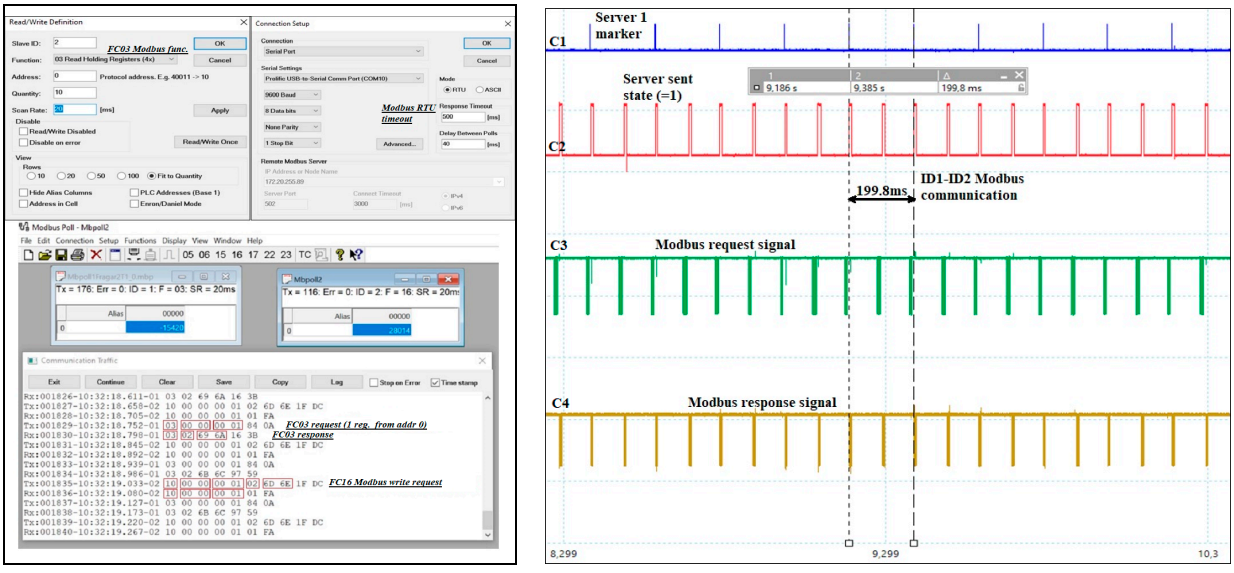
(c) A timeout event is recorded: For timeout (tout) the delay can be about 500 ms at the client application level (tAC = tAll1 (1, FC03, 10) + tAll2 (2, FC16, 10) + tout).

4. Experimental Results

In this section, we present our laboratory experimental results from testing the message exchange between devices belonging to the same Modbus communication protocol based

on the SMARTConvert platform. In Section 3, we presented the theoretical relationships given by the delay of the Modbus transactions on a RS485 bus. Devices connected on the Modbus RS485 bus must have the same communication speed and know the Modbus protocol. We used a USB–3.3 V adapter and a virtual COM on a PC for testing. As the Modbus stations did not meet the hardware requirements in this study, we designed an application on an STM32F7 Discovery development kit [23] that simulated 247 stations, each with 50 holding registers. In this study, one of the stations could support the type of test performed (we designed the application for the situation where the use of all addresses for the user, from 1 to 247, was required). For these 50 registers available at the station level (server), the required Modbus functions, namely, FC03 for reading and FC16 for writing, were available.

We connected the PC and development kit with a USB–3.3 V converter (on the serial port) identified as a Prolific USB-to-Serial-Comm Port (COM10). Figure 4a shows the Modbus poll and two server stations with addresses 0x01 and 0x02. One register was read from server ID = 0x01 using the FC03 Modbus function. The register was written to the server with ID = 0x02, at memory address 0x00, with the related time stamps. In terms of acquisition cycle experimental data, the cursors delimited a two-server acquisition cycle. We connected four probes (Ci signals in this paper) of the oscilloscope as follows (Figure 4b): probe 1 (C1) to an output pin of the microcontroller (pin PF8) that indicated the Server 1 station marker, probe 2 (C2) to an output pin of the microcontroller (pin PF9) as a spy to indicate the send/receive status of Server i, and probes 3 (C3) and 4 (C4) to UART (RX, pin PF6; TX, pin PF7). In the laboratory experiment, we used RX and TX lines for testing, which we directly connected between two client–server devices, a USB–3.3 V adapter, and a server station emulator (up to 256 stations). We implemented a half-duplex connection, so we used no RTS/CTS signals or line drivers to control their direction. The STM32F7 microcontroller automatically used RTS to switch the communication direction, even if a line driver was used. The measured values for the period of an acquisition cycle with two Modbus RTU server stations were as follows: 199.8, 200.5, 200.7, 200.2, and 196.9 ms. The experimental results showed that the time period related to the server with address 1 was between 196.9 and 200.7 ms.

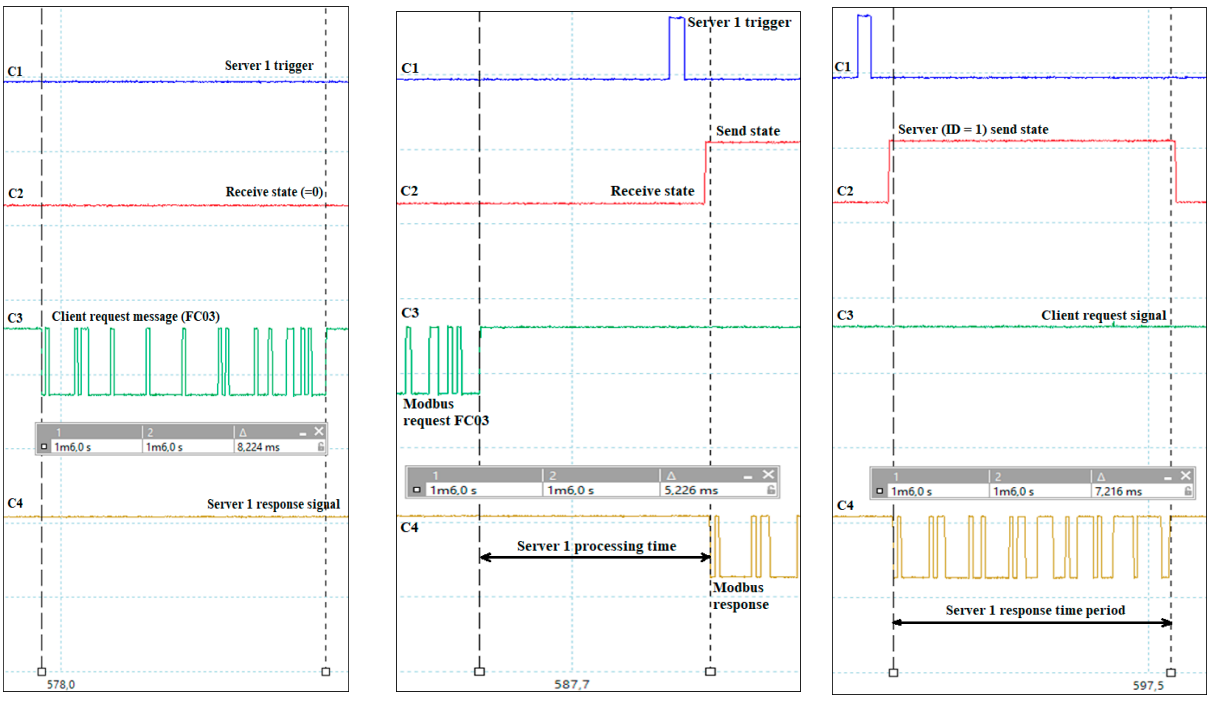


(**a**) (**b**)

Figure 4. Modbus protocol performance analysis: (**a**) Modbus connection parameterization, FC03 and FC16 function definitions and Modbus Poll interface (read/write Modbus function (0x03/0x10), start address (0x00, 0x00), number of registers (0x00, 0x01), byte counter 0x02, register data (0x69, 0x6A/0x6D, and 0x6E), and CRC); (**b**) experimental data for Modbus acquisition cycles analyzed in experimental laboratory project (39 servers) using PICOSCOPE 6404D (Pico Technology).

During this period, the server driver switched from reception to broadcast and sent the reply message to the client. In Figure 4, the red cursor (C2) indicates that the communication driver was switched at server i from reception to broadcast. The green cursor (C3) represents the request message at the USB converter–TTL (PC client) level. The brown cursor (C4) represents the server response message (hardware simulator device). The values recorded for acquisition cycles with one register were as follows: 196.9, 200.2, 200.5, and 200.7 ms. Therefore, we observed a maximum difference of 3.8 ms between the presented acquisitions cycles. The times between two consecutive requests from the server with ID 1 (FC03) to the server with ID 2 (FC16) were 97.76, 97.62, 100.2, and 100.6 ms. The experimental data validated a maximum difference of 2.98 ms between the obtained results. The time intervals between two consecutive requests from the ID 2 (FC16) server to the ID 1 (FC03) server following the measurements were 94.25, 99.34, 100.1, and 99.94 ms, with a jitter of only 5.85 ms.

As shown in Figure 5, for the FC03 read function addressed to the server with ID 0x01, the query message period was between 8.224 and 8.251 ms, with a difference between them of only 27 µs. The server processing time was only 5.2 ms, with the experimental values measured as follows: 5.12, 5.226, 5.412, and 5.598 ms. The jitter recorded in the laboratory tests was only 478 µs. The transmission periods for the ID 0x01 server response message, characteristic of the FC03 function, were 7.19 and 7.216 ms. Table 2 defines the transaction time mathematical functions based on different parameters and Modbus functions. Discrete event-based systems implemented using cyclic or periodic queries are called time-triggered systems or data-sampling systems. Implementations of discrete event systems using interrupts or internal software events are called event-triggered systems. The choice of a communication system for interconnecting industrial control applications largely depends on the application type and constraints imposed.



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

Figure 5. Modbus protocol experimental tests: (**a**) server query message timing with ID 0x01 and function FC03; (**b**) time required for processing at server with ID = 0x01 and FC03 (analysis of request and sending response; forced delay = 1 ms); (**c**) time required for response message of server with ID 0x01 and function FC03.

Table 2. Modbus transaction-timing-based tTFxx (FCxx, α, err, tout, and nreg) function.

Condition	Result Obtained
If FCxx = FC03, α ∈ (0,1), err = 0, tout = 0	$tTF03(FC03, \alpha, 0, 0, nreg) = tTF03OK = (20 + nreg \times (2 + 3 \times \alpha) + 18 \times \alpha) \times lframe \times tbit$
If FCxx = FC03, α ∈ (0,1), err = 1, tout = 0	$tTF03(FC03, \alpha, 1, 0, x) = tTF03ERR = (20 + 18 \times \alpha) \times lframe \times tbit$
If FCxx = FC03, α = 0, err = 0, tout = 0	$tTF03(FC03, 0, 0, 0, nreg) = tTF03OK = (13 + nreg \times 2) \times lframe \times tbit$
If FCxx = FC03, α = 0, err = 1, tout = 0	$tTF03(FC03, 0, 0, 0, x) = tTF03ERR = 13 \times lframe \times tbit$
If FCxx = FC03, α = 1, err = 0, tout = 0	$tTF03(FC03, 1, 0, 0, nreg) = tTF03OK = (38 + 5 \times nreg) \times lframe \times tbit$
If FCxx = FC03, α = 1, err = 1, tout = 0	$tTF03(FC03, 1, 0, 0, x) = tTF03ERR = 38 \times lframe \times tbit$
If FCxx = FC03, α = x, err = x, tout = TOUT	tTF03(xx,x,x,x,TOUT) = TOUT
If FCxx = FC16, α ∈ (0,1), err = 0, tout = 0	$tTF16(FC16, \alpha, 0, 0, nreg) = tTF16OK = (24 + nreg \times (2 + 3 \times \alpha) + 24 \times \alpha) lframe \times tbit$
If FCxx = FC16, α ∈ (0,1), err =1, tout = 0	$tTF16(FC16, \alpha, 1, 0, x) = tTF16ERR = (21 + nreg \times (2 + 3 \times \alpha) + 19,5 \times \alpha) \times lframe \times tbit$
If FCxx = FC16, α = 0, err = 0, tout = 0	$tTF16(FC16, 0, 0, 0, nreg) = tTF16OK = (24 + nreg \times 2) \times lframe \times tbit$
If FCxx = FC16, α = 0, err = 1, tout = 0	$tTF16(FC16, 0, 0, 0, x) = tTF16ERR = 21 + 2 \times nreg \times lframe \times tbit$
If FCxx = FC16, α = 1, err = 0, tout = 0	$tTF03(FC16, 1, 0, 0, nreg) = tTF16OK = (48 + 5 \times nreg) \times lframe \times tbit$
If FCxx = FC16, α = 0, err = 1, tout = 0	$tTF16(FC16, 1, 0, 0, x) = tTF16ERR = (21 + 2 \times nreg) \times lframe \times tbit$
If FCxx = FC16, α = x, err = x, tout = TOUT	tTF16(xx,x,x,x,TOUT) = TOUT

Figure 6 shows the experimental data for the acquisition cycle and the updating registers with FC16. The write function was characterized by longer transmission periods than the FC03 reading function.

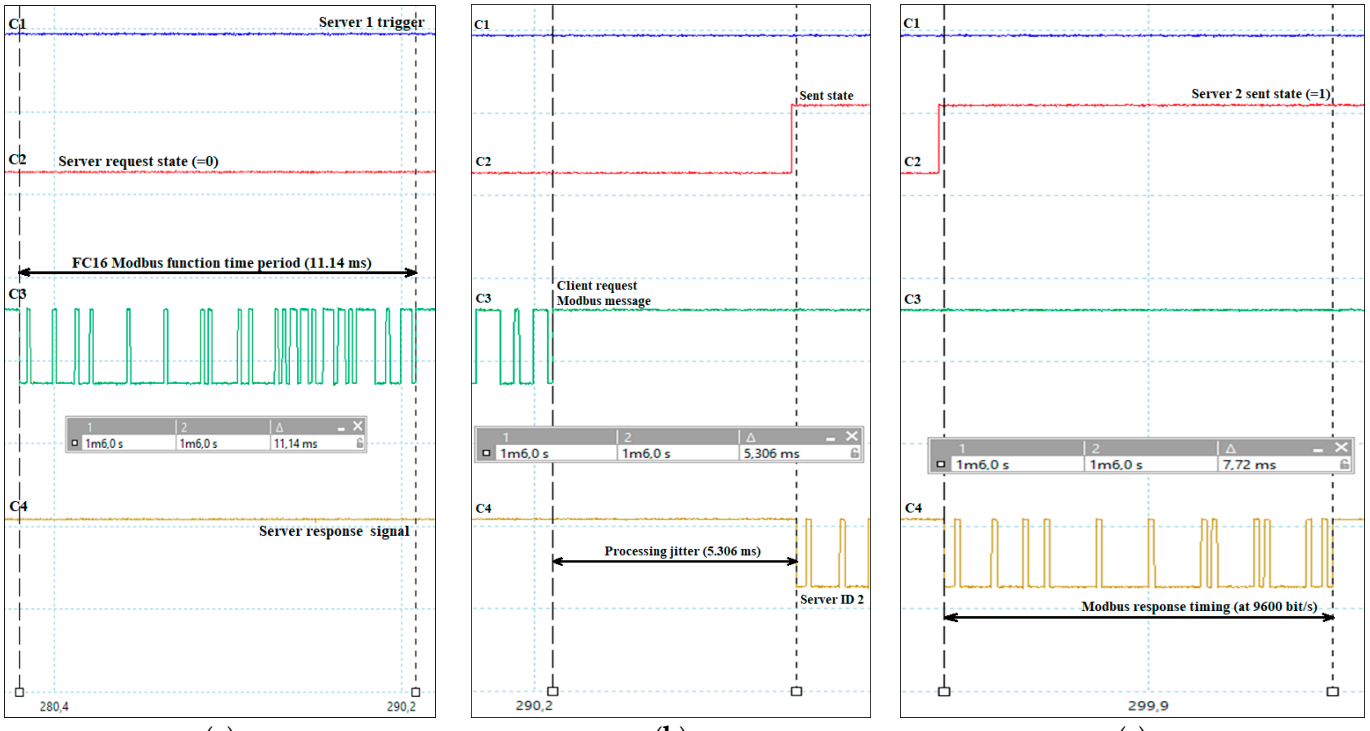


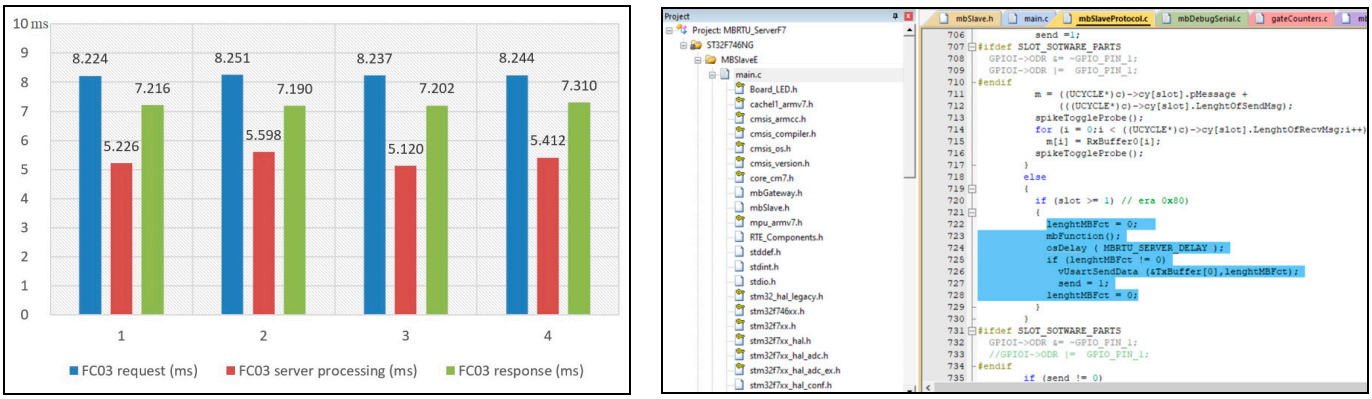
Figure 6. FC16 function experimental validation: (**a**) period of server 0x02 request message and FC16; (**b**) server processing jitter with FC16; (**c**) response message period of server with ID 0x02 and FC16.

5. Discussion

The research and design methodology for laboratory models based on the Modbus protocol consists, first of all, in writing and presenting a mathematical model for the Modbus communication protocol. This step allowed the definition and implementation of the tests necessary to determine the update times of values from/to a device connected to a Modbus RTU bus. In step two, Figure 2 illustrates the principle diagram of Modbus message communication, highlighting the various components that could cause measurable delays. Then a Modbus transaction was defined that takes place on the physical Modbus layer. Based on this, the structure of an acquisition slot at the level of a client running on an evolved operating system was presented. This slot contains a single Modbus transaction. Now we have all the theoretical observations and relationships given by the delay of Modbus transactions on the RS485 bus. As there were no Modbus stations available to meet the project requirements, in step 3 an application was built on an STM32F Discovery development kit on which 247 Modbus Server stations were emulated. To improve the acquisition cycle times, in step 4 an extension (ModbusE) of the incompletely defined Modbus protocol was proposed. Thus, the practical results obtained were presented and analyzed in the paper. The paper also addresses the TCP/IP stack as the authors are considering the design of a Modbus Gateway. The IoT gateway will be equipped with a MODBUS TCP/IP server. Server implementations based on this protocol will have an interface commonly referred to as a data provider that will provide a wrapper between the server and the MODBUS TCP/IP driver. Thus, via the OPC UA server, the gateway will be accessible from IoT applications. To recreate an environment similar to the one in the presented experiment, researchers can simply use the open Modbus protocol, a development kit (RTX RTOS and Keil design environment) to emulate a server, a Client PC, Modbus Poll software and an oscilloscope for AC measurement.

In this study, based on the Modbus protocol, we validated the interchange times from a thermostat to a fan coil for the most favorable case: when the thermostat has address 1 and the fan coil has address 2. In these laboratory tests, we considered all as to having 1 or 10 consecutive registers to be updated for a communication speed of 9600 bps. The update times from one Modbus server (thermostat) to another server (fan coil) on the same network for the most favorable case validated the mathematical model for the analysis and testing in this study. We tested the read (FC03) and write (FC16) functions between the client and server devices belonging to the same Modbus communication protocol.

We presented the validation signals and the obtained values in Section 3, considering an acquisition cycle with two Modbus RTU server stations. For this, we used Modbus Poll software and two server stations with addresses 1 and 2; we set the register address to 0x00, and considered the related time stamps. Based on the hierarchical levels of the current communication protocols, we were able to set up complex communication systems in a structured way. In addition, the strictly hierarchical structure of the OSI model allows heterogeneous systems to be interconnected on different layers. As mentioned above, the client sends commands to the different server units to determine the status of the process inputs or to change the output status using the Modbus protocol. We validated the period and jitter of a Modbus transaction, described the mathematical function that defines an acquisition slot (adr, FCxx, and nreg) at the application per client level, and determined the function that defines a tAC (S_AC) acquisition cycle. A best-case two-server acquisition cycle was between 199.8 and 200.7 ms, and the shortest time between two consecutive requests from server ID 0x01 (FC03) to server ID 0x02 (FC16) was 97.62 ms. The average time corresponding to the server query message with ID 0x01 and FC03 function was 8.239 ms (Figure 7a). The processing time of the server (request analysis and response, forced delay = 1 ms) ID 0x01 FC03 was between 5.12 and 5.598 ms, and the maximum jitter of the server response message with ID 0x02, function FC16, was 7.72 ms. We used the Keil uVision environment and the integrated debugger for the design (Figure 7b).



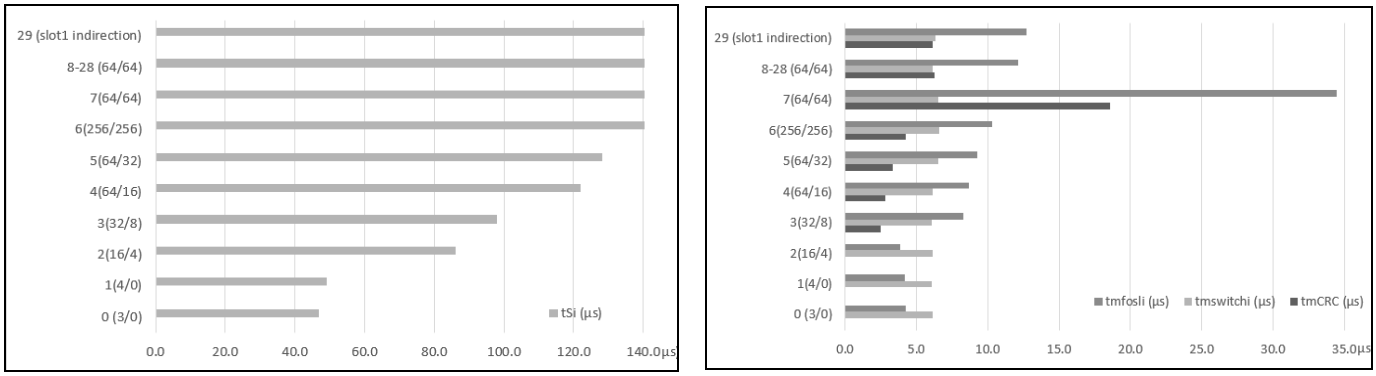
(**a**) (**b**)

Figure 7. Measured times and microcontroller software application: (**a**) measured times corresponding to FC03 and FC16 Modbus functions; (**b**) Modbus server protocol development using Keil uVision environment.

Regarding the software used, RTX RTOS, we integrated the hardware abstraction layer (HAL) libraries, the main task being void mbSlaveThreadProtocol (void const * argv).

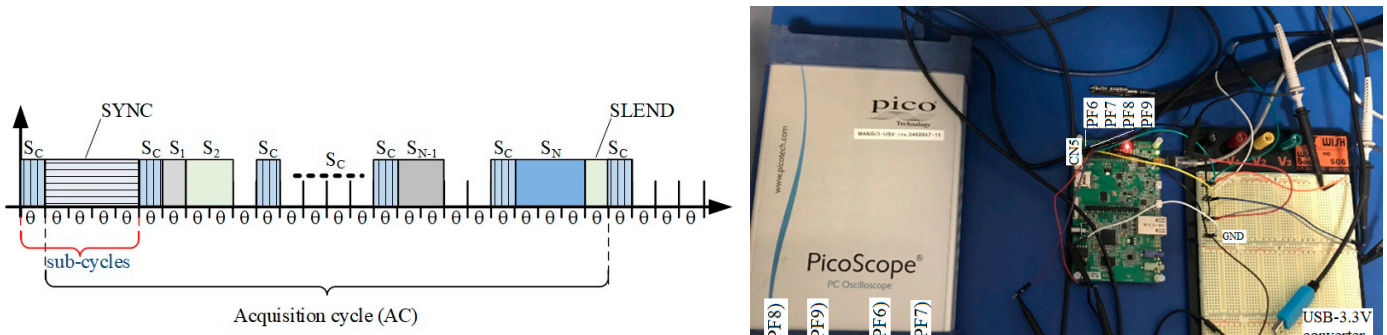
The advantage of the OSI model and its importance for implementation in practice are due to the consistent connection of three key concepts: protocol, service, and interface. Protocol refers to the set of rules that governs layer communication at the same level. To perform the rather complicated task of standardized implementation for data communication, we showed that dividing it into a strictly hierarchical, layer-based model is an advantage in the design of communication systems. Thus, this layered organization can be applied even in industrial communications. All relevant communication functions were ordered into groups that rely on each other. On this basis, a reasonable degree of modularization was sought, so that seven levels represent a feasible compromise, producing the highest performance.

Incompletely defined protocols such as Modbus do not address how and in what order the acquisition occurs on the communications bus. For this reason, in ModbusE, we defined an acquisition cycle that makes the protocol deterministic. Thus, the protocol also allows the addition of a time stamp for messages in the acquisition cycle. In this study, from Modbus communication analysis, we found that the proposed protocol led to decreases in Modbus acquisition times; thus, this ModbusE proposal is a novelty in the field, which we validated through experimental data. We implemented the ModbusE concept at the application level. ModbusE was initially defined [18] specifying the structure of the message, the structure of the AC, and the base station gateway (BSG) as the client device. Our validation of ModbusE allowed the design of a new communication message structure, an acquisition cycle to obtain a temporal behavior, a description of the Modbus devices, and the definition of an architecture for integration in the industrial Internet of things (IIoT). We defined mathematical equations, on which we calculated specific times in a slot of the acquisition cycle. Figure 8a shows the time periods for the acquisition slot, and Figure 8b illustrates the jitter for tmfosli, tmswitch, and tmCRC in the case of ModbusE implementation. Here, tmfosli represents the time required to finish the operations related to the old slot, tmswitchi is the time required to switch the mbeThreadCycleRTU task, and tmCRC represents the CRC calculation timing for a message received in the previous slot. Control loops can be designed using fieldbuses. For example, a control loop might consist of an actuator and a proportional–integral–derivative (PID) controller, which requires the value provided by a sensor. If the control loop cycle is shorter than the AC, a subcycle must be defined (Figure 9). A slot scheduling method was designed for the situation where the Sc time interval between two consecutive slots exceeds the longest acquisition slot [18]. If multiple loops exist in the same subloop, the problem is similarly solved, provided that the remaining difference is larger than the longest span of the planned space.



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

Figure 8. Measured times for acquisition cycle slots at client: (**a**) periods for tSi corresponding to ModbusE acquisition slot; (**b**) jitter for tmfosli, tmswitch, and tmCRC times with ModbusE extension.



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

Figure 9. ModbusE acquisition cycle and related laboratory tests: (**a**) proposed solution for control loop acquisition period that is shorter than acquisition cycle; (**b**) practical measurements related to Modbus communication acquisition cycle.

The gaps between Sc slots can be filled and an acquisition cycle can be created in two ways: gap filling by considering the slot number, such as priorities, and optimal gap filling, in which case the slots are used as efficiently as possible, without unused ticks. So, to calculate the time of an acquisition cycle, a priority-based algorithm can be used, where the priority is the slot number, with the lowest number representing the highest priority. In the proposed algorithm for communication, there are SYNC slots, i = 1, 2,..., N, SLEND and Sc. Thus, the algorithm computes the AC and, consequently, the positions of the slots in the AC, as well as the position and size of the idle times of the communication lines. These gaps can be used to transmit SDO messages if they fall within the gap time period. Gaps are available in BSG; moreover, Sc can be Sc1 + Sc2 + Sc3 +..., provided all control loops have the same period (Figure 9). In most cases, industrial processes are implemented in a hierarchical manner on levels and, depending on the position, the challenges are different. How control software is written also introduces different needs in communications networks. In selecting a communications network, the various considerations that network designers have when building their solution must be understood. A production system is controlled by numerous computers organized on several hierarchical levels. Computer networks provide communications between computers on one level and with some computers on adjacent levels.

Unlike other industrial buses and protocols, no physical interface (OSI level 1) has been defined. Modbus is a simple, flexible, public protocol that allows devices to exchange

discrete and analogue data. Implementing Modbus specifications as the required interface protocol between communication subsystems is one method of integrating the solutions proposed by vendors. Thus, several design options are available for developing data acquisition systems at the lowest cost. The RTU module, sometimes referred to as Modbus-B for Modbus binary, is the preferred Modbus module.

The ASCII transmission mode, sometimes referred to as Modbus-A, corresponds to a typical message that is approximately twice as long as the equivalent RTU message. The Modbus protocol implements transmission and communication error control. Thus, communication errors are found by character framing, parity testing, or cyclic redundancy checking (CRC-16) [24,25]. The latter may vary depending on the Modbus RTU or ASCII transmission mode. Package modules can also be sent through local or regional networks by encapsulating Modbus data in a TCP/IP package [26].

We successfully analyzed and tested the Modbus message exchange between devices belonging to the same communication protocol according to the Modbus protocol specifications. In the future, the performance of the ModbusE acquisition cycle will be integrated into the implementation of an IoT-enabled access gateway (IIoT) ModbusE_Gateway, as part of the IIoT_ModbusE_System. We will evaluate ModbusE times in the form of BSG (SW-OPC UA Client) and server-type workstations (IIoT_Modbus_Server). Thus, the Modbus extension will be validated regarding the new structure of the message and of the acquisition cycle to obtain a deterministic temporal behavior of the IIoT_ModbusE_Gateway-IIoT_Modbus_Server system. OPC UA will evolve into increasingly smaller devices and sensors [27,28]. The smallest OPC UA-based software projects with limited functionality use 35 kB RAM and 240 kB flash. However, chips with integrated OPC UA continue to advance in the sensor world.

6. Conclusions and Future Work

Our contributions with the study include the analysis of Modbus transmission: we measured the Modbus communication signals with an oscilloscope. Thus, we evaluated the delays corresponding to different configurations of the server stations using a different number of messages, i.e., different sizes. Studies of this type are not found in the applied science literature, e.g., in the field of embedded systems. Moreover, we proposed a Modbus extension, thereby improving Modbus message merging and thus reducing communication times and improving data flow on RS485 networks. These proposals and studies were based on the analysis in the first part of the study, in an architecture that incorporates the Modbus protocol, SmartConvert, STM32F7, and RTX RTOS. In future studies, we will use the LTM2881 driver for RS485. Another contribution concerns the measurement of the delays introduced by the Windows 10 operating system with an average load in the communication between the PC client and the server stations.

The ModbusE protocol can be integrated into embedded devices with medium and high response times in data acquisition processes. The specific application of the experiment can be industrial environment and IIoT applications. For implementation and commercialization, the authors plan in the future to integrate the ModbusE concept into a set of Building Internet of Things (BIoT)-based smart switches.

After testing, we concluded that the limitations are produced by the PC operating system, which introduces nondeterministic delays. Tests will determine the maximum of these delays under constant-load conditions. However, ideally, the operating system should only be used for a specific application. Systems that can use the ModbusE protocol are industrial monitoring processes, medium and high response time systems, and the IIoT.

Author Contributions: Conceptualization, I.Z. and V.G.G.; software, I.Z. and V.G.G.; data curation, I.Z. and V.G.G.; writing—original draft preparation, I.Z. and V.G.G.; writing—review and editing, I.Z. and V.G.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the project "119722 / Centru pentru transferul de cunoștințe către întreprinderi din domeniul ICT–CENTRIC-Contract subsidiar 15682/ 02.09.2020/ CRIODRIVE/

Mechatronics Innovation Center", contract no. 5/AXA 1/1.2.3/G/13.06.2018, cod SMIS 2014+ 119722 (ID P_40_305), using the infrastructure from the project "Integrated Center for research, development and innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for fabrication and control", contract no. 671/09.04.2015, Sectoral Operational Program for Increase of the Economic Competitiveness co-funded from the European Regional Development Fund.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing not applicable.

Acknowledgments: The authors would like to thank the editor and the anonymous reviewers for reviewing our manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zurawski, R. *The Industrial Communication Technology Handbook*; CRC Press: Boca Raton, FL, USA, 2015; ISBN 13-978-1-4822-0733-0.
2. Găitan, V.G.; Zagan, I. *Rețele Industriale Locale—Modbus Extins*; Editura Universităţii Ştefan cel Mare din Suceava: Suceava, Romania, 2019; ISBN 978-973-666-552-3.
3. Goldenberg, N.; Wool, A. Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems. *Int. J. Crit. Infrastruct. Prot.* **2013**, *6*, 63–75. [CrossRef]
4. Nyasore, O.N.; Zavarsky, P.; Swar, B.; Naiyeju, R.; Dabra, S. Deep Packet Inspection in Industrial Automation Control System to Mitigate Attacks Exploiting Modbus/TCP Vulnerabilities. In Proceedings of the 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), Baltimore, MD, USA, 25–27 May 2020; pp. 241–245.
5. Wang, Y.; Gaspes, V. A compositional implementation of Modbus in Protege. In Proceedings of the 6th IEEE International Symposium on Industrial and Embedded Systems, Vasteras, Sweden, 15–17 June 2011; pp. 123–131. [CrossRef]
6. Lemay, A.; Fernandez, J.M.; Knight, S. A Modbus command and control channel. In Proceedings of the 2016 Annual IEEE Systems Conference (SysCon), Orlando, FL, USA, 18–21 April 2016; pp. 1–6. [CrossRef]
7. Jean, L. Python Software Foundation—Modbus_tk 0.5.8. Available online: https://pypi.python.org/pypi/modbus_tk (accessed on 20 July 2022).
8. Cena, G.; Cereia, M.; Cibrario Bertolotti, I.; Scanzio, S. A MODBUS extension for inexpensive distributed embedded systems. In Proceedings of the 2010 IEEE International Workshop on Factory Communication Systems Proceedings, Nancy, France, 18–21 May 2010; pp. 251–260. [CrossRef]
9. Al-Dalky, R.; Abduljaleel, O.; Salah, K.; Otrok, H.; Al-Qutayri, M. A Modbus traffic generator for evaluating the security of SCADA systems. In Proceedings of the 9th International Symposium on Communication Systems, Networks & Digital Sign (CSNDSP), Manchester, UK, 23–25 July 2014; pp. 809–814. [CrossRef]
10. Hu, T.; Bertolotti, I.C. Overhead and ACK-induced jitter in Modbus TCP communication. In Proceedings of the 2015 IEEE 1st International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), Torino, Italy, 16–18 September 2015; pp. 392–397. [CrossRef]
11. LPC2468 Product Data Sheet, Rev. 4, NXP B.V., October 2008. Available online: https://www.nxp.com/docs/en/data-sheet/LPC2468.pdf (accessed on 22 July 2022).
12. Urrea, C.; Morales, C.; Muñoz, R. Design and implementation of an error detection and correction method compatible with MODBUS-RTU by means of systematic codes. *Measurement* **2016**, *91*, 266–275. [CrossRef]
13. Lai, Y.; Gao, H.; Liu, J. Vulnerability Mining method for the Modbus TCP using an Anti-sample Fuzzer. *Sensors* **2020**, *20*, 2040. [CrossRef] [PubMed]
14. Zhao, L.; Qu, S.; Zang, W. Design of multi-channel data collector for highway tunnel lighting based on STM32 and Modbus protocol. *Optik* **2020**, *213*, 164388-10. [CrossRef]
15. Figueroa-Lorenzo, S.; Añorga, J.; Arrizabalaga, S. A Role-based access Control model in Modbus SCADA systems. A centralized model approach. *Sensors* **2019**, *19*, 4455. [PubMed]
16. Urrea, C.; Kern, J.; Morales, C. Error detection and correction to enhance the data rate of smart metering systems using Modbus-RTU. *Electtr. Eng.* **2021**, *103*, 115–124. [CrossRef]
17. Urrea, C.; Morales, C. Enhancing Modbus-RTU Communications for Smart metering in building Energy Management systems. *Secur. Commun. Netw.* **2019**. [CrossRef]
18. Găitan, V.G.; Găitan, N.C.; Ungurean, I. A flexible acquisition cycle for incompletely defined fieldbus protocols. *ISA Trans.* **2014**, *53*, 776–786. [CrossRef] [PubMed]
19. Modbus Organization. MODBUS Messaging on TCP/IP Implementation Guide V1.0b. Available online: https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf (accessed on 27 July 2022).

20. Skeie, T.; Johannessen, S.; Holmeide, O. Timeliness of real-time IP communication in switched industrial Ethernet networks. *IEEE Trans. Ind. Inform.* **2006**, *2*, 25–39. [CrossRef]
21. Daniel Flow Products, Modbus Communications Model 2500, Part Number: 3-9000-545 Revision D, November. 1992. Available online: https://www.emerson.com/documents/automation/daniel-modbus-communications-model-2500-manual-en-43890.pdf (accessed on 27 July 2022).
22. Găitan, V.G.; Zagan, I. Experimental Implementation and Performance Evaluation of an IoT Access Gateway for the Modbus Extension. *Sensors* **2021**, *21*, 246. [CrossRef] [PubMed]
23. STM32F7, UM1907, Discovery kit for STM32F7 Series with STM32F746NG MCU. 2020. Available online: https://www.st.com/en/evaluation-tools/32f746gdiscovery.html#documentation (accessed on 22 July 2022).
24. Nugur, A.; Pipattanasomporn, M.; Kuzlu, M.; Rahman, S. Design and Development of an IoT Gateway for Smart Building Applications. *IEEE Internet Things J.* **2019**, *6*, 9020–9029. [CrossRef]
25. Modbus Organization. *MODBUS Application Protocol Specification*; Modbus Organization: Hopkinton, MA, USA, 2012; Available online: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf (accessed on 19 May 2022).
26. Martins, T.; Oliveira, S.V.G. Enhanced Modbus/TCP Security Protocol: Authentication and Authorization Functions Supported. *Sensors* **2022**, *22*, 8024. [CrossRef] [PubMed]
27. Pu, C.; Ding, X.; Wang, P.; Xie, S.; Chen, J. Semantic Interconnection Scheme for Industrial Wireless Sensor Networks and Industrial Internet with OPC UA Pub/Sub. *Sensors* **2022**, *22*, 7762. [CrossRef] [PubMed]
28. Toc, S.; Korodi, A. Modbus-OPC UA Wrapper Using Node-RED and IoT-2040 with Application in the Water Industry. In Proceedings of the 2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY), Subotica, Serbia, 13–15 September 2018; pp. 99–104. [CrossRef]