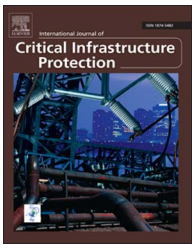
INTERNATIONAL JOURNAL OF CRITICAL INFRASTRUCTURE PROTECTION ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Available online at www.sciencedirect.com

ScienceDirect

www.elsevier.com/locate/ijcip



Implementation of error detection and correction in the Modbus-RTU serial protocol

*Claudio Urrea**, *Claudio Morales*, *John Kern*

Grupo de Automática, Departamento de Ingeniería Eléctrica, Universidad de Santiago de Chile, Avenue Ecuador 3519, Estación Central, Santiago, Chile

ARTICLE INFO

Article history:
Received 15 November 2014
Received in revised form
5 February 2016
Accepted 1 July 2016

Keywords:
SCADA systems
Modbus-RTU protocol
Error detection and correction
Legacy device compatibility

ABSTRACT

Modbus-RTU is currently one of the most widely-used industrial communications protocols. Modbus-RTU is an old protocol – it was developed in 1979 – and has certain limitations with regard to modern automation and control systems. One of the major limitations is that the protocol does not have any mechanisms for ensuring the integrity of transmitted data. To address the limitation, this paper presents a method for recovering corrupted frames due to transmission errors in Modbus-RTU serial communications while retaining complete compatibility with the protocol. The method uses forward error correction codes that add parity information in time windows during which the communications bus is in a rest state. The method complies with the protocol definitions and facilitates the extension of Modbus-RTU characteristics while maintaining compatibility with installed devices and other industrial equipment.

1. Introduction

Modbus is one of the most widely used industrial communications protocols, especially in industrial control applications and energy management and building automation systems [2,9]. The implemented systems often use the Modbus-RTU variant over EIA/TIA-485 serial buses to interconnect plant devices that are integrated via gateways with supervision and management systems, which operate over the Ethernet or with other communications networks that use other protocols.

The Modbus-RTU protocol was developed as far back as 1979. As a result, it has certain limitations with regard to current requirements for automation and control systems, including low transmission rates, limited numbers of networked devices, single master topologies and the absence of mechanisms for correcting messages corrupted by transmission errors.

However, because of its simplicity and widespread use in legacy as well as new systems, Modbus-RTU has been the subject of considerable research. Recent research has focused on approaches for increasing the number of devices that can be connected to a communications bus [9], authentication methods for protecting critical infrastructure assets [7,12], intrusion detection [5] and integration with GPRS networks [14].

In order to ensure the integrity of the data transmitted between Modbus-RTU devices, it is necessary to develop efficient methods for enhancing the Modbus-RTU protocol to enable the recovery of corrupted frames. This would improve transmission quality in environments with substantial electromagnetic interference. Some progress has been made along these lines by Zhang [16], who employed a Reed–Solomon coding/decoding algorithm to achieve fault-tolerant communications in Modbus-TCP (Ethernet based) networks.

*Corresponding author.
E-mail address: claudio.urrea@usach.cl (C. Urrea).

http://dx.doi.org/10.1016/j.ijcip.2016.07.001
1874-5482/

However, a solution for Modbus devices that use serial lines is not yet available. Research has also focused on handling communications errors arising from data collisions [4], but when transmission errors are caused by electromagnetic interference on serial lines, the only available solution is retransmission. This paper addresses both the deficiencies by presenting a method for recovering corrupted frames due to transmission errors induced by electromagnetic (and other interference) in serial Modbus-RTU communications while retaining complete compatibility with the protocol.

2. Modbus-RTU protocol

Modbus-RTU is a variant of the popular Modbus communications protocol used to transmit industrial control commands and data at the plant level. Its specifications are available in two technical documents prepared by the Modbus Organization: (i) Modbus Application Protocol, which defines the structure of messages in a client/server mode at the application layer; and (ii) Modbus Serial Line Protocol, which specifies a master/slave architecture for the data link layer, the ASCII (American Standard Code for Information Interchange) and RTU (remote terminal unit) transmission modes, and the implementation requirements over EIA/TIA-485 and EIA/TIA-232 lines. This paper focuses on the most common implementation, namely transmission in the RTU mode over serial EIA/TIA-485 two-wire lines.

The master–slave architecture for Modbus-RTU stipulates that there can be only one master and between one and 247 slaves in a network. Only the master can start a communication by sending a request message to one or more slaves via the unicast or broadcast modes (see Fig. 1):

- *Unicast mode*: The master sends a request to a single slave. After processing the request, the slave sends a reply message back to the master.

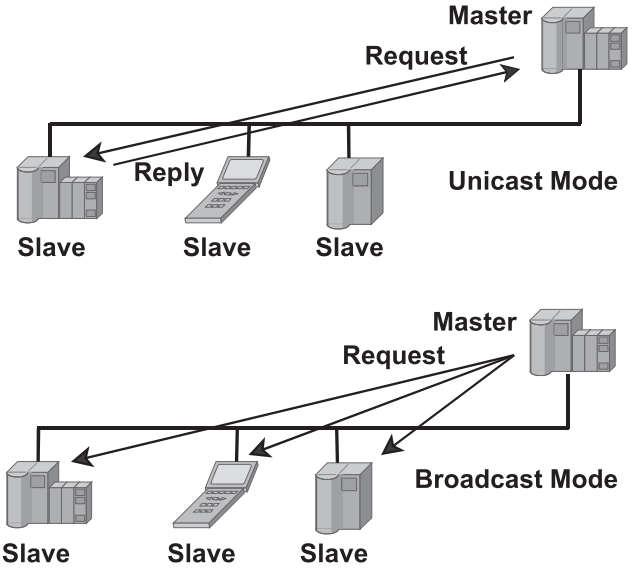


Fig. 1 – Master/slave communications in the unicast and broadcast modes.

- *Broadcast mode*: The master sends a request to all the slaves simultaneously using the broadcast address. Each slave processes the request, but no reply message is sent back to the master.

Master request messages and slave reply messages have the single frame structure shown in Fig. 2. The frame starts at layer 7 of the OSI reference model, where the protocol data unit (PDU) is constituted. The protocol data unit contains information from the master's request or a slave's reply, and it is completed at layer 3, with an address field and a cyclic redundancy code (CRC), to constitute the application data unit (ADU) that is sent to the communications bus. A frame has a variable length that depends on the amount of information to be transmitted; its maximum length is 256 bytes.

A Modbus-RTU frame is transmitted serially as 8-bit characters, each corresponding to two 4-bit hexadecimal data. A message is transmitted as a continuous frame of characters such that it is possible to detect the beginning and the end of the message. This is achieved by defining time intervals between characters and between frames. Message frames must be separated by time intervals that are at least 3.5 times the duration of one character, which is expressed as $t_{3.5}$. If a time interval is longer than 1.5 times the duration of a character ($t_{1.5}$), then the frame is considered to be incomplete and is discarded by the receiver without a notification. Fig. 3 shows the admissible times between Modbus-RTU characters and frames.

After a receiver detects the end of the message, it uses the 16-bit cyclic redundancy code field to verify the integrity of the received data. If there are no errors, the message is processed; otherwise, it is discarded. This approach makes it possible to detect transmission errors and prevent the processing of corrupted data, but it does not ensure the recovery of lost information.

According to the protocol definition, if the receiver of a frame detects an inconsistency when verifying the cyclic redundancy code, it simply discards the message without sending a notification to the sender. In the unicast mode, the master can detect that there has been a loss of information after waiting for a specific period of time (timeout) without receiving a valid reply from the slave; the request can then be retransmitted. However, in the broadcast mode, the master does not expect a reply from a slave and, therefore, no mechanism is available for detecting whether or not the request was discarded by one of more slaves and, therefore, all the slaves did not carry out the requested operation.

When Modbus-RTU messages are transmitted over copper lines in environments with strong electromagnetic interference, the resulting transmission errors degrade network performance because considerable time is required to retransmit the corrupted messages. In these instances, the

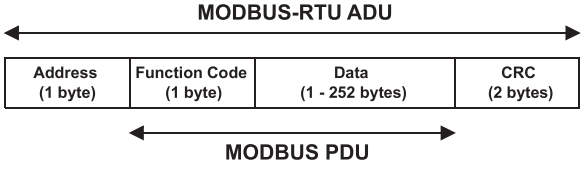


Fig. 2 – Generic Modbus-RTU message frame.

Please cite this article as: C. Urrea, et al., Implementation of error detection and correction in the Modbus-RTU serial protocol, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.07.001

conventional solutions are to use fiber optic cabling or electromagnetic shielding, both of which are expensive propositions. An alternative solution is to add redundant information in Modbus-RTU frames for error correction. As described in this paper, this can be done by taking advantage of the time windows between messages that are defined in the Modbus-RTU protocol.

3. Adding redundant information to Modbus-RTU transactions

Modbus-RTU master–slave communications require that transactions take place one at a time. Specifically, the master does not send a new request until a reply is received from a slave or a communications problem is detected as a result of not receiving a reply (timeout situation). In a typical Modbus-RTU system, the master makes requests to slaves in a cyclic manner at sufficiently-long time intervals (slow polling rate) to ensure that a slave receives the request, processes it and sends its reply, which is then received and processed by the master before the master makes a new request to a slave. This type of transaction produces a time window after every message as shown in Fig. 4. The time window can be leveraged to add parity information for error detection and correction.

The time window that appears after a slave reply is determined by the polling rate. In most industrial control networks, the time window is large enough to place parity information on the bus without the risk of data collisions. However, this is not the case for the time window between the receipt of the master's request by a slave and the sending of the slave's reply. In this case, the time window is much narrower because it corresponds only to the time taken by the slave to process the master's message and set up the reply frame. This is determined by:

- The data transmission rate, which is measured in bits per second (bps).
- The serial transmission format, which defines the number of bits per character.
- The time taken by the Modbus slave to process the master's request and set up the reply message.

The data transmission rate and the serial transmission format are defined by the configuration of the transmission and are related to the instant at which the slave detects the end of the request message, which corresponds to a time equal to 3.5 characters ($t_{3.5}$). For a given transmission rate and frame format, the two variables determine the minimum processing time for each request. For example, for a transmission rate of 19,200 bps with an 8N1 frame format (total 10 bits: one start bit, eight data bits, no parity and one stop bit), every character requires $t_c = 10/19,200 = 0.52$ ms to be transmitted. Thus, the end of message detection by the slave occurs after a time $t_{3.5} = 1.82$ ms. The execution time t_e, on the

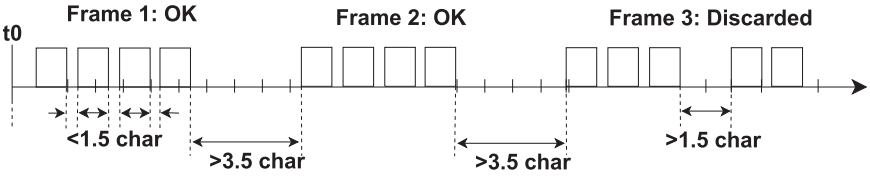


Fig. 3 – Admissible times between Modbus-RTU characters and frames.

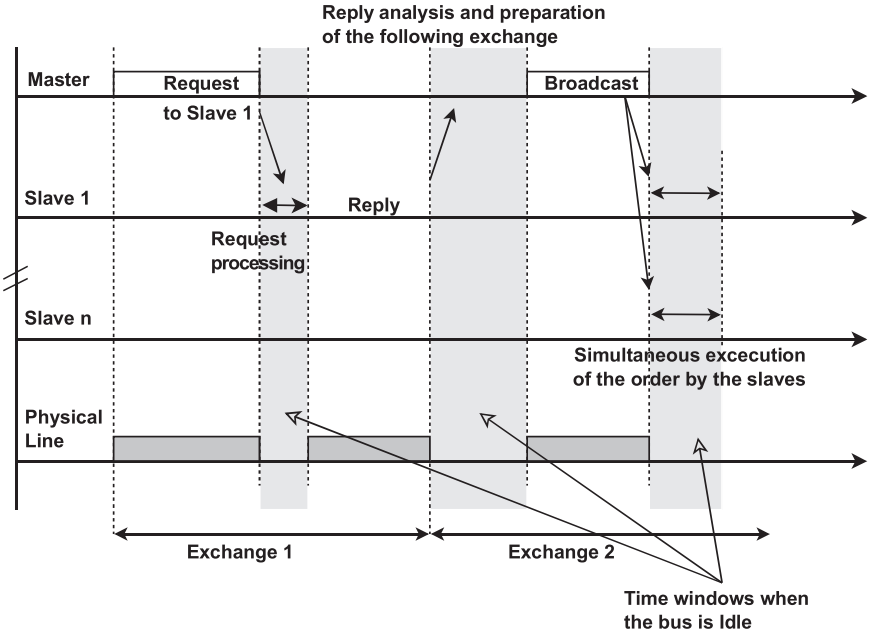


Fig. 4 – Time windows available for adding parity information.

Please cite this article as: C. Urrea, et al., Implementation of error detection and correction in the Modbus-RTU serial protocol, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.07.001

other hand, depends on the data processing speed of the slave device, the priority that the request has with respect to the other tasks executed by the slave and the execution complexity of the request. This time is usually quite variable and depends on the characteristics of the slave device.

As shown in Fig. 5, the total processing time t_p comprises the end of message detection time and the execution time, i.e., $t_p = t_e + t_{3.5}$. During this time window, it is a certainty that no device will place data on the bus. Therefore, it is possible to add parity information during this time window to detect and correct errors.

Preliminary measurements of Modbus-RTU communications in industrial devices yielded execution times t_e ranging from 1.3 ms to 11.5 ms. This makes it possible to add at least two characters (16 bits of parity) at a transmission rate of 19,200 bps with absolutely no possibility of collision.

To ensure compatibility with the Modbus-RTU protocol, an adequate coding method must be selected that keeps the frame unaltered and places the parity information at the end of the frame on the sender (master) side. This leverages the time window given by the processing time so that it can be ensured that no device will transmit data during this window. It is also necessary to perform, on the receiver (slave) side, the decoding of the received frame with the additional parity information, along with the detection and correction of a corrupted frame. Thus, it is possible to reconstruct the original frame and create the reply to the sender of the request (master), reducing information loss and eliminating delays due to message retransmission.

4. Forward error correction

Detection and correction codes at the receiver side, commonly known as forward error correction (FEC), have been used for decades to improve transmission quality in digital communications systems. The error correction is based on codes that add redundancy to the original data so that, even when errors are introduced during transmission (up to some tolerance level), the original information can be recovered [13]. A variety of error correction codes have been proposed in the literature [1]:

- *Linear and nonlinear codes*: These codes are classified according to whether or not they comply with linear algebra.
- *Cyclic and noncyclic codes*: These codes are classified according to whether or not the cyclic rotation of a valid code generates another valid code.
- *Block codes and convolutional codes*: These codes are classified according to whether or not the coding process requires memory.
- *Binary and nonbinary codes*: These codes are classified according to whether error detection is carried out over bits or over symbols (i.e., groups of bits).
- *Systematic and nonsystematic codes*: These codes are classified according to whether or not the original message appears unaltered in the coded message.

This paper employs systematic codes because they allow redundant (parity) information to be added to the original message without making any changes to the messages. Thus, the original information appears unaltered in the coded message. The parity information is computed and transmitted at the sender side. Errors are detected and the original message is reconstructed at the receiver side. This process is illustrated in Fig. 6.

The following systematic codes stand out in the literature:

- *Low density parity check codes*: These codes exhibit excellent performance, but have complex coder and decoder designs. They are mainly used in digital video transmission [1].
- *Luby transform systematic codes*: This is a well-known family of rateless codes for binary erasure channels [15].
- *Systematic raptor codes*: This subclass of Luby transform codes is used for wireless transmissions by cellular phones [3].
- *Systematic turbo codes*: These codes are reliable and have energy efficiency values very close to the Shannon limit. The codes also have good error correction properties, but

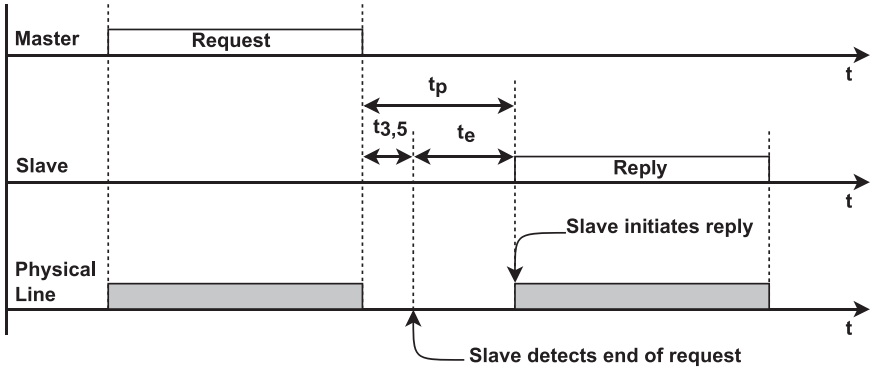


Fig. 5 – Processing time for master–slave transactions.

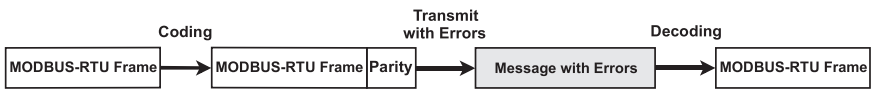


Fig. 6 – Coding/decoding process with systematic codes.

Please cite this article as: C. Urrea, et al., Implementation of error detection and correction in the Modbus-RTU serial protocol, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.07.001

they are inadequate for most applications, except for satellite communications where the latencies due to the great distances are so large that code latency is irrelevant [11].

- *Reed–Solomon codes*: These codes have less coding gain compared with low density parity check codes and turbo codes, but they have higher coding rates and lower complexity. They are used in various data saving and transmission applications, from compact disks to aerospace communications [10].

As a first approximation, any of these or other systematic codes may be used to add parity information to correct errors in Modbus-RTU communications as long as the information can be included in the time window set by the reply time of the slave device. The task of determining the coding method that most efficiently satisfies the application at hand is left for future research.

5. Applications

The method proposed for error detection and correction in Modbus-RTU communications is especially applicable in situations where bus segments are prone to transmission errors due to electromagnetic interference. A pair of devices would be positioned in a bus segment to perform the coding/decoding tasks using a systematic forward error correction code that enables the detection and correction of transmission errors within the available time windows. These devices are called repeater/error correctors (R/ECs). The two devices would improve the transmission quality in the face of electromagnetic interference without affecting the operation of other devices connected to the bus.

Fig. 7 shows a typical application of the proposed method. Fig. 7(a) shows a network comprising a master and several slaves where the communications between the master and slave *n* pass through a segment prone to errors caused by electromagnetic interference. Communications between the master and the remaining slaves take place without any problems. To improve the quality of the transmissions between the master and slave *n*, one repeater/error corrector is positioned at the master end and the other at the slave *n* end (Fig. 7(b)). The repeater/error correctors allow the construction of corrupted frames caused by transmission errors between the master and slave *n* while being transparent to communications between the master and other slaves connected directly to the bus.

Fig. 8 shows the error detection and correction process for Modbus-RTU transactions with repeater/error correctors. The repeater/error corrector on the sender side codes the message to be transmitted and places it on the communications bus. The frame transmitted to the bus comprises a repetition of the original message followed by the parity information needed to detect and correct errors. The repeater/error corrector on the receiver side takes the data and parity information from the communications bus and decodes them to determine if transmission errors exist. If no errors exist, the received message is passed to the destination device; otherwise, the original message is reconstructed and then transmitted to the destination device.

Fig. 9 illustrates transactions between a master with a repeater/error corrector and other connected slaves without repeater/error correctors. In order to avoid affecting the operation of the devices that are connected directly with the bus, the parity information that follows a master request must be transmitted after a time equal to 3.5 characters from the last bit of the transmitted information frame ($t_{3.5}$), which

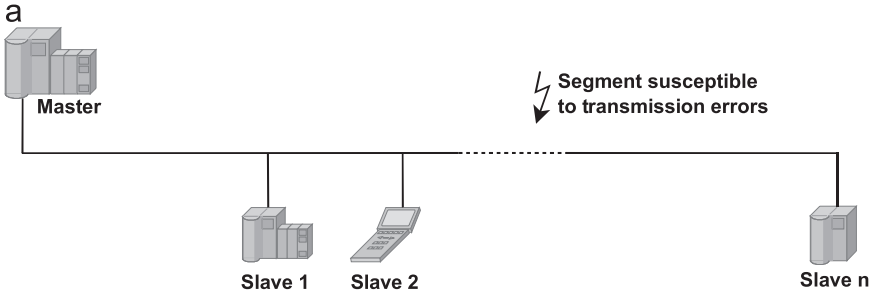


Fig. 7 – Using Modbus-RTU repeater/error correctors: (a) Example network and (b) network with repeater/error correctors.

Please cite this article as: C. Urrea, et al., Implementation of error detection and correction in the Modbus-RTU serial protocol, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.07.001

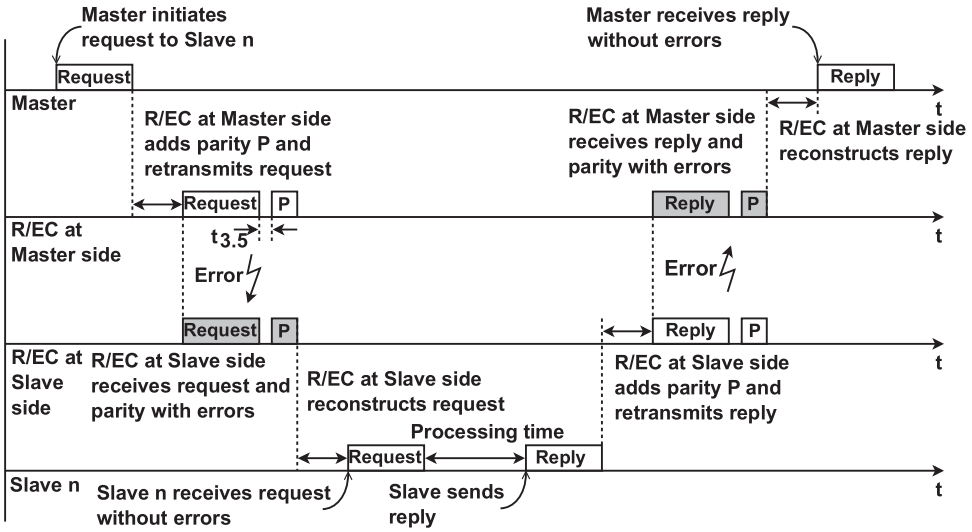


Fig. 8 – Modbus-RTU transactions involving a master and slave with repeater/error correctors.

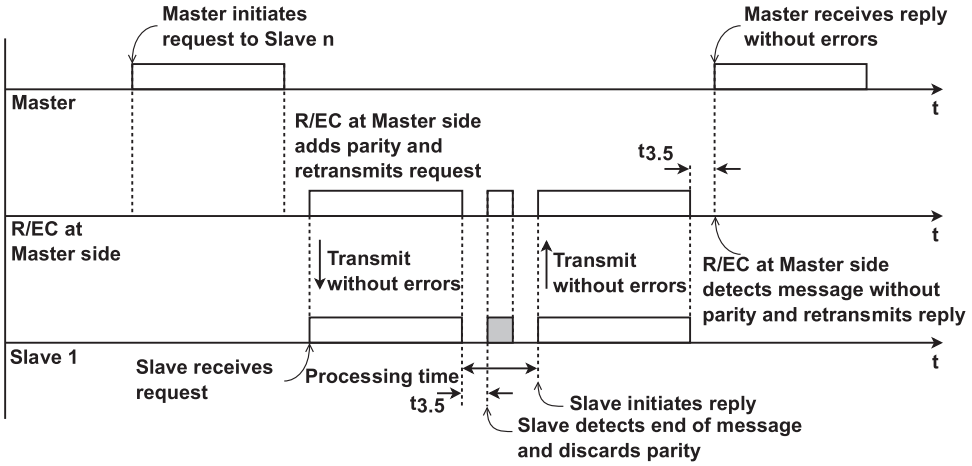


Fig. 9 – Modbus-RTU transactions between a master with a repeater/error corrector and other connected slaves without repeater/error correctors.

is the time after which the slaves detect the end of the message. Thus, the parity information can be discarded by the slaves without repeater/error correctors. A reply message from a slave without a repeater/error corrector is received by the detector/error corrector on the master side; this detector/error corrector would see that the message has no parity information and does not have to be processed for errors, so the message is simply transmitted to the master.

To improve communications, the repeater/error correctors can be attached to any number of slaves in a network. The only limitation is the latency introduced by each transaction, which is comparable in magnitude to the duration of the request frame plus the parity characters and the reply frame. This issue must be considered when setting the maximum polling rate in a large network.

6. Experimental results

This section presents the experimental results obtained using the proposed error detection and correction method on a manipulator robot with five degrees of freedom. The master corresponded to a pcDuino version 1 embedded system while the slave corresponded to a pcDuino version 1 interface in a Modbus-RTU network. Reed–Solomon codes were employed by the repeater/error correctors.

The repeater/error correctors were designed based on an RS(255, 223) coder/decoder with eight-bit symbols [1,8], which, in turn, was based on generic ANSI C code for RS(n, k) over GF(2 m) coders/decoders in [6]. The code libraries were modified to work with RS(255, 251) code in [6] in order to be implemented on a dsPIC30F3010 microcontroller.

Fig. 10 shows the implementation of the message transmission system for controlling the joint trajectory of the manipulator robot. The Modbus-RTU network architecture incorporates a master and a slave, each with a repeater/error corrector.

To verify the ability to detect and correct errors, an impulse noise generator was connected to the communications bus to induce transmission errors. The device incorporated a signal generator adjusted to produce square waves with 500 Hz frequency and a duty cycle of 25%, which was coupled to an impulse transformer. At the specified

Please cite this article as: C. Urrea, et al., Implementation of error detection and correction in the Modbus-RTU serial protocol, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.07.001

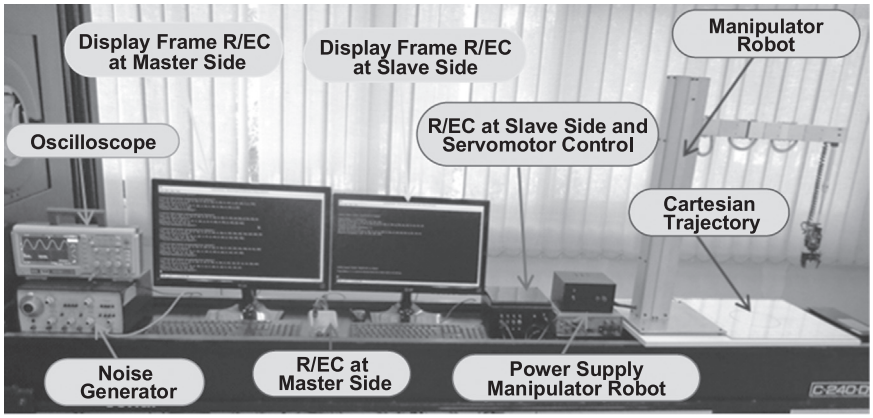


Fig. 10 – Message transmission system implementation.

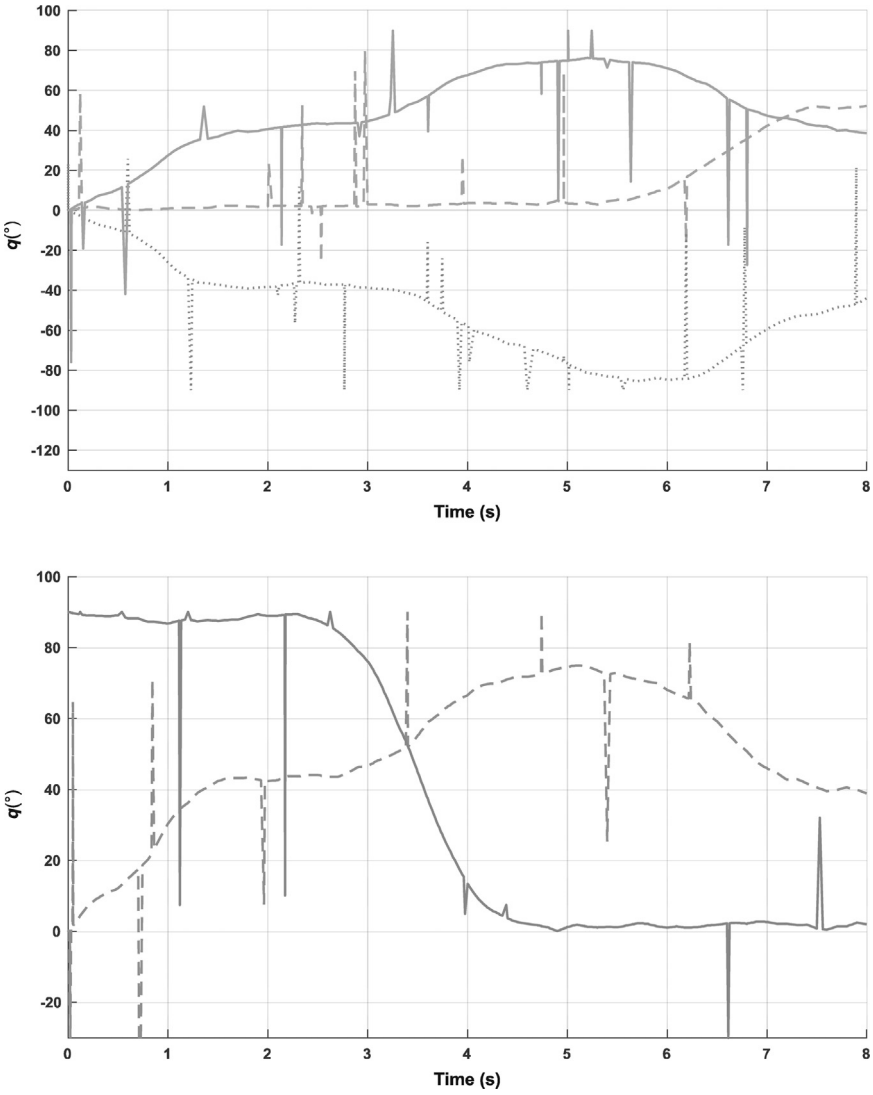


Fig. 11 – Noisy joint trajectory data transmitted without using repeater/error correctors.

frequency and duty cycle, the generator introduced perturbations in the bus every 2 ms that lasted for approximately 0.5 ms. The 2 ms time interval was comparable with the duration of the Modbus frames used in the experiments, and the 0.5 ms was approximately the time taken to transmit a character at 19,200 bps. Thus, the generator caused random perturbations in some Modbus-RTU frames that produced errors in one or two characters in messages.

Please cite this article as: C. Urrea, et al., Implementation of error detection and correction in the Modbus-RTU serial protocol, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.07.001

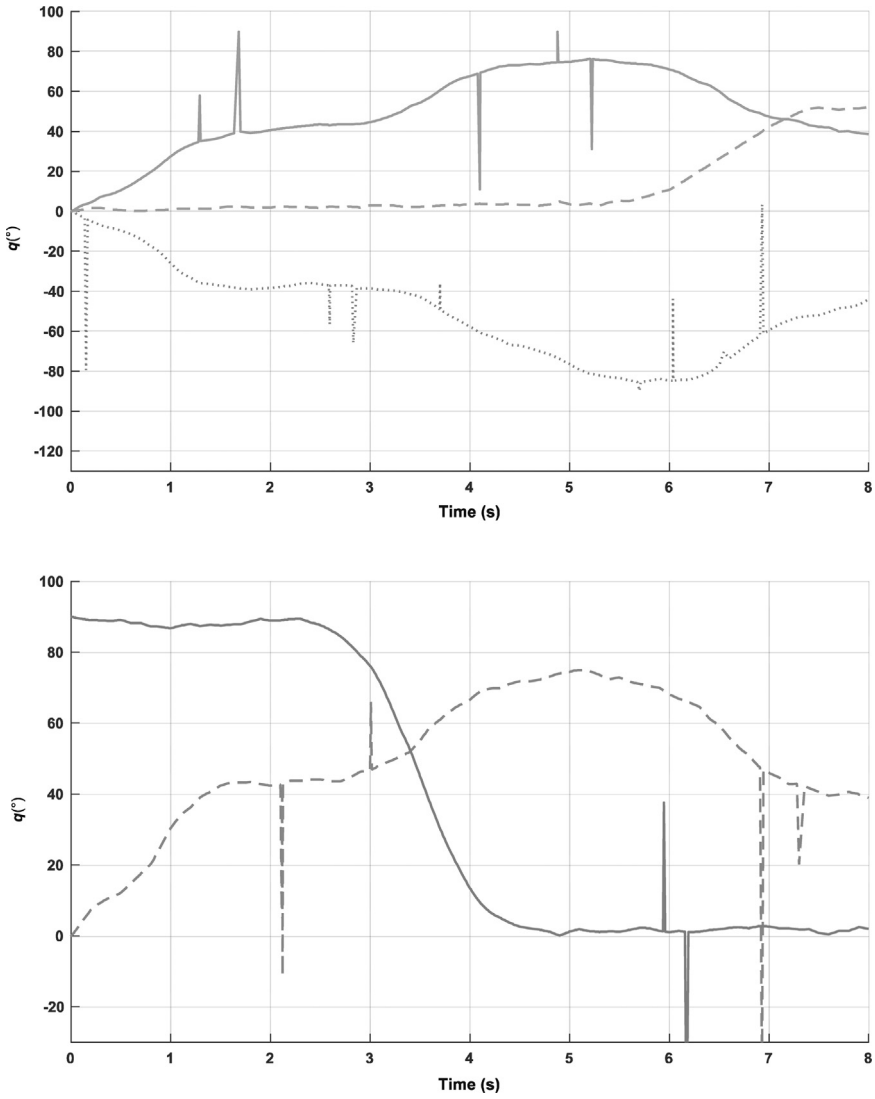


Fig. 12 – Noisy joint trajectory data transmitted using repeater/error correctors.

The joint trajectory data transmitted contained the five individual trajectories ($q1$ to $q5$) corresponding to the motions of the manipulator robot with five degrees of freedom. Fig. 11 shows the curves corresponding to the joint trajectories transmitted with noise to the manipulator robot when repeater/error correctors were used. Fig. 12 shows the curves for the case when repeater/error correctors were not used. In both cases, the intensities of the perturbation impulses were adjusted experimentally until error rates of approximately 60% were achieved (i.e., two of every three transactions failed due to transmission errors).

Comparisons of the curves in Figs. 11 and 12 reveal that a smaller number of frames were affected when repeater/error correctors were used. However, Fig. 12 shows that, when the impulse generator was adjusted to induce perturbations with periods shorter than 2 ms, more than two characters were altered in some messages.

Modbus testing software revealed that the errors were in the form of requests without responses (errors in sending requests) or invalid responses (errors in sending replies). Fig. 13 presents a screenshot of the Modbus testing software results when a transmission was made in the presence of noise without using repeater/error correctors. Note that master requests are dark gray, slave replies without errors are black and slave replies with errors are light gray.

Fig. 14 presents a screenshot when a transmission is made in the presence of noise while using repeater/error correctors. As before, master requests are dark gray, slave replies without errors are black and slave replies with errors are light gray. The screenshot shows a dramatic reduction in errors. However, some errors exceed the corrective ability of the proposed method. This is because more than two characters in the messages were affected, which exceeded the error correcting ability of the RS(255, 251) code. Under these conditions, the transmission errors once again appeared as requests without responses or as invalid responses. A different Reed–Solomon code, such as RS(255, 247), would be able to detect and correct these errors. However, additional parity characters would have to be added at the end of each frame, which would introduce a slightly higher latency for each transaction.

Please cite this article as: C. Urrea, et al., Implementation of error detection and correction in the Modbus-RTU serial protocol, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.07.001

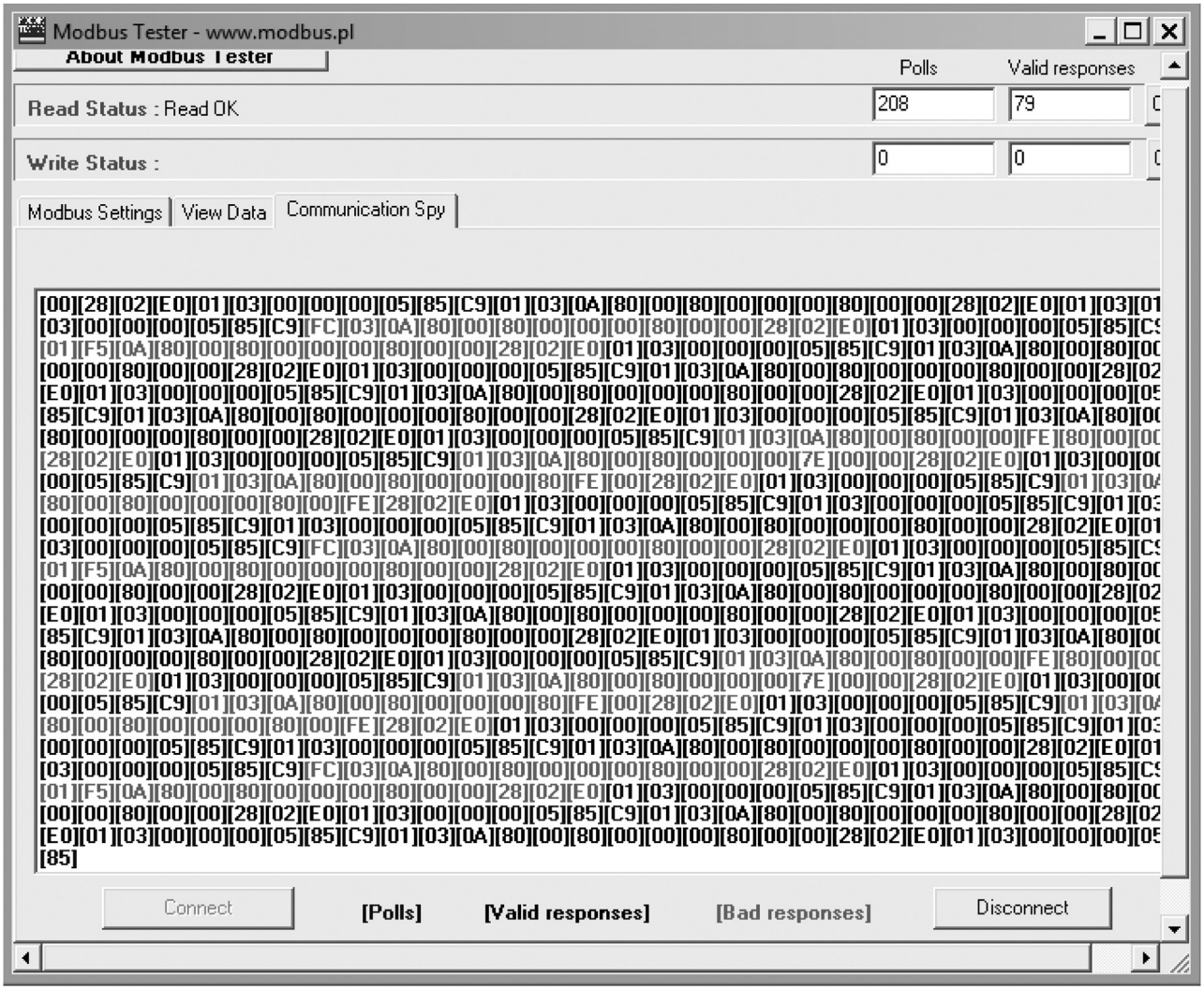


Fig. 13 – Transmission in the presence of noise without using repeater/error correctors.

7. Conclusions

A major limitation of the popular Modbus-RTU protocol is that it has no mechanisms for ensuring the integrity of transmitted data. The proposed method addresses this limitation by adding parity information using a systematic forward error correction code at the end of each Modbus-RTU frame. The parity information is added in time windows between the end of a message and the beginning of the next message. The time windows correspond to the times when devices are busy, either waiting for a new message, carrying out processing tasks or executing instructions; therefore, the communications bus is available for information transmission.

The use of a systematic code keeps the transmitted information unaltered. However, the repeater/error correctors that detect, correct and transmit corrupted messages must have short latency times to avoid data collisions and remain transparent to other Modbus-RTU devices. This also enables the extension of Modbus-RTU characteristics to enhance network performance and reliability while maintaining compatibility with installed devices and other industrial equipment.

The proposed method is especially useful in communications bus segments that are prone to transmission errors caused by electromagnetic interference. In such situations, the method can be a less-expensive option than using fiber optic cabling or electromagnetic shielding. However, it is important to note that, at transmission speeds less than 19,200 bps, most slaves in an industrial Modbus-RTU network take less time to process replies than to transmit the parity characters. In these cases, the proposed method may not be compatible with all the devices attached to the communications bus. To ensure compatibility with all devices, a limited amount of parity data has to be introduced in each frame. Thus, the error detection and correction functionality is tied to the width of the time window corresponding to the execution time of the fastest device connected directly to the communications bus. However, if a repeater/error corrector is positioned at each device, the only practical constraint introduced is latency, which limits the maximum polling rate.

Please cite this article as: C. Urrea, et al., Implementation of error detection and correction in the Modbus-RTU serial protocol, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.07.001

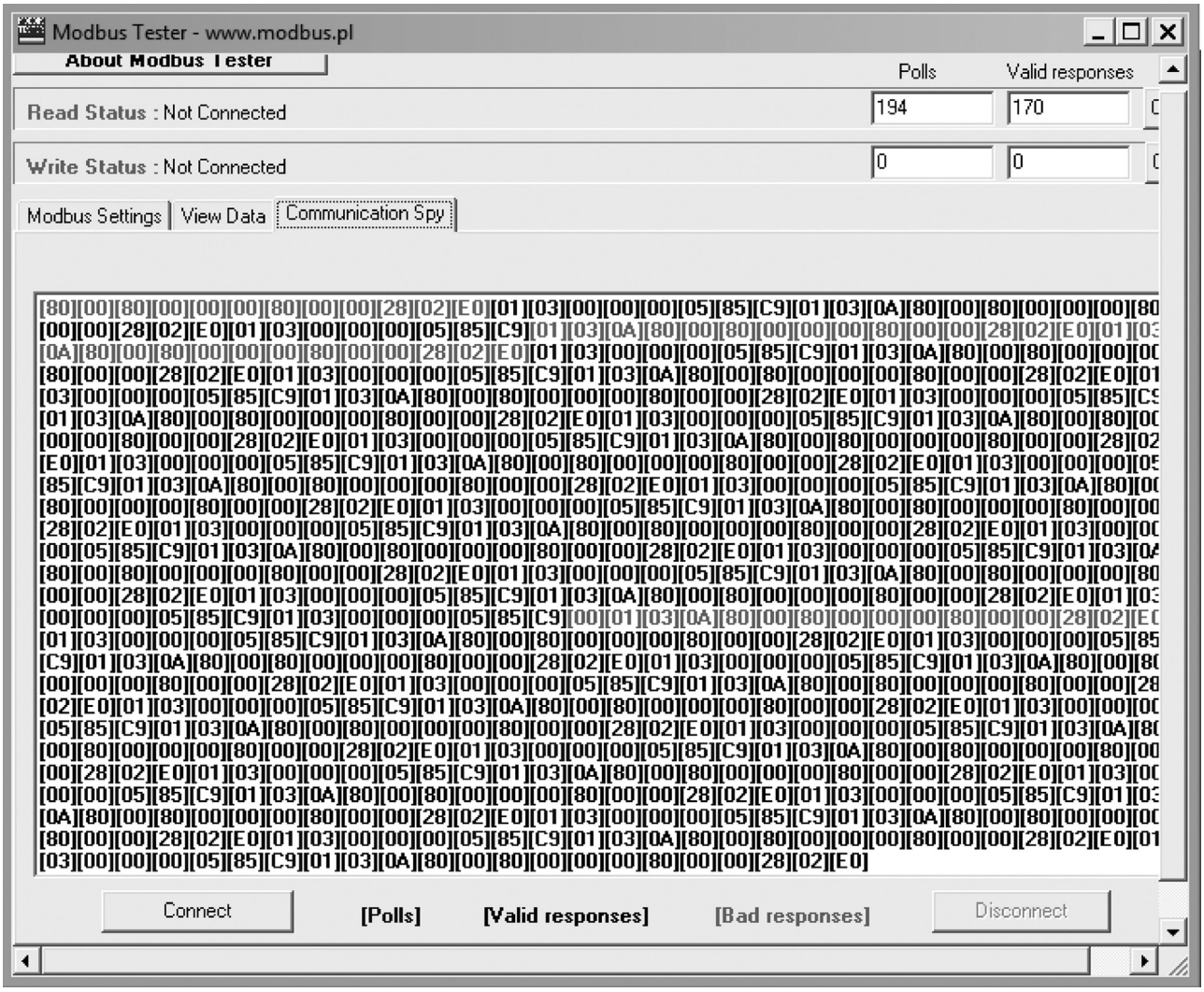


Fig. 14 – Transmission in the presence of noise using repeater/error correctors.

Future research will evaluate the performance of several candidate forward error correction codes and their implementations in repeater/error correctors. The results will enhance the detection and reconstruction of corrupted Modbus-RTU messages, leading to practical applications in a range of industrial control systems.

Acknowledgment

This research was supported by Proyectos Basales and the Vicerrectoría de Investigación, Desarrollo e Innovación (VRIDEI), Universidad de Santiago de Chile, Santiago, Chile.

REFERENCES

[1] M. Agrawal, Implementation of Reed–Solomon Error Correcting Codes, B.Tech. Thesis, Department of Electronics and Communication Engineering, National Institute of Technology, Rourkela, India, 2011.

[2] G. Cena, M. Cereia, I. Bertolotti and S. Scanzio, A Mod bus extension for inexpensive distributed embedded systems, *Proceedings of the Eighth IEEE International Workshop on Factory Communication Systems*, pp. 251–260, 2010.

[3] U. Demir and O. Aktas, Raptor versus Reed–Solomon forward error correction codes, *Proceedings of the International Symposium on Computer Networks*, pp. 264–269, 2006.

[4] Y. Fang, X. Han and B. Han, Research and implementation of collision detection based on the Modbus protocol, *Journal of Engineering Science and Technology Review*, vol. 6(1), pp. 91–96, 2013.

[5] N. Goldenberg and A. Wool, Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems, *International Journal of Critical Infrastructure Protection*, vol. 6(2), pp. 63–75, 2013.

[6] P. Karn, Reed–Solomon coding/decoding package, v1.0 ⟨www.piclist.com/techref/method/error/rs-gp-pk-uoh-199609/index.htm ⟩, 1996.

[7] T. Morris, A. Srivastava, B. Reaves, W. Gao, K. Pavurapu and R. Reddi, A control system testbed to validate critical infrastructure protection concepts, *International Journal of Critical Infrastructure Protection*, vol. 4(2), pp. 88–103, 2011.

[8] A. Ortega, PIC18/dsPIC30 Reed–Solomon (aaortega.blogspot.com/2010/06/pic18dspic30-reed-solomon.html), 2010.

[9] R. Phan, Authenticated Modbus protocol for critical infrastructure protection, *IEEE Transactions on Power Delivery*, vol. 27(3), pp. 1687–1689, 2012.

Please cite this article as: C. Urrea, et al., Implementation of error detection and correction in the Modbus-RTU serial protocol, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.07.001

[10] B. Sharma, Reed–Solomon error correction, *International Journal of Emerging Research in Management and Technology*, vol. 4(2), pp. 19–22, 2015.

[11] H. Shehab and W. Ismail, Hardware implementation for error correction using a software-defined radio platform, *European Journal of Scientific Research*, vol. 38(2), pp. 337–350, 2009.

[12] R. Solomakhin, P. Tsang and S. Smith, High security with low latency in legacy SCADA systems, in *Critical Infrastructure Protection IV*, T. Moore and S. Shenoi (Eds.), Springer, Berlin, Heidelberg, Germany, pp. 63–79, 2010.

[13] S. Vanstone and P. van Oorschot, *An Introduction to Error Correcting Codes with Applications*, Springer-Verlag, Boston, Massachusetts, 1989.

[14] Z. Yu, Z. Wu, H. Hong and L. Feng, Remote monitoring application based on Modbus and China Unicom 3G, *Proceedings of the IEEE International Conference on Intelligent Computing and Intelligent Systems*, vol. 2, pp. 445–448, 2010.

[15] X. Yuan and L. Ping, On systematic LT codes, *IEEE Communications Letters*, vol. 12(9), pp. 681–683, 2008.

[16] Y. Zhang, Reed–Solomon Error Correction Code and Modbus Communications Protocol, M.S. Thesis, Department of Applied Computer Technology, University of Electronic Science and Technology, Chengdu, China, 2010.

Please cite this article as: C. Urrea, et al., Implementation of error detection and correction in the Modbus-RTU serial protocol, International Journal of Critical Infrastructure Protection (2016), http://dx.doi.org/10.1016/j.ijcip.2016.07.001