The Fail Safe over EtherCAT (FSoE) protocol implemented on the IEEE 802.11 WLAN

Alberto Morato
CMZ Sistemi Elettronici s.r.l.
and *Dept. of Information Engineering*
University of Padova
alberto.morato@dei.unipd.it

Stefano Vitturi
National Research Council of Italy
CNR–IEIIT
stefano.vitturi@ieiit.cnr.it

Angelo Cenedese
Dept. of Information Engineering
University of Padova
angelo.cenedese@unipd.it

Giampaolo Fadel
CMZ Sistemi Elettronici s.r.l.
giampaolo.fadel@cmz.it

Federico Tramarin
Dept. of Management and Engineering
University of Padova
federico.tramarin@unipd.it

***Abstract*—Wireless networks are ever more deployed in industrial automation systems in various types of applications. A significant example in this context is represented by the transmission of safety data that, traditionally, was accomplished by wired systems. In this paper we propose an implementation of the Fail Safe over EtherCAT (FSoE) protocol on the top of IEEE 802.11 WLAN. The paper, after a general introduction of FSoE, focuses on the implementation of such protocol on commercial devices running UDP at the transport layer and connected via the IEEE 802.11 Wireless LAN. Then the paper presents some experimental setups and the tests that have been carried out on them. The obtained results are encouraging, since they show that good safety performance can be achieved even in the presence of wireless transmission media.**

***Index Terms*—industrial automation, WiFi, factory automation, functional safety.**

I. Introduction

The traditional safety systems of industrial plants, based on wiring, are gradually evolving [1]. Thanks to the development of safety communication protocols such as Fail Safe over EtherCAT (FSoE) and Profisafe [2], [3] the execution of safety functions and/or the request of safety-relevant data may now take place via industrial communication systems such as fieldbuses or real time Ethernet (RTE) networks, ensuring high safety integrity levels (for example SIL3 with residual error probability $< 10^{-9}/h$) [4]. In today's systems this opportunity clearly brings significant advantages in terms of reduced wiring complexity, and ease of maintenance [5].

However, in the era of Industry 4.0 and in particular of the Industrial Internet of Things (IIoT), several manufacturing units, whose functions are dependent on one another, can be physically located in different sections within the production area or even in distinct places. This means that due to the ever-larger size of the plants and their distributed locations, the cabling is increasingly complex and branched. Wires are difficult to install and to maintain, and therefore the possibility of replacing them with wireless reconfigurable links seems a very attractive opportunity [6], [7]. Although wireless networks have greatly improved, their performance figures are not yet comparable those of the wired industrial counterparts such as Profinet or Ethercat. Indeed, such networks are able to ensure cycle times below 100 μs [8]–[10], a value that (for the time being) can not be achieved by any industrial wireless system. However, industrial networks used for the transmission of safety data have, to a certain extent, distinct requirements from those of typical factory automation applications. Indeed, in a master-slave communication implemented by a safety protocol, it has to be ensured that safety data are transferred by the master without corruption, and that the response data from the slave are returned within a specified timeout, typically in the order of hundred of milliseconds. Such kind of performance figures can be provided by industrial wireless networks, in particular by the IEEE802.11 Wireless LAN [11]. This suggests that safety protocols can be, in principle, implemented over wireless networks, thus opening the field towards a further employment of wireless networks in the context of factory automation [12].

In this paper we consider such a topic and describe an example of practical implementation of the popular FSoE safety protocol over the IEEE802.11 WLAN. Specifically, we first provide some implementation details and then describe the results of some practical measurements carried out on some different experimental setups.

The paper is organized as follows. Section II gives a brief description of the FSoE protocol. Section III provides some details about the software structure of the protocol implementation. Section IV presents a theoretical analysis of the FSoE protocol. Section V describes the hardware devices chosen for the implementation of the protocol and reports the results of the experimental tests. Finally,

978-1-7281-0303-7/19/$31.00 ©2019 IEEE

Section VI concludes the paper and points out some future directions of research.

II. Basics of FSoE

FSoE [2] describes a protocol to transfer safety related data, encapsulated in Safety Protocol Data Units (PDUs), between two or more communication partners. The protocol is based on a master–slave relationship with a single device referred to as FSoE master and several FSoE slaves. The master–slave communication takes place via the so called FSoE Connections, which are established between the master and each slave.

1 B	1 B	2 B	2 B
CMD	Data	CRC	Conn. ID

Fig. 1. Basic FSoE frame

The shortest FSoE frame, shown in Fig. 1, comprises 6 bytes, which can carry 1 byte of safety data. The first byte contains the command to which a specific state of the FSoE connection corresponds and determines the meaning of the safety data. The Data field is followed by its own CRC and then by two bytes that report the Connection ID, which is an univocal number assigned by the FSoE master to each FSoE slave during the initialization phase. Notably, in order to ensure the lowest error probability, the CRC is elaborated in a different manner with respect to traditional protocols. Indeed, a device receiving a Safety PDU calculates the CRC on a more complex data structure, which includes 11 fields, as reported in Fig. 2.

Field Nr.	Field
1	received CRC (bit 0-7)
2	received CRC (bit 8-15)
3	ConnID (bit 0-7)
4	ConnID (bit 8-15)
5	Sequence Number (bit 0-7)
6	Sequence Number (bit 8-15)
7	Command
8	SafeData[0]
9	0
10	0
11	0

Fig. 2. Structure for CRC Calculation

Such a data structure contains two additional fields that identify the sequence number, namely a 16–bit counter ranging from 1 to 65535, which is re–initialized to 1 when the maximum value is reached. Each safety device manages its own sequence number, which represents the progressive number of the Safety PDU it transmits. The device also maintains the sequence number of the Safety PDU it expects to receive from its partner. Since these two values must coincide, the actual matching is verified by including them in the structure on which the CRC is calculated. Moreover, three additional zero octets are added to the CRC calculation as prescribed in [2]. The iterative procedure for the calculation of the FSoE CRC is reported in Algorithm 1, where t_1 is the pre-computed CRC table of $CRC(X)$ for the polynomial $P(z)$ with $X \in [0, 0xFF]$.

Algorithm 1: FSoE CRC calculation

$CRC_0 = 0$
for *each field listed in Fig. 2* **do**
$CRC_i = t_1[Field_i \oplus (CRC_{i-1} >> 8)] \oplus (CRC_{i-1} << 8)$
end
$CRC = CRC_i$

A more complex Safety PDU is used when more than one byte of safety data have to be transmitted. The structure of such PDU is reported in Fig. 3. As can be seen, it contains several blocks of two–byte data, each followed by its own CRC[1].

During normal operation of the protocol, the FSoE master sends a Safety PDU to the addressed slave and waits for the answer. Then, it moves to the next slave of the network. The state diagram of a FSoE slave is reported in Fig. 4.

The slave moves from the "Reset" state to "Session" and then to "Connection", where the FSoE connection is actually established, upon suitable commands received from the master. Subsequently, in "Parameter" the operational safety parameters are exchanged and, finally, the master sends a command to enter the "Data" state. In every state, an immediate transition to "Reset" may take place due to either a command received from the master or a problem detected by the slave.

In each device (master as well as slave), a watchdog timer monitors the FSoE communication cycle in order to detect possible delays on the network. If the FSoE master does not receive the answer from a queried slave within a specified time–out, then the watchdog timer of the master triggers the re-initialization of the FSoE connection with that slave. Conversely, if a FSoE slave is not queried by the master within a time–out, then the watchdog timer of the slave forces such device to enter the reset state. In this state, the slave waits to be re-initialized by the master. The FSoE master can handle several slaves by establishing a unique FSoE connection with each of such devices.

The communication medium, from the safety point of view, is seen as a black channel. With such an approach, safety applications and standard applications can coexist, sharing the same communication system at the same time [2].

The safety protocol encompasses procedures to systematically detect faults or errors that could occur during operation. The types of faults considered in this work are classified in Tab. I, which also lists the approach adopted

[1]The structure adopted for CRC calculation is modified accordingly.

1 B	2 B	2 B		2 B	2 B	2 B
CMD	Data_0	CRC_0	···	Data_i	CRC_i	Conn. ID

Fig. 3. Extended FSoE frame

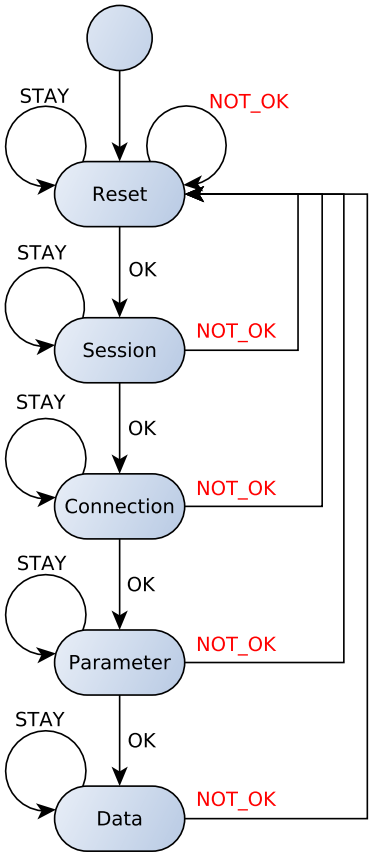


Fig. 4. FSoE state machine of a slave device

by FSoE to detect their occurrences. As an example, the corruption of a Safety PDU is detected by checking the CRC. Analogously, the loss is detected by a wrong sequence number as well as by the intervention of the watchdog timer. With the aforementioned measures, the FSoE standard ensures that the residual error probability P_{Res} can be kept below $10^{-9}/h$ which allows to achieve the highest safety performance, referred to as SIL3.

III. FSoE Implementation

The FSoE stack has been developed and implemented on three different types of devices, namely, a Personal Computer running the Linux operating system, a Raspberry Pi board [13], and an Espressif ESP8266 ultra low cost SoC (System on Chip) module [14]. All such devices have Wi–Fi interfaces and are equipped with the full TCP/IP protocol stack. The implementation of FSoE has

Fault	**Safety Measures**			
	Con. Id	**Seq. Nr**	**W.dog**	**CRC**
Corruption				X
Interchange	X	X		
Repetition		X		
Insertion		X		
Loss		X	X	
Delay			X	
Misrouting	X			

TABLE I
Safety measures adopted by FSoE to detect errors and faults

been made on top of the User Datagram Protocol (UDP). In practice, FSoE frames are encapsulated within UDP frames as described in Fig. 5.

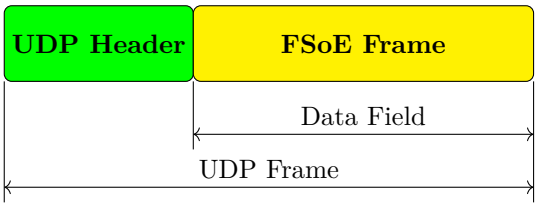


Fig. 5. Encapsulation of FSoE Frames within UDP Frames

With this type of implementation, the black channel comprises not only the communication medium, but also the Wi–Fi modules and the protocol stacks of both master and slave, up to the interface with the FSoE stack. As can be seen in Fig. 6, the transmission of safety data is started by a safety application running on the master that issues a request to the FSoE protocol stack. The arrival of the safety data is notified via an indication primitive to the safety application of the slave, which issues the response primitive carrying the response data. The service is then concluded with the arrival of the confirm primitive to the master. Clearly, if during the elaboration within a node (either master or slave) a packet corruption occurs, data are protected by the CRC of the FSoE protocol, which detects the problem.

The FSoE master has a protocol instance for each slave. The master assigns a Connection ID to this protocol instance and associates it with the IP address of the slave. In this way, the master establishes a one-to-one relationship with the slaves, so that a single UDP frame contains a single FSoE frame uniquely associated with a specific slave. This type of addressing allows to obtain a

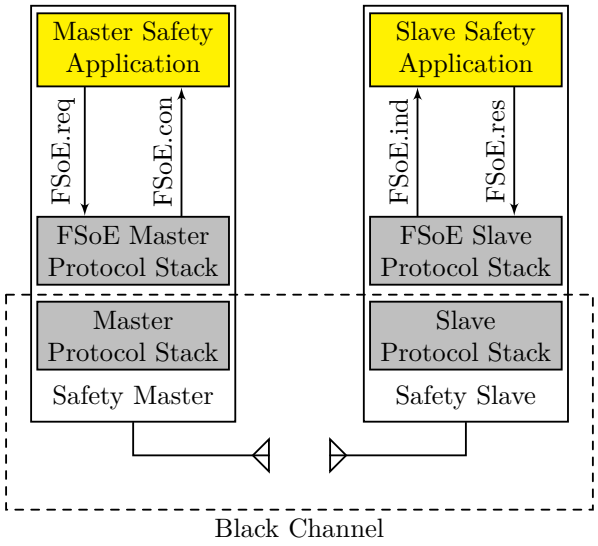


Fig. 6. Implementation of the FSoE Protocol Stack

double check of the correct addressing of the slave thanks to the correspondence between IP address and connection ID. Indeed, the slave knows both its connection ID and IP address, therefore, upon receiving a frame, it can check whether the addressing is correct. Similarly, upon receiving a frame, the master can immediately check the matching between IP address of the slave and connection ID. Also, the master can check the possible duplication of Connection IDs. If any mismatch is detected, the FSoE Master resets the FSoE Connection and forces a re-initialization.

As an alternative, a further way of exchanging safety data could be devised. With this solution, the safety data addressed to all slaves, prepared by the FSoE protocol, are delivered via a single broadcast UDP frame. Such a frame is composed of several fields, each of them associated with a specific slave. Then each slave answers with a unicast communication frame to the master. The access to the channel by the slaves may be regulated by a sequence specified in the frame sent by the master. In this way, each slave knows when it can transmit the response data. However, also leaving the slaves to access the channel in a random way, might be a viable strategy. Indeed, the FSoE protocol does not impose a strict deterministic procedure on data exchange but, rather, it requires that safety data arrives within a specified time–out. Although this alternative way has not been implemented, however it looks appealing from both a communication and a computational delay aspect, which are particularly important for resource constrained setups. Indeed, it dramatically reduces the number of exchanged UDP frames (for example, with N slaves, in the former case $2N$ UDP frames are required, whereas, with the alternative strategy, they could be reduced to $N + 1$); moreover, the overall elaboration time might reduced as well, since all slaves receive data from the master simultaneously, and then they can prepare the response data in parallel.

IV. Timing Analysis

In practical applications, the delivery of Safety PDUs from the master takes place cyclically with a period, T_{cycle}, determined by the requirements of the application itself. Since all the slaves must be queried within a period, the polling time of a single slave represents a meaningful index of the protocol performance.

A. Polling Time of a FSoE Slave

The polling time of a slave has been calculated as the time that elapses between the generation of a FSoE frame transmission request by the master and the reception of the confirm primitive from the slave. As can be seen in Fig. 6, the transmission sequence includes the times necessary to

- execute the FSoE protocol stacks in both master and slave;
- execute both master and slave standard protocol stacks;
- transmit the frame from master to slave and vice–versa.

Under the assumption that the stack execution times are the same for both master and slave, the polling time T_p can be expressed as

$$T_p = 2T_{FSoE} + 4(T_{stack} + T_{MAC}) + 2T_{tx} \qquad (1)$$

where T_{FSoE} is the time to execute the FSoE protocol stack, T_{stack} is the time to execute the protocol stack, T_{MAC} is the execution time the last two bottom layers of the protocol stack, and T_{tx} is the time to transmit a frame. It is worth remarking that the term relevant to the execution time of the FSoE stack in Eq. 1 is $2T_{FSoE}$ since the generation of both the response and confirm primitives is made automatically by the stack, i.e. the FSoE protocol stack is executed only once at both the master and the slave units.

B. Time Slot Allocation to FSoE Slaves

The FSoE protocol has been implemented assuming that each slave may enter/exit the network dynamically. This implies that, periodically, the master starts a discovery procedure to identify the N connected slaves. Then, the master assigns a time–slot to each slave determined as

$$t_{slot} = \frac{T_{cycle}}{N} \qquad (2)$$

In case the polling of a slave is carried out within t_{slot}, the master waits until the expiration of the whole t_{slot} and then moves to poll the next slave. Conversely, if there is a time–out (i.e. the polling has not been concluded within t_{slot}), the master forces the slave into the “Reset” state.

V. Experimental Results

Two different configurations have been adopted in the experimental sessions we carried out. The first one, referred to as "Basic Configuration" comprised one FSoE master and one FSoE slave, whereas, the second one, referred to as "Multiple Configuration" comprised three FSoE slaves. In both configurations, we measured the polling time of the slaves and the percentages of lost packets, which reflects on the number of FSoE connection re–initializations. The node acting as a master is configured as an IEEE802.11 Access Point (AP), whereas the slaves are configured as IEEE 802.11 Stations (STAs). In this way they can connect directly to the FSoE master.

A. Basic Configuration Results

In the Basic Configuration, two different setups were deployed operating in the 2.4 GHz band. In the first one, we used only ESP8266 modules and the wireless connection was implemented by IEEE 802.11g at 54 Mbits/s. In the second setup the FSoE master was a PC running the Linux operating system and the FSoE slave was a Raspberry Pi board. In this second case, IEEE 802.11n was used at a rate of 72 Mbits/s. The experimental sessions, in both configurations, lasted 5 hours and implied the transmission of more than 9,000,000 packets in total. The experiments were executed in an industrial environment (the laboratory of a factory), where interference from other networks was monitored and the IEEE 802.11 channel was selected in order to limit such a phenomenon. In both the experimental setups, nodes were located at the distance of 1 m. Fig. 7(a)-(c) report, respectively, the distribution and the cumulative distribution function of the polling time for the ESP8266 setup[2], whereas Table II (top) shows its statistics. Moreover, Table II (bottom) reports the performance in terms of lost FSoE packets.

Polling time $[\mu s]$				
mean	std	min	median	max
1484.67	1378.95	1051.00	1246.00	76004.00

Packet loss			
Total pkt	pkt lost	% pkt lost	test duration $[h]$
9169850	45	0.000491	5

TABLE II
Statistics for ESP8266 modules:
Polling Time (top) and Lost FSoE Packet (bottom)

Although there are no specific requirements for the polling time in the FSoE standard, the measured values can be considered as suitable for a wide number of applications. Conversely, the percentage of lost FSoE packets is definitely not tolerable, since the SIL3 requirement specifies as a necessary condition that at most one FSoE connection can be re–initialized once every 5 hours. In our experiments, 45 re–initializations occurred, since each lost packet corresponds to one re–initialization .

In order to investigate such a behavior we carried out additional tests, using the second setup in which, as mentioned above the two ESP8266 modules were replaced by, respectively, a PC and a Raspberry Pi board. The results of the experiments are listed in Fig. 7(b)-(d), which report the distribution and the cumulative distribution function of the polling time, whereas Table III, reports the polling time statistics and the percentage of lost FSoE packets.

Polling time $[\mu s]$				
mean	std	min	median	max
1773.28	2763.69	1096.52	1301.14	249887.00

Packet loss			
Total pkt	pkt lost	% pkt lost	test duration $[h]$
9584522	10	0.000104	5

TABLE III
Statistics for PC–Raspberry Pi set–up:
Polling Time (top) and Lost FSoE Packets (bottom)

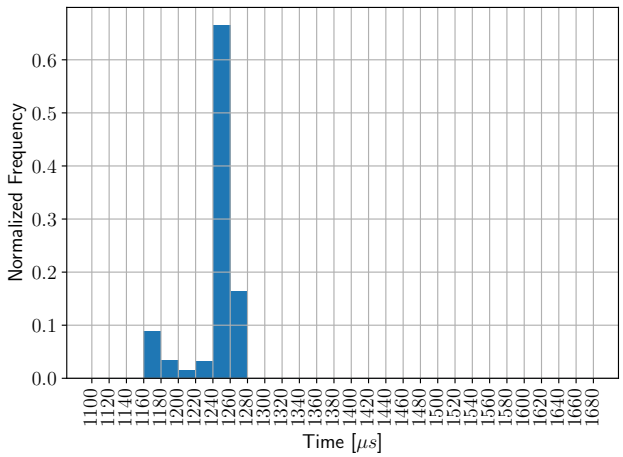
As it can be seen, the values of the polling time have slightly incremented with respect to the ESP8266 setup and the frequency distribution more spread out. This is likely due to the fact that, differently from the ESP8266, both the PC and the Raspberry Pi board use a general purpose operating system which may impact on the whole execution times of the implemented protocol stacks. Conversely, the percentage of lost packets is dramatically reduced. Such results allow to argue that the number of lost packets is mostly determined by the performance of the Wi–Fi modules, which are of a lower quality in the (cheap) ESP8266 modules.

These considerations are also confirmed by the lower panels of Fig. 7, (e)-(f) that report the polling time behavior over the experiment for, respectively, the ESP8266 and the Raspberry Pi setups. Here the timeout values that determine the lost packet fraction are also reported, thus proving the different behavior between the two configurations. In addition, the IEEE 802.11 protocol stack in the PC–Raspebrry Pi setup is configured to make use of the optional Request To Send/Clear To Send (RTS/CTS) mechanism: clearly this allows to reduce the frame collision during the transmission and thus to obtain a lower packet loss rate compared to the ESP8266 setup, in which this feature could not enabled.

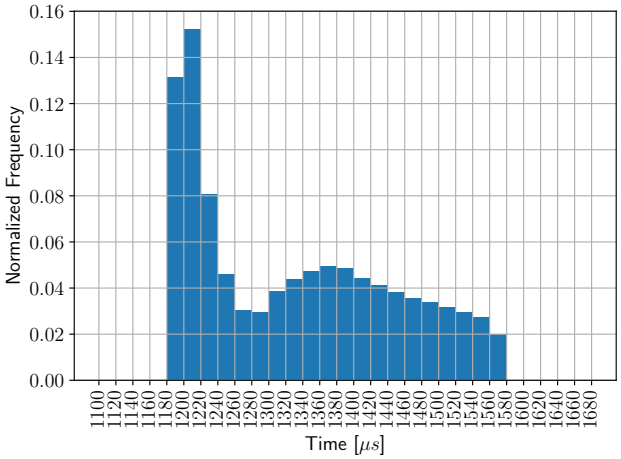
B. Multiple Configuration Results

To better assess the (relatively) poor performances of the ESP8266, a test with multiple slave has been carried out. In this configuration, only ESP8266 modules were

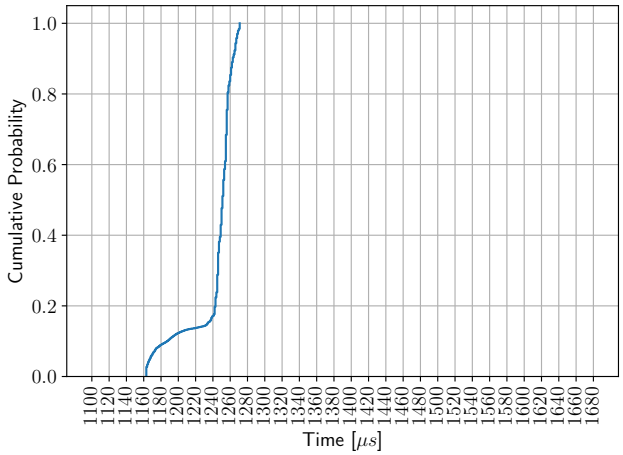
[2]To ease the readability of the plots and the comparison among different configurations, Fig. 7(a)-(b)-(c)-(d) report data comprised in the interval median ± one standard deviation. Also note that a different y-axis scale is used between (a) and (b), for the same reason.



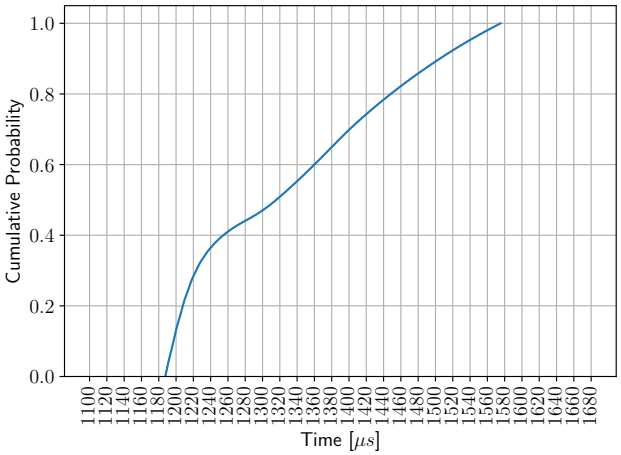
(a) ESP8266: Polling Time distribution



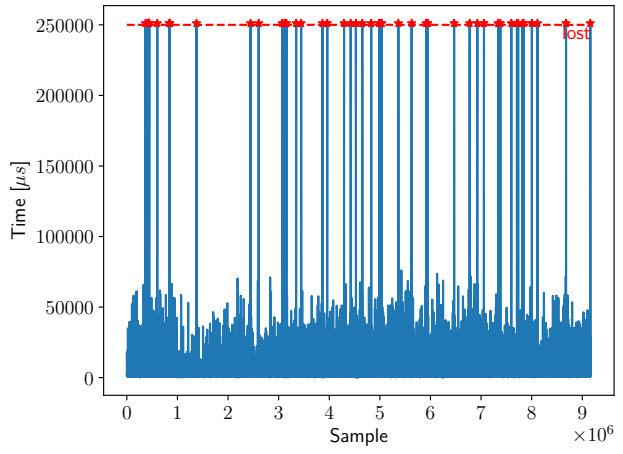
(b) Raspberry Pi: Polling Time distribution



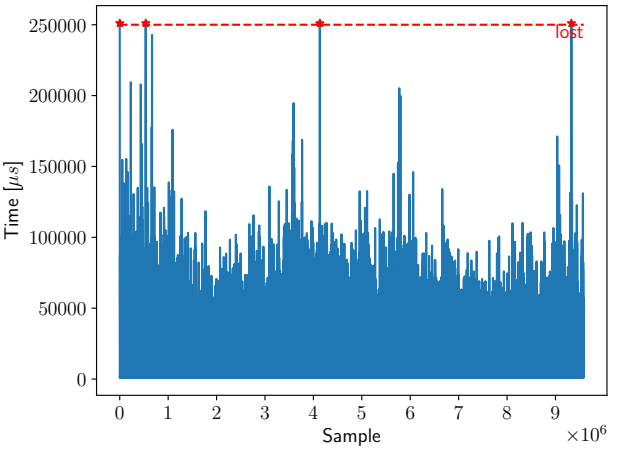
(c) ESP8266: Polling Time cumulative distribution function



(d) Raspberry Pi: Polling Time cumulative distribution function



(e) ESP8266: Polling Time evolution



(f) Raspberry Pi: Polling Time evolution

Fig. 7. Polling time distribution for the "Basic Configuration" scenario: (a) ESP8266 modules and (b) PC–Raspberry Pi setup. Polling time cumulative distribution function for the "Basic Configuration" scenario: (c) ESP8266 modules and (d) PC–Raspberry Pi setup. Comparison of polling time per sample, where the dashed red line represents the time-out value and star markers highlight lost packets: (e) ESP8266 modules and (f) PC–Raspberry Pi setup.

used and the network comprised one FSoE master and three slaves.

The results of the experimental measurements are listed in Table IV, which reports the statistic of the globally lost packets as well as the details for each FSoE slave.

Packet loss			
	Total pkt	pkt lost	% pkt lost
Global	9713527	2206	0.022710
Slave 0	9713527	567	0.005837
Slave 1	9713527	894	0.009203
Slave 2	9713527	745	0.007669

TABLE IV
LOST FSOE PACKET STATISTICS FOR MULTIPLE ESP8266 MODULES

It is evident that with multiple slaves the percentage of lost packets is even greater than that of the single slave setup, which may be due to the higher demand, in terms of system resources, to handle multiple slaves, confirming the inadequacy of such modules for a safety application.

Clearly, even in the PC – Raspberry Pi setup, the percentage of lost packets is not acceptable by FSoE. However, some techniques to mitigate such a problem are available. For example, the transmission speed could be reduced, since lower rates adopt more robust modulations. Another possibility relies on the use of rate adaptation techniques. In this case, the transmission speed is selected according to the quality of the wireless link. Since the purpose of this work is to experience the wireless capability of the FSoE and its limitations, the implementation of such features is left to future work.

On the other hand, it is worthwhile to note that, although the tested configurations reveal not strictly compliant with the regulations, nonetheless it has been proved that it is possible to transfer successfully safety PDUs based on FSoE using IEEE 802.11 WLAN. This actually suggests that these implementations can be used in those less demanding applications where a connection re–initialization that occurs more frequently than once every 5 hours is acceptable. Anyway the applicability of the proposed architecture must be evaluated and validated through safety assessments case by case.

As a further observation, Table V reports the execution time of the FSoE stack protocol for the ESP8266 module and the Raspberry Pi board respectively. As can be seen, thanks to the larger computational capability, the Raspberry Pi board is able to elaborate a FSoE frame more quickly with respect to the ESP8266 module. Anyway, for both setups the FSoE stack protocol execution time gives only a minor contribution to the polling time that, for the most, is due to the execution of the protocol stack.

Finally, considering the statistics of the polling time distribution for both the ESP8266 and Raspberry Pi setups, we aim to find the the minimum time slot in order to allow at most one lost packet every 5 hour. Let be N_{frame}

FSoE execution time [μs]					
	mean	std	min	median	max
ESP8266	50.63	3.06	47.00	50.00	111.00
Rasp.Pi	1.28	1.90	0.34	0.94	140.89

TABLE V
FSOE PROTOCOL STACK EXECUTION TIME
FOR THE ESP8266 AND THE RASPBERRY PI BOARDS

the number of frames exchanged in 5 hours. We want to calculate t_{slot} such that

$$P[T_p > t_{slot}] = 1 - P[T_p \leq t_{slot}] \leq \frac{1}{N_{frame}} \qquad (3)$$

where T_p is the polling time and t_{slot} is the time slot. It results that the minimum time slot t_{slot} has to be 8621 μs and 16084 μs respectively for the ESP8266 and Raspberry Pi setups. Hence, with a cycle time of 250000 μs it is possible to connect at most 28 slave devices to the ESP8266 master device and 15 devices to the Raspberry Pi setup.

If we consider Eq. 1, a further evaluation of the experimental timings can be drawn. The transmission time T_x (in $[\mu s]$) can be obtained from [15] as

$$T_x = 20 + 4\left\lceil 2\frac{36 + 2.75 + l}{r}\right\rceil + 6 \qquad (4)$$

where $l = 43$ byte is the data length presented to the 802.11 layer and r is the data rate (in $[Mbit/s]$), which is $54Mbit/s$ for the ESP8266 setup and $72Mbit/s$ for the Raspberry Pi setup: it results $T_x = 38.11$ μs and $T_x = 30.54$ μs for the two setups, respectively. Giving that for the ESP8266 t_{stack} has been calculated experimentally in [16] as equal to 48 μs, and for the Raspberry Pi it results 35 μs according to [17], it follows that the T_{MAC} value is around 220 μs and 270 μs for the two considered configurations. These preliminary values are currently under further investigation and assessment.

VI. CONCLUSION

In this paper we have addressed the implementation of Fail Safe over Ethercat (FSoE) on the IEEE 802.11 WLAN.

We initially provided a general description of the FSoE protocol with a particular focus on the safety measures for faults recognition. Subsequently, we presented the implementation of FSoE on the top of the User Datagram Protocol (UDP) and two addressing technique for the safety slave devices: the first one uses an one-to-one association between the IP address and the connection ID and where safety slaves are polled one at a time; with the second technique, the safety master addresses all the safety slave with a single broadcast transmission allowing to reduce the number of exchanged UDP frames. Based on the first technique, a theoretical analysis of the polling time has been carried out.

The experimental campaign on three different setups (single safety slave ESP8266 module, PC to Raspberry Pi board, and multiple safety slave ESP8266 modules) has evidenced similar results for the polling time, which results about 1.5 ms. On the other hand, packet loss results rather different for the three setups and, more importantly, does not comply with the limit imposed by the regulation in none of the experimentsp. In particular, at the best, a ratio of 10 lost packet every 5 hours has been achieved, while the regulations impose a maximum rate of 1 lost packet every 5 hours.

In this context, it can be pointed out that the proposed setups of wireless-FSoE can be used by less demanding application scenarios. However, current research activities are focusing on the introduction of suitable techniques, e.g. rate adaptation strategies, to improve the quality of the wireless links and minimize the packet loss rate, so that the full specifications for safety applications can be met.

References

[1] S. Vitturi, C. Zunino, and T. Sauter, "Industrial communication systems and their future challenges: Next-generation ethernet, IIoT, and 5G," *Proceedings of the IEEE*, vol. 107, pp. 944–961, June 2019.

[2] "Industrial communication networks - Profiles - Part 3-12: Functional safety fieldbuses - Additional specifications for CPF 12," standard, International Electrotechnical Commission, 2010.

[3] "Industrial communication networks - Profiles - Part 3-3: Functional safety fieldbuses - Additional specifications for CPF 3," standard, International Electrotechnical Commission, 2016.

[4] "IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems," standard, International Electrotechnical Commission, 2016.

[5] "PROFIBUS Nutzerorganisation e.V." https://www.profibus.com/technology/profisafe/benefits/. [Online; accessed 12-June-2019].

[6] A. Frotzscher, U. Wetzker, M. Bauer, M. Rentschler, M. Beyer, S. Elspass, and H. Klessig, "Requirements and current solutions of wireless communication in industrial automation," in *2014 IEEE International Conference on Communications Workshops (ICC)*, pp. 67–72, June 2014.

[7] T. K. Refaat, R. M. Daoud, H. H. Amer, and E. A. Makled, "WiFi implementation of wireless networked control systems," in *2010 Seventh International Conference on Networked Sensing Systems (INSS)*, pp. 145–148, June 2010.

[8] D. Orfanus, R. Indergaard, G. Prytz, and T. Wien, "EtherCAT-based platform for distributed control in high-performance industrial applications," in *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, pp. 1–8, Sep. 2013.

[9] J. Robert, J.-P. Georges, E. Rondeau, and T. Divoux, "Minimum cycle time analysis of Ethernet–based real-time protocols," *International Journal of Computers, Communications and Control*, vol. 7, no. 4, pp. 743–757, 2012.

[10] G. Prytz, "A performance analysis of EtherCAT and PROFINET IRT," in *International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 408–415, IEEE, 2008.

[11] "IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pp. 1–3534, Dec 2016.

[12] S. Vitturi, A. Morato, A. Cenedese, G. Fadel, F. Tramarin, and R. Fantinel, "An innovative algorithmic safety strategy for networked electrical drive systems," in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, pp. 368–373, IEEE, 2018.

[13] "Raspberry Pi Foundation." https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf. [Online; accessed 26-June-2019].

[14] "Espressif Systems." https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. [Online; accessed 26-June-2019].

[15] F. Tramarin, S. Vitturi, M. Luvisotto, and A. Zanella, "On the use of ieee 802.11 n for industrial communications," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, pp. 1877–1886, 2016.

[16] I. C. Bertolotti and T. Hu, "Real-time performance of an open-source protocol stack for low-cost, embedded systems," in *ETFA2011*, pp. 1–8, 2011.

[17] G. Chuanxiong and Z. Shaoren, "Analysis and evaluation of the TCP/IP protocol stack of Linux," in *WCC 2000-ICCT 2000. 2000 International Conference on Communication Technology Proceedings (Cat. No. 00EX420)*, vol. 1, pp. 444–453, 2000.