A Diversity-Based Approach for Communication Integrity in Critical Embedded Systems

Amira Zammali, Agnan de Bonneval, Yves Crouzet

A diversity-based approach for communication integrity in critical embedded systems

Amira Zammali[(1),(2)]
[(1)] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[(2)]Univ of Toulouse, UPS, LAAS, F-31400, Toulouse, France
Email: zammali@laas.fr

Agnan de Bonneval[(1),(2)]
[(1)] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[(2)]Univ of Toulouse, UPS, LAAS, F-31400, Toulouse, France
Email: agnan@laas.fr

Yves Crouzet[(1),(2)]
[(1)] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[(2)]Univ of Toulouse, LAAS, F-31400, Toulouse, France
Email: crouzet@laas.fr

***Abstract*—We present, in this paper, a fault-tolerant approach to cope with accidental communication data corruption in critical embedded systems. One of the classical integrity approaches is the redundancy-based approach that consists particularly in replicating the message and sending all copies via the same communication channel consecutively or sending them via replicated communication channels. Yet, such approach is vulnerable to some cases of Common-Mode Failure. So, we propose to diversify the copies to be sent by diversifying either the error detection function (which generate the check bits) or the data payload. This paper focus on the first proposal by presenting experiments and results to validate its effectiveness. Besides, it describes basic theoretical concepts of the second proposal. Our case study is the Flight Control System (FCS). Yet, our approach could be deployed in other systems for which we describe the key properties.**

Keywords*—*fault tolerance; diversity; communication integrity; critical embedded systems; flight control system.

I. INTRODUCTION

In critical embedded systems, it is extremely important to enhance the system safety. In fact, the occurrence of failures (e.g., a computer failure) could lead if not detected and recovered to damages and even fatalities. To meet safety requirements and since systems are too complex and operate in uncertain environments, many designers rely on fault tolerance means in addition to fault avoidance, fault elimination and fault forecasting techniques. As defined in [1], fault tolerance is aimed at avoiding service failures despite the occurrence of faults, based in particular on redundancy. For example, to tolerate faults in a system (or in some parts of the system), one safety measure is to duplicate data (to be stored or transmitted), software components (e.g., operating systems) or hardware components (e.g., sensors and network nodes) of the system. Thus, the system could continue to operate and to fulfil its functionalities while some of its components do not operate correctly.

In this paper, we address a particular aspect of the system safety which is communication integrity. Indeed, the occurrence of corruption in exchanged data could result in catastrophic events particularly if corrupted messages are not detected and recovered.

The occurrence of errors in exchanged data is due to several factors namely noise and failures in network nodes (edge or interstage). To enhance the communication integrity, a common classical approach consists in the transmission of several similar copies of the same message in different refresh cycles [2] (called temporal redundancy) or via duplicated communication channels [2] (called spatial redundancy). These latter methods belong to the class of "symmetric redundancy" which is vulnerable to the problem of "Common-Mode Failure". This problem makes replicated data likely to exhibit the same errors (repetitive errors). Since our goal is to cope with multiple repetitive random errors, we propose to adopt alternatively an "asymmetric redundancy" approach that consists in diversifying the message copies to be sent. To enhance communication integrity, we explore, in this paper, the latter idea taking inspiration of existing "asymmetric redundancy techniques", that are aimed at enhancing software dependability based on diversity [2] [3]. Such techniques are generally based on the traditional concept of "Design Diversity" that is sometimes recommended to tolerate software design faults [4]. We aim at proposing a diversity-based fault-tolerant integrity approach to be deployed in the application layer in order to cover all residual errors occurring in the under-layers .

To diversify message copies, we consider both payload and check bits (added extra-data to detect errors) parts of the message. We propose two diversity-based policies. A policy based on a concept called "data diversity" [5] [6] and a policy consisting in using a multi-function error detection policy whom we describe the basic concepts in [7] and which was inspired from the works described in [8]. The former proposal (Data Diversity based Policy DDP) consists in sending copies of the message comprising differently re-expressed data (generated by different re-expressing functions). An error is detected if restored data of different received copies do not match. The latter proposal (Multi-Function error Detection Policy MFDP) suggests the use of a different error detection function for each transmitted message. So, each message has its own different check bits. The different used error detection functions must be complementary in terms of error detection capability in order to improve the error detection performance.

Our intent in this paper is to present both proposed policies. Yet, the emphasis is on describing the Multi-Function error Detection Policy by presenting simulations and results in order to prove and validate its effectiveness. We first present some background by describing the characteristics of targeted critical embedded systems in Section II. Section III discusses the com-

munication integrity concerns in such systems. It presents the techniques usually adopted to cope with data corruption and describes characteristics of commonly used network technologies. Section IV introduces the Multi-Function error Detection Policy. Then, Section V details the case study of a Flight Control System (FCS) to illustrate this approach. Section VI presents the MFDP validation by describing simulations and results. Section VII introduces basic concepts of our ongoing work on the Data Diversity based Policy. Finally, section VIII concludes this paper.

II. Key properties of targeted critical embedded systems (CES)

Defining an efficient strategy for ensuring communication integrity is closely dependent on the specific features of the investigated systems. In this section, we describe the key properties of our targeted systems. We start by presenting generic features that impact the communication integrity strategy to be adopted. The second part of this section is devoted to describing different redundancy strategies which are deployed in critical embedded systems to enhance communication integrity.

A. High criticality, limited resources and complex networks

Targeted critical embedded systems (CES) are defined by strict safety requirements. In fact, the criticality level of a system is judged according to the consequences of potential failures. If failures could result in fatalities or significant damage (material or environmental), this system is safety-critical. Safety-critical systems are used in different application areas (e.g., railways, aircraft flight control and nuclear systems). Typically, in such systems, the failure rate must be less than a threshold to be set according to the system specificity (10^{-9} to 10^{-7} failures per hour of system service). To meet such strict safety requirement, designers often rely on redundancy.

Embedded systems are based on limited resources. Indeed, we particularly focus on the fact that many embedded systems do not include huge amounts of resources (e.g., memories, processors and power supplies). Moreover, they are usually based on short communication messages. So, one cannot rely, in such systems, on an error detection policy with high overheads and computational costs. Redundancy is deployed in such systems to enhance safety but given the limited resources, it must be optimized.

Finally, communications in our targeted systems are based on complex networks including a large number of nodes (e.g., calculators, actuators and sensors). There are not only sources and sinks but also intermediate communication nodes (interstage nodes) which are active. The "active" feature means that the node has memory and processing capabilities and its role is not just relaying data messages. This characteristic increases the occurrence likelihood of repetitive errors (e.g., systematic errors) leading to Common-Mode Failure (due e.g., to defective registers). We focus, in this paper, on these latter types of errors.

B. Redundancy

Three redundancy types deployed in critical embedded systems can be distinguished: spatial, temporal and value redundancy. Theses different kinds of redundancy can be either symmetric (the replication of the same system part) or asymmetric (the diversification of the replicated part). We focus here on symmetric redundancies that are commonly used in CES.

Spatial redundancy: Spatial redundancy consists in deploying (locally or in a distributed manner) multiple copies of the same system component [2]. For example, the use of redundant sensors to collect the data. Thus, even when one of any sensors produces an erroneous data or does not produce data (in the case of a failure), a correct copy of data could be available. In the context of communications, it consists in either using multiple communication channels or multiple interstage nodes. This type of redundancy allows to handle potential physical failures and tolerate the loss of system components.

Temporal redundancy: Temporal redundancy is defined abstractly by the fact to do the same thing several times [2]. It is usually used in critical systems to enhance communication integrity [3]. This technique consists in transmitting several copies of the same message over the same communication channel at distinct time instants [3]. This repetitive transmission is either done systematically or triggered by the occurrence of errors.

Value redundancy: The value redundancy consists in adding extra-data to the data to be sent in the message [2]. The extra-data is commonly called "control data" or "check bits". It is typically generated by error detection and/or correction codes, hash functions or a cryptologic methods in order to detect and/or correct potential errors on data.

We focus in the rest of this paper on error detection codes and particularly those used in critical embedded systems (described later). In fact, our multi-function error detection policy is based on such codes to cope with data corruption.

III. Communication integrity in CES

This section presents an overview of the communication integrity concerns in critical embedded systems. It describes first some used error detection codes. Then, this section presents commonly used embedded network technologies and details the error detection techniques they employ.

A. Common error detection codes in CES

We present here common error detection techniques used to enhance data integrity. Such techniques can be used at lower communication layers or even at the application level.
We have considered these techniques as examples to design and illustrate the diversity-based fault tolerant communication approach presented in this paper.

Parity check codes: This technique consists in adding a one bit as extra information. This parity bit is used to check even or odd parity (e.g., for odd parity, it must be equal to 0 if the number of 1-bits of the data field is an odd number). This technique is convenient for systems where the probability of occurrence of errors is very low since it detects only an odd number of erroneous bits.

Arithmetic checksums: This technique consists in adding bytes of extra-information calculated by an arithmetic formula. Fletcher and Adler checksums are commonly used in critical embedded systems. Fletcher is calculated by summing the data bytes modulo $2^n - 1$ such that n is the size of the checksum (the number of extra-bits). Adler checksum is calculated by summing the data bytes modulo the first prime number inferior to 2^n such that n is the size of the checksum (the number of extra-bits). On the receiver side, the checksum of the received data is calculated then result is compared to the received checksum. If they do not match, an error is detected. Compared to parity code, arithmetic checksums have better error detection capabilities but require a higher redundancy in terms of check bits and their use increases the computational cost.

Cyclic Redundancy Codes: A CRC code is defined by its generator polynomial $G(x)$. The check bits are the rest (the remaining polynomial) $R(x)$ of the division of the polynomial corresponding to the data bits $Q(x)$ by the generator polynomial $G(x)$ ($R(x) = Q(x) div G(x)$). On the receiver side, the same division is made then the result is compared to the received CRC field. If they do not match, an error is detected. This technique is rather carried out in the hardware by applying shift registers and bitwise XOR. CRC codes provide a high detection performance compared to parity check codes and arithmetic checksums. In all cases, CRCs detect all single bits and all burst errors with a size up to its generator polynomial degree and all double bit errors if the generator polynomial $G(x)$ has at least three terms and odd numbers of bit errors if $G(x)$ has a factor of $(x + 1)$.
We call CRC-x a CRC whose generator polynomial degree is equal to x and x is also the number of check bits.

B. Common network technologies in CES

We present here commonly used network technologies in critical embedded systems in order to clarify the specificity of embedded technologies used in real critical applications.

Controller Area Network CAN: CAN is a flooding bus based on CSMA/CR (Carrier Sense Multiple Access with Collision Resolution) technique. The size of exchanged messages can reach a maximum of 135 bits including the payload data, the eventual resynchronizing bits and the bits of the identifier which are equal to 11 for the standard version of CAN and equal to 29 for the extended version. CAN relies on a set of error detection mechanisms. In particular, it uses CRC error detection code to cope with data corruption. For CAN 2.0, the used CRC is defined by the following generator polynomial: $G(x) = x^{15}+x^{14}+x^{10}+x^8+x^7+x^4+x^3+1$. Moreover, CAN uses the stuffing technique which consists in adding a 0-bit (respectively 1-bit) after each 5 consecutive 1-bits (respectively 0-bits).
So, an error is detected if the received data includes 6 similar consecutive bits. It relies also on verifying the frame structure in order to track out data corruption.

FlexRay: FlexRay is a fault-tolerant protocol which can be based on a bus topology, a star (simple or multiple) topology or a combination of different topologies. It supports payload data from 8 bytes up to 254 bytes. The FlexRay message is composed of 5 bytes of header, data payload and a set of CRC bits to enhance data integrity. It supports two different CRCs. The first one is a CRC-11 used to protect exclusively the header data. The second one is a CRC-24 used to protect the header and the payload data. The FlexRay protocol provides increased bandwidth over CAN and also incorporates new mechanisms to increase communication fault tolerance and determinism while retaining a high degree of flexibility [9]. FlexRay is based on local bus guardian that was designed to be a simple temporal enforcement mechanism and has evolved to become in current implementation more complex having the same capabilities as a communication controller [9] and proving high error detection performance.

Local Interconnect Network LIN: LIN is a protocol for embedded networks which is based on a serial bus and is characterized by a low cost and low complexity. The message size of LIN could reach 8 bytes. To cope with data corruption, LIN is based on a 8-bit arithmetic checksum [10].

Time-Triggered Protocol TTP: TTP is based on a duplex star bus based on TDMA (Time Division Multiple Access). It is based on CRC-16 in order to enhance data integrity. TTP deploys an hybrid redundancy (spatial and temporal) consisting in a nodes duplication. This redundancy is managed by a Fault Tolerant Unit (FTU) such that each node sends a copy of the same data in each TDMA round. Another possibility of redundancy offered is the deployment of passive redundant nodes that take the place of the principal node in the case of a failure. This redundancy is called passive redundancy.

ARINC 429: ARINC 429 is based on simplex transmission (a communication channel allows to transmit data in one direction). Bidirectional transmission requires two channels or buses. ARINC 429 is based on end-to-end topology. It is based on parity code to enhance communication integrity. The size of exchanged messages is equal to 32 bits.

IV. The Multi-Function error Detection Policy (MFDP)

Traditional deployed redundancy consists in a simple replication of data or system software or/and hardware components. This approach could deal with the problem of system failures in many cases. It makes possible to tolerate failures by providing a back-up that keeps the system fulfilling its functionalities despite the fact that there are some components that are failed or in the presence of corrupted data. Yet, this does not deal with the problem of Common-Mode Failure.

Thus, we propose to substitute traditional symmetric redundancy with an asymmetric redundancy. This means diversifying the redundant parts of the system. We introduce here the proposed Multi-Function error Detection Policy to enhance communication integrity.

The multi-function error detection policy we propose is inspired from preliminary work described in [8] for which we present here a significant extension. Our idea consists in exploiting the symmetric redundancy that already exists in targeted systems. We focus on spatial or/and temporal redundancies that result in transmitting N copies of each message (N to be set according to the system specificities). These N copies are sent either in the same refresh cycle via

different communication channels (spatial redundancy) or in different refresh cycles via the same communication channel (temporal redundancy). A refresh cycle is the time between two successive transmissions. This duration may either be a design choice, or a design constraint imposed by the application or the system.

Despite the fact that sending multiple messages copies seems to be restrictive, we can notice that many systems transmit messages repetitively to enhance communication dependability (e.g., via duplicated channels described in [9]) by dealing with the problem of data corruption, message loss or links failures.

In addition to redundancy, we focus on error detection codes. Considering commonly used network technologies in critical embedded systems (see Section III), regardless of the code used, they all use always the same principle: a unique function is used for all messages to be sent (e.g., parity bit in all transmitted messages in ARINC 429). Each error detection function is vulnerable to some particular kinds of error patterns (e.g., even errors for parity code, errors that are multiple of the generator polynomial for CRC). In this case (when the code is vulnerable to the considered error) and even though the message is sent N times, the error will never be detected.

Thus, we propose to use a multi-function error detection policy based on a set of complementary error detection codes to minimize the likelihood of the problem of Common-Mode Failure. Different error detection codes have different weaknesses and are vulnerable to different error patterns. Thus, the idea behind diversifying the error detection function is to increase the likelihood to detect at least one erroneous copy of the message subjected to a repetitive error. In the proposed multi-function error detection policy, we do not consider the occurrence of one erroneous undetected message as a dangerous event to be avoided, but rather the case to avoid is when all N erroneous copies of the message are not detected. Thus, our goal is to detect the error at least once. Then, the system could be notified and it could be able to recover the error according to the deployed recovery strategy.

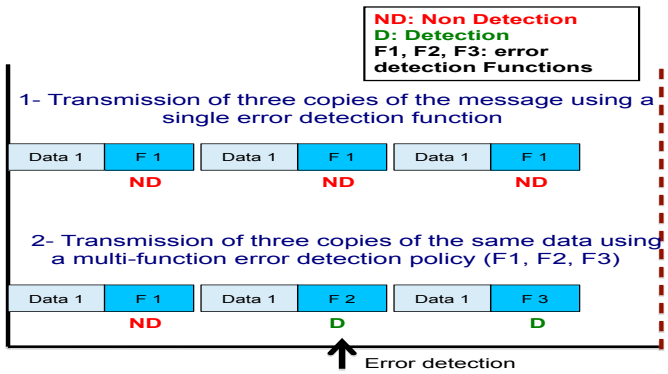


Fig. 1. A multi-function error detection policy (F1, F2, F3)

Fig. 1 gives an overview of our proposed multi-function error detection policy. In this example we assume that $F1$ is vulnerable to the error to which all copies are subjected. $F1$ does not detect error but $F2$ and $F3$, being different and not vulnerable to the error, detect it. This shows that the proposed multi-function policy increases the detection capability. It is noteworthy that we take into account in MFDP these two specific assumptions: i) the policy is implemented at the application layer so it is independent of the network architecture and ii) communication channels are binary (two symbols 0 and 1) and symmetric (0 and 1 bits have the same probability to be erroneous).

The question now is how to select the set of different codes to be used. The selection of such functions is not an easy task. It is a trade-off of the following criteria.

Computational cost: given that we target critical embedded systems, the adopted integrity approach should have an acceptable low computational overhead. In fact, trends now aim to speed up heavy error detection codes in order to make them relevant for small embedded processors (e.g., using lookup tables for CRC codes).

Intrinsic detection performance: to select relevant error detection codes, we rely mainly on their intrinsic error detection capabilities. This means that we choose codes that are proven to be efficient in systems sharing similar features with our targeted ones (e.g., message size, criticality requirements). We do not rely on standards or conventional classifications but rather on works evaluating error detection codes.

Complementarity property: the most difficult challenge of our policy is how to find complementary codes. Let us define this property: two error detection functions are said to be complementary if the permanent error is not detected by the first error detection function, there is a chance that it will be detected by the second function. This can be formalized as follows:

$(F1\ and\ F2\ are\ complementary)\ iff\ (ND1 \cup ND2 \neq ND1\ and\ ND1 \cup ND2 \neq ND2)$, where $F1$ and $F2$ are two error detection functions. $ND1$ and $ND2$ are the set of errors not detected by $F1$ and $F2$ respectively.

V. Case Study: Flight Control System (FCS)

To illustrate the proposed MFDP, we consider a Flight Control System (FCS) as a case study. We start by describing the main relevant features of an FCS. Then, we explain the specificity of the temporal redundancy existing in FCS. Finally, we detail how to apply our approach to FCS. It is noteworthy that our approach could be deployed in other case studies.

A. Description of the Flight Control System (FCS)

We consider FCS as a case study since it is an industrial application studied in the context of a past project involving our research team and an aerospace company. FCS controls the airplane trajectory by exchanging control-command messages with the actuators controlling the movable surfaces. Commands are calculated periodically and sent via a digital network (Fig. 2).

A Flight Control System (FCS) is a critical systems characterized by a set of strict safety requirements, including integrity requirements. It relies on the fly-by-wire technology and the control is made by a set of control-command signals transmitted by Flight Control Computers (FCC) to the controlled actuators. Digital networks make communications

faster and more accurate but increase the vulnerability in communications. They increase particularly the probability of occurrence of permanent errors since interstage nodes have memory and processing capabilities. Hence, memory faults could induce permanent errors in all messages memorized and relayed by the considered interstage nodes.

In civil aviation, events are classified according to their severity which is defined by the level of damages caused by the failure occurrence. The most common classification [11] defines four severity levels which are "Minor", "Major", "Hazardous" and "Catastrophic". Since erroneous control-command messages could cause, if not detected and recovered, a system failure (crash), the occurrence of erroneous and undetected messages is considered as "Catastrophic" and its rate must be less than 10^{-9} per hour.

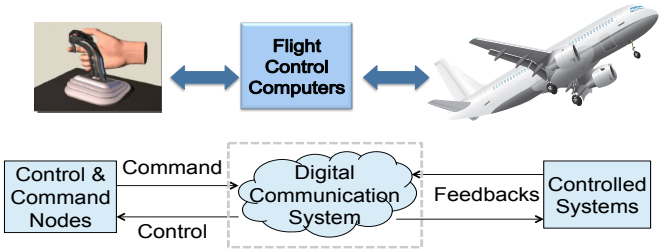


Fig. 2. The targeted part in the Flight Control System

B. FCS: a slow-dynamic system with temporal redundancy

Before considering the example of Flight Control System, we introduce here the definition of a slow-dynamic system [7]. In fact, we classify systems into two classes (as shown in Fig. 3): i) fast-dynamic; ii) slow-dynamic. The "slow-dynamic" property means that the duration of the interval between two successive significant changes (a significant change means for example the variation of the value of data to be transmitted) is much larger than the refresh cycle duration (the refresh cycle is the duration between two successive transmissions). So, almost the same data (the same value) is transmitted during a set of refresh cycles before a significant change occurs. In the example of Fig. 3, the same message is sent several times. Yet, in fast-dynamic systems, the duration of the interval between two successive significant changes is very close to the duration of the refresh cycle so the message can be sent only once.

The Flight Control system belongs to the class of slow-dynamic systems. In fact, airplane surfaces are designed to move slowly and the result of the trajectory calculation remains the same throughout a set of calculation refresh cycles before that the result changes. Thus, the same control-command message with the same data values is sent during this time. This behaviour is a kind of temporal redundancy where the same message is sent several times in different refresh cycles.

VI. MFDP Validation

To define a multi-function error detection policy (MFDP), the first step is the selection of relevant codes. Our approach to select codes is based on a performance assessment of different error detection codes relying on both related work and our

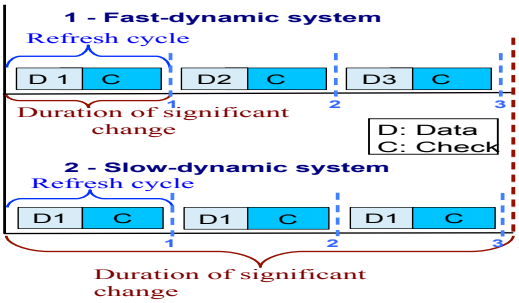


Fig. 3. Comparison between fast-dynamic and slow-dynamic properties

experiments. The codes selection strategy consists in a trade-off of the following three criteria: the error detection intrinsic capability, the computational cost and the complementary property with other codes (see section IV).

A. Relevant error detection codes to be used in MFDP

According to related work on error detection codes and particularly works targeting systems sharing similar properties as those of safety-critical embedded systems (adopting relatively lightweight codes), CRC codes, Adler and Fletcher checksums are relevant to enhance communication integrity in such context. Thus, we decided to consider these codes. We target exclusively 8 check bits because for short messages typically exchanged in CES, it is crucial to avoid overhead given limited processing and memory resources.

The first criterion to be met by selected codes is a low computational cost. Since the computational cost is independent of the error type we rely on results of related work. According to [12], it is confirmed that, for the same number of check bits, CRC has the highest computational cost compared to Adler and Fletcher checksums. In fact, using CRC results in a factor of two to four higher computational cost [12] compared to Adler and Fletcher which have similar computational cost. Although there are different techniques to speed up CRCs (e.g., lookup tables [13] and Fast-CRCs [14]), we can not conclude that these techniques make CRCs as lightweight as Adler and Fletcher checksums.

According to these results and in order to define a lightweight MFDP, we should rather use jointly Fletcher and Adler checksums or at least alternate a CRC and an Adler/Fletcher checksum. We introduce now our experiments considering the other two criteria.

B. Experiments: methodology and simulation scenarios

Our experiments are designed not only to explore the feasibility of MFDP but also to assess its performance. It is noteworthy that if all data and error patterns are considered, experiments could be prohibitively slow. For this reason, we rely mainly on fault injection (via Monte Carlo simulations) rather than carrying out exhaustive simulations requiring unrealistic simulation durations.

In our simulations, all generated messages are erroneous. In each simulation, we generate 10^7 erroneous messages. Messages are generated first such that the payload data is generated randomly then the whole message (payload and check bits)

is subjected to errors. For error generation, we consider a random strategy too and we simulate the repetitive feature of the error. In fact, the major goals of our experiments are: i) evaluating the intrinsic effectiveness of different considered codes; ii)assessing the complementarity property of considered codes and iii) comparing the error detection effectiveness of proposed multi-functions policies to classical single-function policies.

In this context, we consider different payload data sizes: 16, 32, 64 and 96 bits. We have investigated the following error detection codes:

- As checksums: Fletcher-8, Adler-8, Adler-16 and Fletcher-16 (see description in section III-A).
- As CRCS: CRC-8 whose generator polynomial is $G(x) = x^8 + x^4 + x^3 + x^2 + 1$, the Fast-CRC-8 whose generator polynomial is $G(x) = x^8 + x^2 + x + 1$, a CRC-16 whose generator polynomial is $G(x) = x^{16} + x^{15} + x^2 + 1$, the CRC-15-CAN which is deployed in CAN and whose generator polynomial is $G(x) = x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$ and the Fast-CRC-16 (whose generator polynomial is $G(x) = x^{16} + x^2 + x + 1$).

It is noteworthy that we consider 16-bits codes in order to compare single-functions policies using 16 check bits in all messages with multi-function policies using only 8 check bits and alternating two different 8-bits codes.

We remind that the terminology Code-n means that the code uses n check bits. We evaluate non-detection rate (NDR) metric which is the ratio of the number of non detected erroneous messages to the number of injected erroneous messages.

C. Validation: Results and conclusive remarks

We present here our results and conclusive remarks considering the two remaining criteria which are: i) intrinsic error detection capability and ii) complementarity property.

a) Intrinsic error detection capability of codes: From our experiments, we notice that different error detection codes (single function error detection policies) bring similar error detection performances if using the same number of check bits. Fig.4 describes the error detection effectiveness of the considered 8-check bits codes (Adler-8, Fletcher-8, CRC-8 and Fast-CRC-8). The NDRs of different codes are almost similar (3.88E-03 $\leq$ NDR $\leq$ 3.93E-03) with slight differences according to data payload size. This result is consistent with the theoretical value (which is called "upper bound" in [15]) of NDR which is equal to 2^{-n} for an n-check bit code. For a 8-check bits code, this value is equal to 3.91E-03. These first series of experiments validate our prediction that for multiple repetitive random errors, codes having the same check bits size exhibit similar effectivenesses. This is different from results of related work targeting burst errors with a fixed size of erroneous bits, experiments carried out in [16] show that, for burst errors and for short messages ($\leq$100 bits), Fletcher and Adler exhibit similar error detection performances for 8-check bits codes. Yet, Fletcher lightly outperforms Adler for larger messages ($\geq$ 100 bits). In the same context and according to [12], CRC brings better error detection performance than Fletcher and Adler checksums in the case of short messages particularly if its generator polynomial is well selected [17].

Since we target multiple repetitive random errors and according to our results, in terms of error detection capability, CRC, Adler and Fletcher exhibit similar good performances. Yet, critical embedded systems require better error detection performances than those provided by single function policies. So, The question now is whether the codes are complementary and whether the MFDP brings better performance.

b) Complementarity property of different codes: the last selection criterion is the complementarity property. Only the work presented in [8] investigated this property considering only CRC codes. This work proposed to use jointly a set of CRCs based on different generator polynomials sharing a fewest factors in common in terms of primitive polynomials (it is noteworthy that each generator polynomial can be decomposed into a set of primitive polynomials). So, if an error is not detected by the first CRC, there is a chance that it could be detected by another used CRC. But this raises another limitation: it is not easy to find a set of polynomials that are both more different from each other and each individually with a high intrinsic error detection capability. In fact to be sure that a CRC detects some error patterns, the generator polynomial must meet some criteria (e.g., it must include the term 1 in order to detect odd errors). All used generator polynomials in [8] are multiple of (x+1) which decreases the complementarity property. Another limit of this work consists in the fact that the CRC computational cost was about 10% of the refresh cycle which is not adequate when considering the strict requirements of CES.

Theoretically, two codes having different vulnerabilities could be complementary. It was noticed in [18] that: i) Fletcher and Adler checksums are vulnerable to burst errors that invert bits from all zeros to all ones; ii) Fletcher is vulnerable to some 2 bits errors where the two bits have different values not both 0 or both 1; iii) CRC fails to detect errors that are multiple of its generator polynomial. We notice that CRC, Adler and Fletcher are vulnerable to different error patterns. So, they could be complementary. This conclusion is to be confirmed by simulations.

We propose to evaluate the effectiveness of some multi-error detection functions which combine two 8-check bits based codes. We combine either two checksums (e.g., Adler and Fletcher) or a checksum and a CRC (e.g., Adler and CRC). Fig.5 illustrates the effectiveness of some multi-error detection functions which combine two 8 check-bits based codes. Either two checksums are combined (e.g., Adler and Fletcher) or a checksum and a CRC (e.g., Adler and CRC). These choices reflect the need to provide lightweight policies that take profit of the low computational cost of checksums and the complementarity property of combined codes. The results of the application of a single code are also included in Fig. 5 for the sake of comparison, and to better highlight error detection performance improvement. Combining two error detection codes reduces the non detection rate (NDR) and then increases the error detection performance. As an example, for a payload data size equal to 96 bits, CRC-8 brings an error non detection rate equal to 3.92E-05 to be compared to 1.51E-05 when Adler-8 is combined to CRC-8 (the theoretical upper bound if using 16 check bits is equal to 1.53E-05).

These results illustrate the effectiveness of our proposed multi-function error detection approach and the complementarity of the considered codes.

The MFDP opens the way to another relevant scenario illustrated in Fig.6: it makes it possible to reduce the number of check bits per message while providing the same detection performance. The scenario presented in Fig.6 compares the conventional CRC code deployed in CAN (considered as applied to each message), with several MFDP combining two 8 check-bits based codes. Results show that results are similar or even better. For example, the combination of Fletcher-8 and Adler-8 leads a NDR equal to 1.81E-05, compared to 3.10E-05 for CRC-15-CAN. We chose CRC-15-CAN in this experiment because it is widely used.

The scenario described in Fig. 7 aims at evaluating the error detection performance of proposed multi-function error detection policies. More precisely, the scenario compares the use in each single message, of either two 8 check bits based codes, or a single 16 check bits based codes. We notice that proposed MFDPs bring similar performances. Yet, we remind that our policies reduce the computational costs since proposed MFDP combine either two arithmetic checksums or a CRC and an arithmetic checksum which require a lower computational cost in policies using a CRC code in all transmissions.

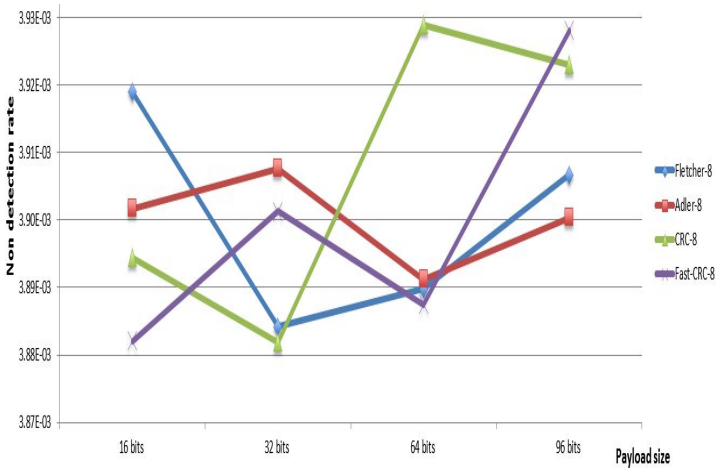
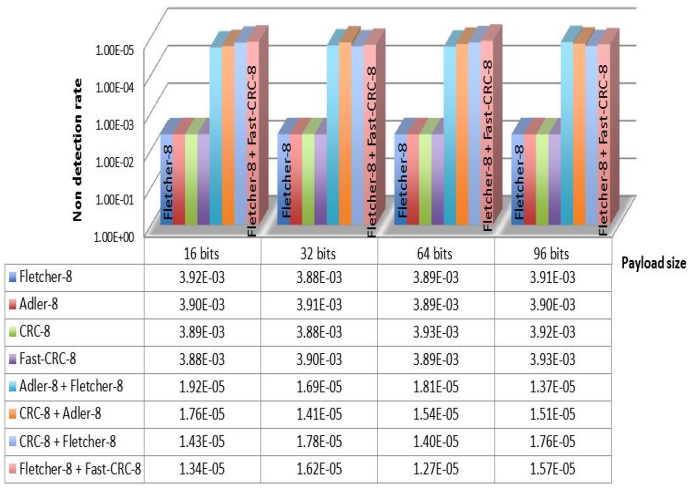


Fig. 4. The non detection rates of different 8 bits error detection codes

VII. Data Diversity based Policy (DDP): basic concepts

In the same context of diversifying the message copies to be sent taking profit of redundancy, our second diversity-based proposal is an alternative approach based on the concept of "data diversity" (Fig. 8) described in some papers devoted to systems safety and security [5] [6]. This concept is usually used to enhance security, software integrity, data storages or communication integrity. For software integrity, it consists in diversifying the data inputs in order to avoid selecting inputs causing failures. For communication integrity, its use is less common, it consists in using re-expression functions in order to transform data to be transmitted. On the receiver side, original data provided by different functions are restored and compared in order to detect potential errors. Paper [6] adopts this concept to define an approach for enhancing security.



	16 bits	32 bits	64 bits	96 bits
Fletcher-8	3.92E-03	3.88E-03	3.89E-03	3.91E-03
Adler-8	3.90E-03	3.91E-03	3.89E-03	3.90E-03
CRC-8	3.89E-03	3.88E-03	3.93E-03	3.92E-03
Fast-CRC-8	3.88E-03	3.90E-03	3.89E-03	3.93E-03
Adler-8 + Fletcher-8	1.92E-05	1.69E-05	1.81E-05	1.37E-05
CRC-8 + Adler-8	1.76E-05	1.41E-05	1.54E-05	1.51E-05
CRC-8 + Fletcher-8	1.43E-05	1.78E-05	1.40E-05	1.76E-05
Fletcher-8 + Fast-CRC-8	1.34E-05	1.62E-05	1.27E-05	1.57E-05

Fig. 5. The non detection rates of different error detection policies based on single or multiple 8 bits error detection bits

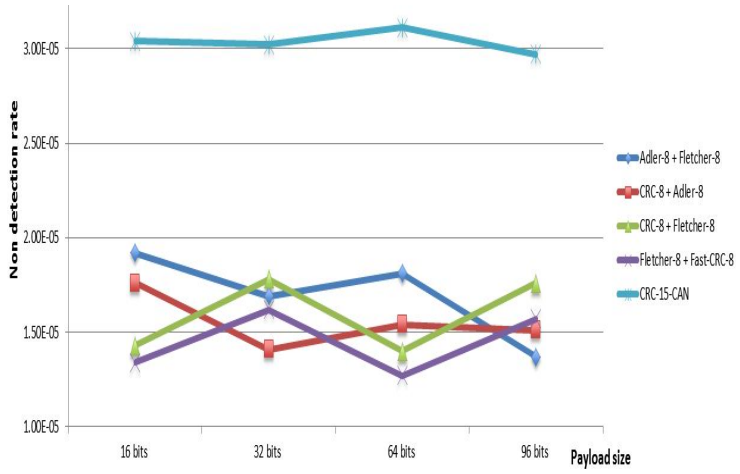


Fig. 6. Comparison of non detection rates between MFDP combining two 8 bits codes and CRC-15-CAN

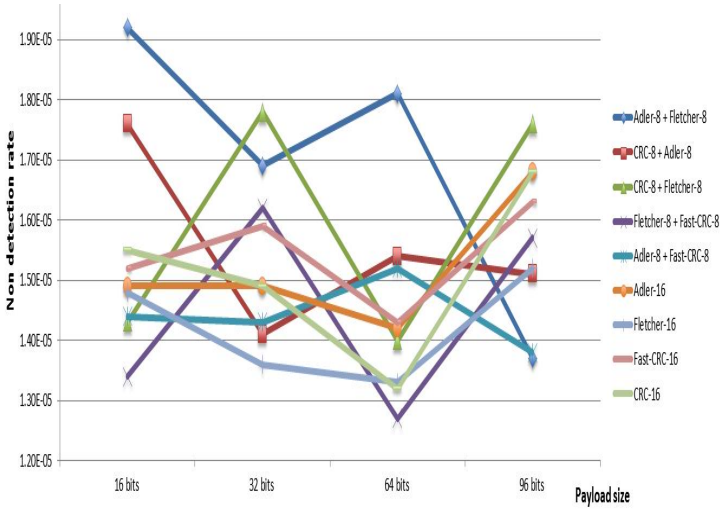


Fig. 7. Comparison of non detection rates of MFDP combining two 8-bits codes and 16 bits error detection codes

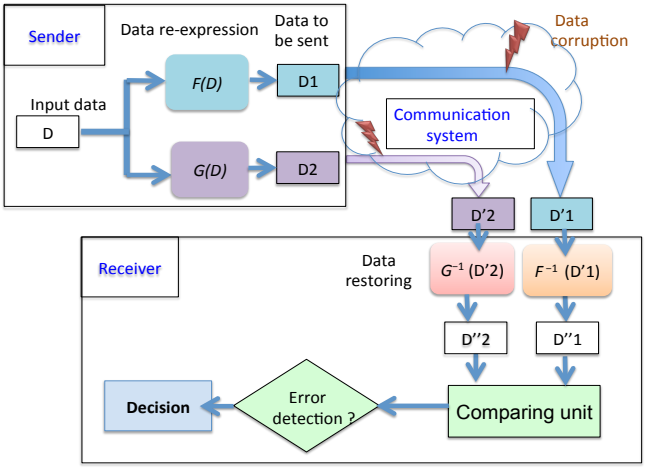


Fig. 8. Overview of Data diversity approach for communication integrity

Our work is aimed at adapting this approach in order to make it relevant to communication integrity in targeted critical embedded systems. Fig. 8 gives a simplified overview of the principle of diversity data applied to the case of CES communication (or network), based on two different data re-expression functions ($F(D)$ and $G(D)$). Similar to the re-expression functions of the approach targeting system security [6], the adopted re-expression functions in our fault-tolerant integrity approach must satisfy the two following criteria:

The inverse property: this means that the re-expressing function has an inverse function ($F^{-1}(D)$ and $G^{-1}(D)$ in Fig. 8) that makes it possible to restore the original data. This property is important for two reasons: i) it avoids the likelihood to loose the semantic of the data currently sent, ii) restored data is used to compare different received data in order to detect errors.

The disjointedness property: this means that if we apply two different functions to the same data vector, the two results do not match (they are not equal). This allows to detect errors since the system is notified systematically by the presence of an error if there are two received similar data vectors provided by two different data re-expression functions.

In this paper, we just introduce basic concepts of this approach, Yet, future simulations are needed to validate it.

VIII. Conclusion

This paper explored the use of a diversity-based approach to enhance data communication integrity for critical embedded systems. Two different policies are discussed: i) diversifying the error detection functions by combining multiple error detection codes, and ii) adopting data diversity. Our approach targets critical embedded systems which deploy in particular temporal and/or spatial redundancy. We have mainly focussed on the second approach and carried out simulation experiments to validate its performance. The simulation allowed us to highlight the benefits of our proposed approach compared to conventional strategies based on a single error code. Future work will be devoted mainly to further explore the data diversity based policy by defining appropriate data re-expression functions and illustrating their effectiveness based on simulations.

References

[1] A. Avizienis, J. C. Laprie, B. Randell and C. Landwehr, *Basic Concepts and Taxonomy of Dependable and Secure Computing*. IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, January-March 2004.

[2] K. Dorow, *Flexible Fault Tolerance in Configurable Middleware for Embedded Systems*. International Conference on Computer Software and Applications (COMPSAC 2003), Dallas, TX, USA, 3-6 Nov. 2003, pp.563-569.

[3] L. Marques, V. Vasconcelos, P. Pedreiras and L. Almeida, *Tolerating Transient Communication Faults with Online Traffic Scheduling*. International Conference on Industrial Technology (ICIT 2012), Athens, Greece, 19-21 March 2012, pp.396-402.

[4] A. Avizienis and J. P. J. Kelly, *Fault tolerance by design diversity: concepts and experiments*. IEEE Computer, vol.17, August 1984, pp. 67-80.

[5] P. E. Ammann and J. C. Knight, *Data diversity: an approach to software fault tolerance*. IEEE Transactions on Computers, vol. 37, no.4, Apr. 1988, pp.418-425.

[6] A. Nguyen-Tuong, D. Evans, J. C. Knight, B. Cox and J. W. Davidson,*Security through Redundant Data Diversity*. International Conference on Dependable Systems andNetworks (DSN 2008), Anchorage, Alaska, 24-27 June 2008, pp.187-196.

[7] A. Zammali, A. de Bonneval and Y. Crouzet, *A multi-function error detection policy to enhance communication integrity in critical embedded systems*. International Conference on Software Security and Reliability (SERE), San Francisco, CA, USA, 30 Jun- 2 Jul 2014.

[8] A. Youssef, Y. Crouzet, A. de Bonneval, J. Arlat, J. J. Aubert and P. Brot, *Communication Integrity in Networks for Critical Control Systems*. The European Dependable Computing Conference (EDCC), Coimbra, Portugal, 18-20 Oct. 2006, pp.23-34.

[9] M. Paulitsch and B. Hall, *FlexRay in Aerospace and Safety-Sensitive Systems*. Aerospace and Electronic Systems Magazine, vol.23, Sept. 2008, no.9, pp.4-13.

[10] M. Rahmani, B. Muller-Rathgeber, E. Steinbach, *Error Detection Capabilities of Automotive Network Technologies and Ethernet - A Comparative Study*. Intelligent Vehicles Symposium (IV'07), Istanbul, Turkey, 13-15 June 2007, pp.674-679.

[11] Federal Aviation Administration, *System Safety Handbook* - chapter 3: Principles of System Safety, 19 pages, Dec. 2000. Available at: http://www.faa.gov/regulations_policies/handbooks_manuals /aviation/risk_management/ss_handbook/media/Chap3_1200.pd

[12] T. Maxino and P. Koopman, *The effectiveness of checksums for embedded control networks*. IEEE Transactions on dependable and secure computing, vol. 6, no. 1, January-March 2009.

[13] I. Indu and T. S. Manu, *Cyclic Redundancy Check Generation Using Multiple Lookup Table Algorithms*. International Journal of Modern Engineering Research (IJMER), vol.2, Issue.4, July-Aug. 2012, pp-2445-2451.

[14] G. D. Nguyen, *Fast CRCs*. IEEE Transactions on computers, vol. 58, no. 10, Oct. 2009.

[15] M. Paulitsch, J. Morris, B. Hall, K. Driscoll, E. Latronico and Ph. Koopman, *Coverage and the use of Cyclic Redundancy Codes in Ultra-Dependable Systems*. The international Conference on Dependable Systems and Networks (DSN), Yokohama, Japan, 28 Jun-1 Jul. 2005, pp.346-355.

[16] T. Maxino, *Revisiting Fletcher and Adler Checksums*. The international Conference on Dependable Systems and Networks (DSN) Student Forum, Sheraton Society Hill, Philadelphia, PA, USA, 25-28 Jun. 2006.

[17] P. Koopman and T. Chakravarty, *Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks*. The international Conference on Dependable Systems and Networks (DSN), Florence, Italy, 28 Jun-1 Jul. 2004, pp.145-154.

[18] A. Nakassis, *Fletchers Error Detection Algorithm: How to Implement It Efficiently and How to Avoid the Most Common Pitfalls*. Computer Comm. Rev., vol. 18, no. 5, Oct. 1988, pp. 63-88.