A Multi-function Error Detection Policy to Enhance Communication Integrity in Critical Embedded Systems

Amira Zammali, Agnan de Bonneval, Yves Crouzet

A multi-function error detection policy to enhance 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, ongoing work that investigates a new error detection policy aiming at enhancing communication integrity in the presence of permanent errors (single and multiple). We consider critical embedded systems which are based on complex networks including active interstage nodes. This property increases the occurrence probability of permanent errors. The novelty of the proposed policy lies in the fact that unlike classical policies using a single error detection function, it is based rather on a set of different error detection functions. The different used functions must be complementary in terms of detection capability in order to increase the resultant error detection capability. Our reference application to illustrate the proposed concepts is the Flight Control System (FCS). However, our objective is also to apply the proposed approach to other application domains sharing similar features and characteristics.**

***Keywords*-communication integrity; multiple permanent errors; critical embedded systems; error detection codes; safety; flight control system;**

I. Introduction and context

In critical embedded systems, the occurrence of errors in exchanged messages could lead to a system failure resulting in damages and even in fatalities. So, ensuring the communication integrity is crucial.

Nowadays, these systems are based on complex networks that are vulnerable to binary errors affecting the transmitted data. Different causes could lead to such errors, e.g., permanent faults in interstage nodes, noise, etc. In this work, we address the particular case of permanent errors (single and multiple) affecting communicated messages.

To deal with errors in communications, the usual integrity approach consists in using a single error detection function based on an error detection code (e.g., Cyclic Redundancy Check "CRC") and if it is necessary to increase the detection capability, the usual way is to increase the number of check bits (e.g., increasing the generator polynomial degree of the used CRC). This approach is efficient for many applications but in others, it is not sufficient, particularly if permanent errors are targeted. Let us give an example of one of these applications. It is an industrial case study that was studied in the context of a project involving our research team and an aerospace company: the Flight Control System (FCS). 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. 1).

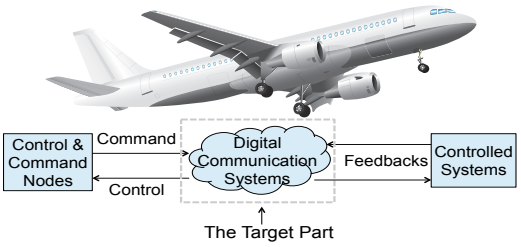

Fig. 1. The targeted part in the Flight Control System

In this previous case study [1], CRC was used to detect errors and it was noticed that if a permanent error is not detected once by the used CRC code, it will never be detected since the same CRC was used in all refresh cycles. So, it was proposed to use a different CRC in each transmission. The used CRCs are based on different generator polynomials. As known, CRC fails to detect an error when the erroneous data is a multiple of its generator polynomial. Accordingly, the approach proposed in [1] selected generator polynomials sharing a minimum of common factors, so if the erroneous data is not detected by the first code, there is a chance that the second CRC detects the error. It was proven that the use of different CRCs increased the error detection capability. In this context, the communication integrity is ensured at the application layer. The approach proposed in [1] to deal with the problem of permanent errors was exclusively based on CRCs. Yet, the computation of CRC codes implies a significant performance overhead, about 10% of the refresh cycle duration.

Based on the preliminary work presented in [1], we propose in this paper a significant extension that consists in using a multi-function error detection policy that instead of using a single error detection function, uses a different error detection function in each refresh cycle. The used functions are complementary, this latter property means that the used codes cover different sets of errors. Our approach is not based exclusively on CRC codes, it uses rather different code families (e.g., Fletcher checksum and Adler checksum).

Our ultimate goal is to meet the integrity requirement while reducing the computational cost and the redundancy in terms of check bits. In this paper, we present the basic concepts of the proposed multi-function error detection policy. Section II presents related work and points out our main contribution compared to existing approaches sharing with us the same core objective to meet the required integrity level in critical embedded systems. Section III describes the systems we target and the assumptions. Section IV presents our proposed policy and section V presents our simulations and results. Finally, section VI concludes this paper.

II. Related work

Given the importance of ensuring communication integrity in critical embedded systems, many works have been devoted to this issue. The proposed approaches are different due to two reasons: i) the fact that each case study has its own specificity (e.g., some systems do not permit redundancy) and ii) the progress of technologies that enables the use of some techniques used to be too complicated to be deployed (e.g., recent advances in computer-based systems offer the possibility to use error detection techniques with heavy computations).

The first lesson that we have learnt from related work is the significant impact of problem assumptions on the integrity policy. In fact, paper [2] details assumptions taken explicitly or implicitly by most of existing works and shows that they do not reflect sometimes the real system requirements. Unrealistic assumptions are likely to result in an inefficient policy. So, we must select reasonable and valid assumptions in order to define an efficient error detection policy.

We noticed that many approaches are based on CRC codes. The designers usually think that choosing the CRC code, which is defined by its generator polynomial, is a matter of conventional wisdom [3]. So they use conventional CRC codes used in common protocols and standards (e.g., CRC-32 polynomial used in the IEEE 802.3). However, as pointed out in [4], relying on the conventional wisdom to select the generator polynomial of CRC is not the right strategy of selection. They demonstrate, via exhaustive experimentations, that there are non-standard generator polynomials that provide much better performances in terms of detection capability. It was also shown that the choice of generator polynomials is closely dependent on the data size to be protected (experimentations are done on data packets from 8 to 2048 bits).

Some works proposed approaches using non-CRC codes and providing good performances. In [5], authors rely on Adler and Fletcher codes in their error detection approach. Their results show that good error detection capabilities can be obtained with these codes which are more lightweight than CRC. Moreover, the work presented in [6] provides also interesting insights about the distribution of errors and their impact on the error detection capabilities. It is assumed that different areas in the message packet do not exhibit the same bit error rate (BER). So, a new integrity approach is proposed to deal with this problem. This approach relies on flexible unequal error control codes that meet the different integrity requirement of each packet area.

Other approaches not based on codes have been also investigated. The work presented in [7] proposes a double-bus architecture that guarantees the availability and the integrity of exchanged messages by duplicating the communication links between every pair of nodes and comparing the transmitted data. An other way to enhance the error detection is the concept of "data diversity". This concept is usually used to enhance integrity in software or in data storages. It consists in diversifying the data inputs in order to avoid failures-causing inputs. Paper [8] adopts this concept to ensure integrity in data storage.

From all described possibilities, we decide to further explore the benefits of using proposed lightweight codes in the context of a multi-function error detection policy. We could consider later the flexible unequal error control codes since we target permanent errors and the occurrence of such errors means that there are some areas of the data packet that are more vulnerable than others. So, these vulnerable areas need much stricter protection. In our future works, we could also consider the "data diversity" and the double-bus architecture to enrich our integrity approach but right now we focus on error detection codes and we assume no communication channels redundancy in our work.

III. Targeted systems and generic assumptions

We present here our reference application which is the Flight Control System and we detail the key properties and generic assumptions of other targeted systems.

A. Reference application: the Flight Control System

A Flight Control System (FCS) is a critical system. In fact, a failure in such system could lead to fatalities. So, FCS is characterized by a set of strict safety requirements namely the integrity requirements. FCS 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 node. In civil aviation, events are classified according to their severity which is defined by the level of damages caused by the event occurrence. The most common classification [9] 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.

FCS belong to a class of systems that we call slow-dynamic. In fact, we classify systems into two classes [1]: i) fast-

dynamic; ii) slow-dynamic as shown in Fig. 2. 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 duration is the duration between two successive transmissions). So, the transmitted data (e.g., control-command messages) remains the same (the same value) during a set of refresh cycles before a significant change occurs. In the example of Fig. 2, the same message (Data 1 in Case 2 of Fig. 2) is sent several times. Yet, in fast-dynamic systems, the duration of this interval is very close to the duration of the refresh cycle so the message can be sent only once. Considering the example of FCS, airplane surfaces are designed to move slowly and the result of the calculation of the trajectory remains the same in a set of calculation refresh cycles so the same message is sent during this time. In slow-dynamic systems, the fact that the message is sent several times induces that some undetected erroneous messages can be tolerated before error detection and recovery.

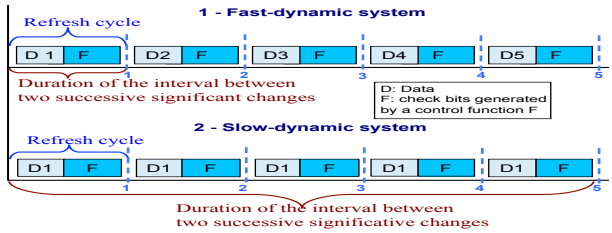

Fig. 2. Comparison between fast-dynamic and slow-dynamic properties

B. Other targeted systems: key properties and generic assumptions

Aiming at applying our proposed approach in other application areas (e.g., automotive systems), we define the key properties and the generic assumptions required to apply our proposed approach.

1- *High criticality:* this property induces strict safety requirements. In fact, in such systems, failures may lead to catastrophic events. Typically, the failure rate must be less than a threshold to be set according to the system specificity (10^{-9} to 10^{-7} failures per an hour of system service).

2- *Embedded and limited resources:* we particularly focus on the fact that many embedded systems do not include huge amounts of resources (e.g., memories and processors). Moreover, communications are based on short messages (e.g., 100 bits for the FCS). So, we cannot rely, in such systems, on an error detection policy with a high redundancy in terms of check bits since this increases performance overheads and computational costs.

3- *Complexity:* 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 nodes which are active (with memory and processing capabilities). The "active" feature increases the occurrence probability of permanent errors.

4- *Repetitive transmission:* this property consists of sending each message N times ($N > 1$). So, up to some undetected erroneous messages among the N transmitted messages can be tolerated and the goal is to detect the permanent error in at least one message among the N messages (N to be set according to the system specificity). Despite the fact that this latter property seems to be restrictive, we can notice that many systems transmit messages repetitively for different reasons: i) by design like in the case of slow-dynamic systems; ii) to make communications reliable and to deal with the problem of packet loss or to enhance the message integrity by applying the packet comparison strategy that consists in considering correct the copy of message having the maximum occurrences among received copies.

IV. An Overview of the Multi-Function Error Detection Policy

We now describe with details our policy. We first present our specific assumptions, then we detail our proposed policy.

A. Specific assumptions

Since the error detection policy depends closely on the problem assumptions [2], we detail here our specific assumptions which are: 1) our policy is implemented in the application layer so it is independent of the network architecture, 2) a calculation refresh cycle of some ms (10 ms for FCS), 3) no redundancy of the communication channels and 4) communication channels are i) memoryless (output at the time t depends only on the input at the time t and not on previous ones); ii) binary (two symbols 0 and 1) and iii) symmetric (the error probability is independent of the transmitted symbol).

B. Proposed approach

Our approach is based on error detection codes. Yet, the idea is to take profit of the fact that, in the targeted systems, every message is transmitted N times ($N > 1$). This latter property allows us to adopt a fault-tolerant approach. In fact, our targeted systems do not consider the occurrence of one erroneous undetected message as a dangerous event to be avoided but rather the worst case is when the message is erroneous and the error is permanent and not detected in all N transmissions of the message. Thus, our goal is to detect the error at least once among the N transmissions. If the same error detection function is used in all transmissions and it fails to detect the permanent error once, it will never be able to detect it. So, the idea is to use diverse error detection functions to increase the likelihood to detect at least one erroneous message among the N transmitted messages. Logically and by design, error detection codes are not vulnerable to the same error patterns (an example of error pattern is the set of errors multiple of the generator polynomial of a CRC). In other words, error detection codes have different weaknesses. 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 and raises a set of challenges that have to be taken up.

1- *A good intrinsic detection performance:* to select relevant error detection codes, we rely mainly on their intrinsic error detection capabilities that are described in many papers. According to existing works evaluating the performance of these codes, the most adequate codes to be used to detect errors on exchanged data in embedded systems are CRC, Fletcher and Adler. In the paper [5], it is confirmed that theoretically the Adler checksum has roughly the same detection performance properties as those of the Fletcher checksum (e.g., using 16 check bits, they detect both all burst errors less than 8 bits long) but experimentations show that Fletcher checksum is lightly superior to Adler checksum in almost all cases. Yet, if Fletcher is compared to CRC [10], it is clear that for short messages (typically exchanged in embedded networks), using a CRC can bring better error detection performance for a factor of about four performance penalty (e.g., for burst errors, CRC with 16 check bits detect up to 16 bits burst errors). Thus, we can conclude that in terms of detection performance, CRC, Adler and Fletcher are adequate to enhance the communication integrity in our targeted systems. CRC has the best intrinsic capability outperforming Fletcher which outperforms Adler. However, this is not the single criterion to be taken into account in order to select codes to be used.

2- *The trade-off of computational cost and error detection performance:*it is true that CRC, Adler and Fletcher are the three potential codes to be used by our approach according to their error detection performances in embedded systems. But, as described in the section III, because of limited resource, our policy is not based exclusively on the intrinsic error detection performance criterion but we consider also the computational cost to select codes to be used. In related work on error detection codes, authors confirm that CRC has the highest computational cost compared to other error detection codes. CRC outperforms Fletcher and Adler in terms of error detection effectiveness in return for a factor of two to four of computational cost [10].

3- *Complementarity property:* the real challenge of our policy is not how to find appropriate codes but rather it consists in the way how to define the multi-function error detection policy that requires complementary codes. Two error detection functions are complementary means that 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 respectively by $F1$ and $F2$.

To guarantee the complementarity property, previously[1], used CRCs were defined by different generator polynomials having the fewest factors in common in terms of primitive polynomials (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 a chance that it could be detected by the second CRC. We are inspired of this way to find complementarity between the used error detection codes. Indeed, it was noticed that Fletcher and Adler checksums are vulnerable to burst errors that invert bits from all zeros to all ones [11], Fletcher is vulnerable to some 2 bits errors where the two bits have different values not both 0 or both 1 and 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.

Thus, we propose a multi-function policy that uses CRC jointly with Adler or/and Fletcher. The use of non-CRC codes in some refresh cycles makes the approach more lightweight compared to using CRC codes in all refresh cycles. Moreover, this approach takes profit of the high intrinsic detection effectiveness of CRC codes and deal with their weaknesses by using other lightweight codes.

C. Example of a multi-function error detection policy

Fig. 3 gives an example of a multi-function error detection policy. We assume, in this example, that: 1) the message is transmitted $N = 5$ times, 2) all transmitted messages are erroneous and the error is multiple and permanent, 3) $F1$, $F2$ and $F3$ are three complementary error detection functions, 4) $F1$ is vulnerable to the considered error pattern so it does not detect it, $F2$ and $F3$ detect the error. In the first case, since $F1$ is not able to detect the permanent error and is used in all transmissions, the error is never detected. In the second case,

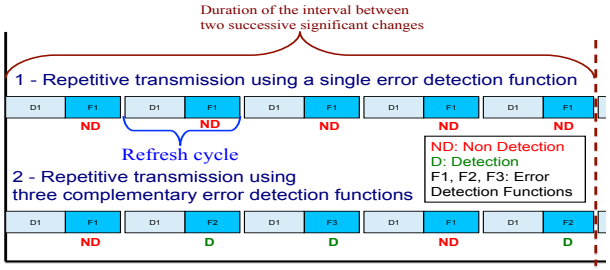

Fig. 3. An example of a multi-function error detection policy

we change cyclically three error detection functions ($F1$, $F2$, $F3$, $F1$...). Since $F2$ and $F3$ are able to detect the error, the multi-function error detection policy deals with the problem of permanent error.

V. Simulations and Results

We describe, in this section, our current simulations and results to prove the effectiveness of our proposed approach. We first describe our simulation methodology. Then we present some results for illustration.

A. Simulation methodology

To evaluate the performance of our proposed approach, our experiments were based on simulated fault injection via Monte Carlo simulations. The main idea of Monte Carlo simulation is to use random samples (values) of parameters or inputs to

explore the behaviour of a complex system and to perform sensitivity analyses. This particular method is used when the exhaustive approach is prohibitively slow in terms of simulation duration which is the case of our simulations if all data patterns and errors patterns are considered. Every simulation consists in a set of iterations, every iteration includes these following four steps.

1- *Data generation:* we generate the data packet randomly.

2- *Error generation:* the packet formed above is subjected then to an error. The error is generated either randomly or with a selective manner (e.g., injecting all 2-bits errors or all single errors).

3- *Evaluation of the detection capability of each error detection function:* for each used error detection function, we calculate the check bits of the erroneous data and we compare it to the existing control part (which could be erroneous too). If they match, the error is not detected. We sum the number of undetected erroneous frames and the non detection rate.

4- *Evaluation of the detection capability of the multi-function error detection policy combining all single functions:* we consider the error detection result of the K used single control functions (detection or non detection). If there is, at least, one function that detects the error so the multi-function policy detects the error. Otherwise (all functions do not detect the error), the multi-function policy fails to detect the error.

B. Simulation platform and considered scenarios

We are using the Matlab-Simulink platform which provides tools to model and simulate communications in critical embedded systems. We define a Simulink model that is based on blocks allowing to generate data and errors and evaluate the error detection performance of used codes (Fig. 4). For CRC codes, Simulink offers a specific blocks allowing to evaluate the performance of CRC. But, to consider Adler or Fletcher codes, it was necessary to develop ourselves specific new blocks. To generate data, we use a random binary data generator based on a Bernoulli process which is a discrete-time stochastic process that takes only two values: 0 and 1. We generate a 116-bits packet, 100 bits of data and 16 check bits. We consider one communication channel and we generate 10^8 data packets. To evaluate the error detection capability of considered error detection functions, we consider the non-detection error rate metric. It is the ratio of the number of undetected erroneous packets to the number of all transmitted packets. In order to conform to the assumption that we target permanent errors, all considered control functions have simultaneously the same erroneous data as input.

In order to simplify the simulation model, we just consider refresh cycles using different error detection functions (we assume here that $K = N$, the realistic case was described in section IV). This assumption does not impact the results since a function used more than once does not cumulate the error detection performance. In our model, if the error is not detected, the output of the considered single control function is equal to 1. For multi-functions policy, we sum the outputs of the used K functions. If the result is equal to K then all the functions fail to detect the error. Thus, the multi-function policy fails to detect the permanent error. We increment then the number of undetected error and we update the rate of non detection.

In our experiment, we simulate the following scenarios of error detection policy: i) using a single function based on CRC code, ii) using a single function based on Fletcher, iii) using a single function based on Adler, iv) a multi-function error detection policy combining CRC and Adler, v) a multi-function policy combining CRC and Fletcher, vi) a multi-function policy combining CRC, Fletcher and Adler. In Fig. 4, we highlight the different simulation steps: 1) corresponds to the data and error generation and 2) to 5) correspond to a subset of the considered simulation scenarios.

It is noteworthy that, in our simulation, the used CRC is defined by its generator polynomial $G(x)$ such that $G(x) = (x+1)G'(x)$, $G'(x)$ is a primitive polynomial so that it is divisible only by itself and by $P(x) = 1$. Thus, it can not be decomposed into other polynomials. We generate selective errors that change the data to make it multiple of $G'(x)$ in order to simulate a case where CRC is vulnerable. We test then the ability of other code families to deal with CRC weaknesses if they are used jointly with CRC.

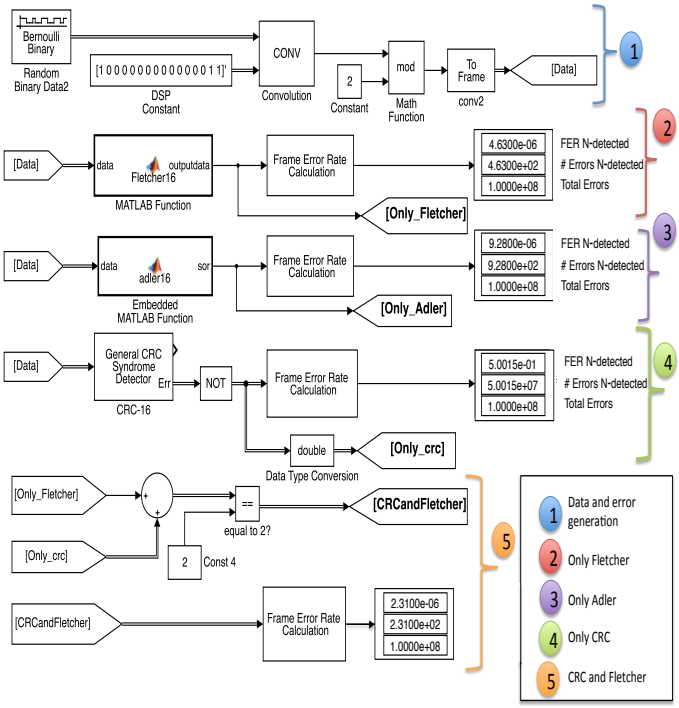

Fig. 4. A part of the Simulink model of our experimentation

C. Results and discussion

We remind that these experiments targeted a particular error pattern and do not reflect the real intrinsic error detection capability of used codes when considering all possible error patterns. Rather, we aim to show that these codes could be complementary and if the first code is vulnerable to the

considered error pattern, the other codes can detect a part of theses errors. The non detection rate of CRC is equal to $5 * 10^{-1}$, this result is predictable given the considered error pattern. The non detection rate of Adler is equal to $9.2 * 10^{-6}$ and the non detection rate of Fletcher is equal to $4.6 * 10^{-6}$. We conclude that Fletcher and Adler are obviously less vulnerable to this particular error pattern. So, they can deal with the weaknesses of CRC if they are used jointly with CRC in a multi-function error detection policy. Yet, this does mean that Fletcher and Adler outperforms CRC if other error patterns are considered. This conclusion is proved by the experimentation result which shows that the multi-function error detection policy combining CRC and Adler improves the detection capability with a non detection rate equal to $4.8 * 10^{-6}$. The multi-function error detection policy combining CRC and Fletcher improves also the detection capability with a non detection rate equal to $2.3 * 10^{-6}$. Finally, the multi-function error detection policy combining CRC, Adler and Fletcher detects all injected erroneous packets and gives a non detection rate equal to 0.

This is a promising result proving that if we use jointly CRC with Adler or/and Fletcher, we can deal with a significant part of the errors not detected by CRC. This result does not prove that these three codes are totally complementary. In fact, we can not conclude that for all possible errors, the multi-function policy combining CRC, Adler and Fletcher detects all errors. It rather increases the detection capability compared to the single error detection function using CRC.

To sum up, our proposed approach improves the error detection rate (Fig. 5). It uses CRC that guarantees a good error detection performance since this code is intrinsically efficient (described in section IV), it uses also Adler or/and Fletcher to complement the effectiveness of CRC by covering a significant part of the errors not detected by CRC. The use of Fletcher and Adler makes also the approach more lightweight compared to the case when CRC is used in all refresh cycles. Results seem promising. However, to confirm the effectiveness of our approach, more future theoretical proofs and simulations are needed. In fact, more scenarios must be considered to prove that this policy is practical for targeted critical embedded systems and to show that it is efficient for different error patterns (e.g., single, burst, etc).

VI. Conclusion

The usual use of error detection codes in digital networks is a proven approach to detect errors. However, using a single error detection fails to detect some errors. So, we propose to deploy an error detection policy using a different error detection function in each refresh cycle. The used error detection functions are complementary so that the resultant detection capability could be increased. Our approach targets mainly Flight Control Systems but could be deployed in other different applications. It uses different codes and aims at increasing the error detection capability for permanent errors with low redundancy and computational cost. In this paper, we have described the problem assumptions and the proposed policy with the set of challenges arising from it. Then we have presented our ongoing simulations and preliminary results. Future work will focus first on validating this approach by more simulations and, if possible, formal proofs. Our goal will be also to define a strategy for scheduling the different used error detection functions.

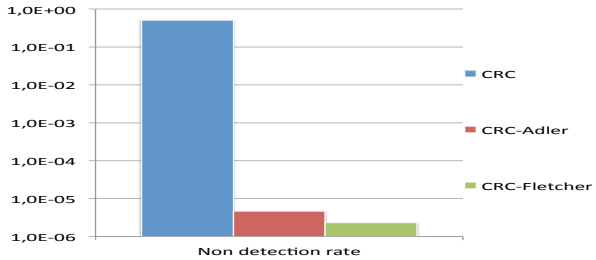

Fig. 5. The error non detection rates of different error detection policies

References

[1] 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, pp.23-34, 18-20 Oct. 2006.

[2] 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, pp.346-355, 28 Jun-1 Jul. 2005.

[3] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C (2nd ed.), Cambridge Press, 2002.*

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

[5] 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.

[6] L. J. Saiz-Adalid, P. J. Gil-Vicente, J. C. Ruiz-Garcia, D. Gil-Tomas, J. C. Baraza and J. Gracia-Moran, *Flexible Unequal Error Control Codes with Selectable Error Detection and Correction Levels*. The International Conference on Computer Safety, Reliability and Security (Safecomp), Toulouse, France, 24-27 Sep. 2013.

[7] M. Paulitsch and B. Hall, *Insights into the Sensitivity of the BRAIN (Braided Ring Integrity Network) -On Platform Robustness in Extended Operation*. The international Conference on Dependable Systems and Networks (DSN), 2007.

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

[9] 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.pdf

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

[11] 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, pp. 63-88, Oct. 1988.