A Tunable Add-On Diagnostic Protocol for Time-Triggered Systems [*]

Marco Serafini and Neeraj Suri
TU Darmstadt, Germany
{marco, suri}@informatik.tu-darmstadt.de

Jonny Vinter
SP, Sweden
jonny.vinter@sp.se

Astrit Ademaj
TU Vienna, Austria
ademaj@vmars.tuwien.ac.at

Wolfgang Brandstätter
Audi, Germany
wolfgang.brandstaetter@audi.de

Fulvio Tagliabò
Fiat, Italy
fulvio.tagliabo@crf.it

Jens Koch
Airbus Deutschland, Germany
jens.koch@airbus.com

Abstract

We present a tunable diagnostic protocol for generic time-triggered (TT) systems to detect crash and send/receive omission faults. Compared to existing diagnostic and membership protocols for TT systems, it does not rely on the single-fault assumption and tolerates malicious faults. It runs at the application level and can be added on top of any TT system (possibly as a middleware component) without requiring modifications at the system level. The information on detected faults is accumulated using a penalty/reward algorithm to handle transient faults. After a fault is detected, the likelihood of node isolation can be adapted to different system configurations, including those where functions with different criticality levels are integrated. Using actual automotive and aerospace parameters, we experimentally demonstrate the transient fault handling capabilities of the protocol.

1. Introduction and contributions

In both automotive and aerospace X-by-wire applications, TT platforms such as Flexray [1], TTP/C [2], SAFEbus [3] and TT-Ethernet are increasingly being adopted. Most TT platforms develop their static, built-in diagnostic and membership approach. Instead, we define an on-line diagnostic/membership protocol that is a tunable and portable add-on application level module. It can be integrated as a plug-in middleware module (as an application) onto any TT system, without interference with other functionalities. It only uses information that is available at the application level, does not impose constraints on the scheduling of the system, and has low bandwidth requirements. For TT platforms, such as FlexRay, SAFEbus and TT-Ethernet, that do not have a standardized diagnostic or membership protocol, our add-on protocol represents a viable solution for such functionalities.

[*]Research supported in part by EC DECOS, ReSIST and DFG TUD-GK MM

Our diagnostic protocol exploits specific features of TT systems where multiple nodes access a shared broadcast bus using TDMA communication. The ability of a node to *send* correct messages in the designated time window (called *sending slot*) is used as a periodic diagnostic test. The protocol is able to detect bursts of multiple concurrent faults and to tolerate malicious faults. Its resiliency also scales with the number of available nodes.

The key purpose of a diagnostic protocol is to trigger correct and timely recovery/maintenance actions, particularly for safety critical subsystems. However, a diagnostic protocol needs also consider availability and avoid unnecessary substitutions of correct components in case of external transient faults, which are becoming more frequent [4]. An "ideal" diagnostic protocol would exclude only nodes with internal faults. In practice, however, internal faults do not always manifest as permanent faults at the interface of the node (e.g. crashes). They can also manifest as multiple, subsequent intermittent faults which, to external observers, appear similar to external transient faults. We consider an *extended fault model* to characterize *healthy* and *unhealthy* nodes based on the presence of internal faults. In order to recognize unhealthy nodes, a *penalty/reward (p/r) algorithm* delays the isolation of faulty nodes to accumulate on-line diagnostic information. This is a novel extension of the basis developed in [5, 6] and represents an application of our alternative p/r model [7].

A problem similar to diagnosis is membership, which consists of identifying the set of nodes (called membership view) that have received the same set of messages. We will show that a variant of our protocol can act as a membership service and detect the formation of multiple *cliques* of receivers with inconsistent information.

We have implemented the protocol in a prototype, reproducing practical automotive and aerospace settings. Using physical fault injection, we experimentally validate the properties of the protocol and show how to tune the parameters of the p/r algorithm in a realistic environment.

The paper is organized as follows. Following the related work in Sec. 2, we introduce the system and fault models in Sec. 3 and 4. The tunable add-on diagnostic protocol and its properties are presented in Sec. 5 and 6. The protocol is extended to a membership protocol in Sec. 7. Sec. 8 describes the experimental validation of both protocols. We detail parameter tuning in Sec. 9. Sec. 10 discusses the portability of the middleware to different TT platforms.

2. Related work

The general diagnosis problem was formulated in the PMC model [8], where a set of active entities test each other until sufficient information exists to locate the faulty nodes. In on-line, real time settings the comparison approach is recommended [9], where the same functionality is executed on different nodes and the results are compared.

Multiple research efforts have targeted diagnosis for specific error models, and for improving specific attributes such as latency reduction, coverage and bandwidth. The family of diagnostic protocols for generic synchronous systems proposed by Walter et al. [11] considers a frame-based communication scheme where nodes exchange messages in synchronous parallel rounds using a fully connected topology and unidirectional links. Similar to consensus [18, 10], all nodes exchange their local view on the correctness of the messages received by the other nodes and combine them using hybrid voting to achieve consistent diagnosis.

We adapt the on-line diagnosis approach of [11] as a middleware service for TT systems, where multiple nodes access a shared broadcast bus using a TDMA communication scheme. Our add-on protocol explicitly takes into account the internal scheduling of each node and the overall global communication scheduling of the system. We extend the protocol to consider the cases of communication blackout, which can arise if particularly long transient bursts corrupt all sending slots in the TDMA round. We also show how to modify the protocol to provide membership information. Finally, we define a new p/r algorithm to handle transient faults based on the criticality of the applications executing on different nodes.

A count-and-threshold fault detection function (called α-count) is introduced in [5, 6] to discriminate between transient and intermittent faults. The fundamental tradeoffs in its tuning are explored using stochastic evaluation. Our alternate p/r model, which develops an overall FDIR (Fault Detection, Isolation and Reconfiguration) strategy, is introduced and analyzed in [7]. In this work we present how to experimentally tune the p/r algorithm in realistic settings.

The problem of group membership is often defined similar to diagnosis [12]. Cristian [13] proposed a membership protocol for synchronous crash-only systems that is based on an expensive fault-tolerant atomic broadcast primitive to achieve consistency. Such an approach is impractical in TT systems due to its high latency and bandwidth requirement.

A membership protocol specifically designed for TTP/C systems was proposed by Kopetz et al. [2, 14]. It relies on the "single fault assumption", i.e., it does not tolerate simultaneous faults, and assumes non-malicious node failures. The protocol allows identification of one fault in the communication of a message. Besides faulty senders, the protocol also detects if asymmetric receiver faults cause the formation of different cliques of nodes. The latency is two communication slots in the case of sender faults and two TDMA rounds in case of receiver faults. The bandwidth required is $O(N)$ bits per message and $O(N^2)$ bits per round, where N is the number of system nodes. If a (possibly transient) faulty node is detected it is generally restarted, generating a window of vulnerability to subsequent failures. An extension of this protocol was proposed by Ezichelvan and Lemos [15] to tolerate up to half of senders being simultaneously faulty with a latency of three TDMA rounds. Our protocol tolerates multiple coincident non-malicious and malicious faults with the same bandwidth requirement. Due to its add-on and generic nature, it has a higher latency. However, in Sec. 10 we show that a system-level variant of our protocol features a latency of two TDMA rounds.

3. System model

We assume a synchronous system model and a network topology where all nodes access a shared (and possibly replicated) communication bus using a TDMA access scheme, i.e., a periodic schedule where each node is assigned a time window, called *sending slot*, in each *TDMA round* (or round). The periodic *global communication schedule*, including when each slot begins and terminates, is defined at design time and executed by a *communication controller*. The communication controller features a local collision detection mechanism, which checks if messages sent by the node can actually be read from the bus.

The systems consists of N nodes with unique IDs $\{1, ..., N\}$ assigned following the order of the sending slots in the round. Correct nodes can identify a sender by its sending time and there is no message forging. Faulty nodes cannot corrupt messages sent by correct nodes.

Communication among jobs, including those running on different nodes, is abstracted by a vector of shared variables $\langle v_1, \ldots, v_N\rangle$ called *interface variables*. Communication controllers automatically update their value by sending and receiving messages according to the global communication schedule. Copies of the interface variables are updated at the receivers after every sending slot is completed. Updates follow the sending order of the corresponding messages. Interface variables can be updated at most once per round.

Each interface variable has a corresponding *validity bit*. This is set to 0 by the communication controller when the value of the variable can no longer be considered correct. If an interface variable v_i has node i as its unique sender and is updated at each round, we can assume that the communication controller uses its *local error detection mechanisms* to set the validity bit of v_i at the receiver node j to 0 iff node j was not able to receive the last message sent by i that was supposed to update v_i, and 1 otherwise. Validity bits are updated together with the corresponding messages.

Besides the global communication schedule, each node has its own internal *node schedule* that determines when jobs are executed. In a TDMA access scheme, the sending slot of a node overlaps with the computational phase of other nodes. The node schedule can thus have an effect on the "freshness" of the *read* interface state, i.e., the round where the values of the interface variables were sent. For example, if a job is executed at the beginning of a round it will only read values sent in the previous rounds. The node schedule also determines the round when the data written in the interface state is actually *sent* on the bus. A job might be able to send its output data in the same round as it is executed only if it is scheduled before the sending slot of the hosting node. To increase the portability of our add-on protocol, we do not constrain the scheduling of nodes.

4. Fault model

We use a Customizable Fault-Effect Model [16] which refers to the *communication errors* in the broadcast of a message. A received faulty message is *locally detectable* if it is syntactically incorrect in the value domain or early/late/missing in the time domain; it is *malicious* faulty if it is *not* locally detectable but is semantically incorrect in the value domain.

Correspondingly, we partition faults into three classes:

- *symmetric benign*: (or benign) message is locally detectable by *all* the receivers;
- *symmetric malicious*: all the receivers receive the same malicious message;
- *asymmetric*: message is locally detectable by at least one but not all the receivers.

We assume broadcast channels, where different locally undetectable messages cannot be asymmetrically received by different nodes. Asymmetries in the local detection of messages can be an effect, for example, of Slightly-Off-Specification faults (SOS) [17], when the clock of a node is close to the allowed offset and thus the messages it sends are seen as timely only by a subset of the receivers. Another example is when EMI disturbs only part of the bus.

We classify nodes based on the communication errors they display in their outgoing messages, e.g., benign faulty sender, malicious faulty sender etc. We assume that each node can display only one type of communication error throughout one execution of the protocol. *Correct* nodes send messages without faults. *Obedient* nodes follow the program instructions and execute only correct internal state transitions. They can either be correct or suffer omission failures while sending or receiving messages.

For diagnosis, we do not assume permanent faults but consider an *extended fault model* instead, where all nodes alternate periods of faulty behavior, when they are not able to correctly send messages, and periods of correct behavior. We consider a node:

- *healthy*, if it suffers only sporadic and external transient faults;
- *unhealthy*, if it suffers internal faults which manifest as intermittent or permanent communication faults.

We implicitly assume that internal faults will manifest at the interface of the node either (a) as permanent sender faults (a long faulty burst) or (b) as intermittent faults with a shorter time to reappearance than external transient faults. A crashed node, for example, is an unhealthy node that permanently displays benign faults.

5. The on-line diagnostic protocol

The purpose of the on-line diagnostic protocol is to detect and isolate unhealthy nodes from the system at runtime. It is composed of two algorithms. The first algorithm forms a *consistent health vector* to consistently locate benign faulty senders, the second accumulates the diagnostic information using the p/r algorithm to distinguish (in a probabilistic manner) between healthy and unhealthy nodes.

Each node i runs, at each round, the diagnostic job $diag_i$, which sends a non-replicated diagnostic message dm_i and receives all the other interface variables $\langle dm_1, \ldots, dm_N \rangle$. The communication controller provides a validity bit for each interface variable dm_j sent from $diag_j$ to $diag_i$ using its local error detection mechanisms. By checking the validity bits of the diagnostic messages, the protocol diagnoses communication errors. The *local syndrome* of node i is the binary N-tuple containing its local view on the messages sent by other nodes (faulty/not faulty). The diagnostic message dm_i contains the local syndrome broadcast by node i and its size is $O(N)$.

The diagnostic protocol consists of five phases:

1) *Local detection:* Communication errors are locally detected by observing the local validity bits of the diagnostic messages. A new *local syndrome* is formed as a binary N-tuple.
2) *Dissemination*: The local syndrome is *broadcast* using the diagnostic message dm_i.
3) *Aggregation*: Receive all local syndromes dm_j corresponding to the same previous *diagnosed round*. Form a *diagnostic matrix* for that round where *row* i is the local syndrome sent from node i and *column* j is a vector representing the opinion on node j of all other nodes.
4) *Analysis*: A binary N-tuple called *consistent health vector*, which contains the consistent distributed view on the health of all system nodes in the diagnosed round, is calculated. To combine the local syndromes sent by different nodes, a hybrid voting [11, 18] over the *columns* of the diagnostic matrix is performed. If enough nodes observed a benign fault, the sending node is considered faulty.
5) *Update counters*: Based on the consistent health vector, update the *penalty and reward counters* associated to a node, and possibly isolate faulty nodes.

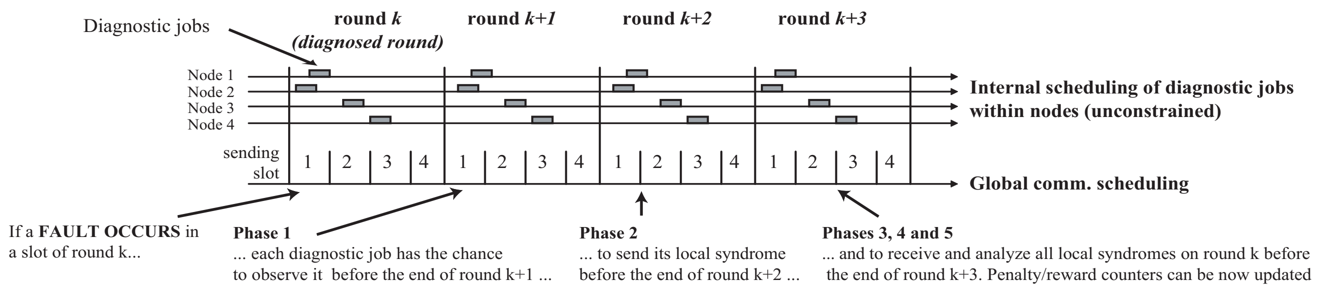
Diagnostic jobs
round k
(diagnosed round)
round k+1
round k+2
round k+3
Node 1
Node 2
Node 3
Node 4
sending slot
1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
Internal scheduling of diagnostic jobs within nodes (unconstrained)
Global comm. scheduling
If a FAULT OCCURS in a slot of round k...
Phase 1
... each diagnostic job has the chance to observe it before the end of round k+1 ...
Phase 2
... to send its local syndrome before the end of round k+2 ...
Phases 3, 4 and 5
... and to receive and analyze all local syndromes on round k before the end of round k+3. Penalty/reward counters can be now updated

Figure 1: High level overview of the diagnostic protocol in a system with four nodes

The phases of the protocol are executed in consecutive TDMA rounds, and phases of multiple instances of the protocol are interleaved at each execution of $diag_i$ (Fig. 1). The pseudo code of the diagnostic job $diag_i$, running on each node i, is presented in Alg. 1.

Consistent location of faulty senders. Local detection and aggregation entail reading the interface variables and their validity bits. We do not constrain the scheduling of the diagnostic jobs in a round. Thus, we need to consider that, in a TDMA communication scheme, diagnostic jobs running on different nodes can see different views of the interface variables, as the freshness of the read data can vary.

Consider a diagnostic job $diag_i$ which, at round k, reads from the interface variables the values of the diagnostic messages $dm_1, \ldots, dm_N$ and their validity bits. As diagnostic messages are sent at every round, the read values were sent (following the sending order) either on round k or $k-1$. Hence there is a locally known integer $l_i \in [0, N-1]$, determined by the internal schedule of $diag_i$ within node i, such that values of $dm_1, \ldots, dm_{l_i}$ were sent in round k, while values $dm_{l_i+1}, \ldots, dm_N$ were sent in round $k-1$ (the same holds for their validity bits)[1]. For all diagnostic jobs executed in round k to consistently use *aligned* diagnostic messages (resp. validity bits) from round $k-1$, the protocol executes a *read alignment* operation (Fig. 2; Alg. 1, Lines 3-6). Read alignment combines in variables al_dm_j (al_ls_j) values $prev\_dm_{[1,\,i]}$ ($prev\_ls_{[1,\,i]}$) from the previous round and of $curr\_dm_{[i+1,\,N]}$ ($curr\_ls_{[i+1,\,N]}$) from the current. This requires buffering of messages and validity bits (lines 16-17), and introduces additional delays in the communication.

For *local detection*, the validity bits are read (Alg. 1, line 2) and combined using read alignment (lines 3-6). The vector al_ls contains in round k the local syndromes corresponding to the messages sent in round $k-1$.

During the *dissemination* phase a *send alignment* is also needed to ensure that, despite unconstrained node scheduling, all local syndromes sent in round k refer to a same previous diagnosed round, as required by the aggregation phase executed in the following round. We define the predicate $send\_curr\_round_j$ to be true if, according to the internal schedule of node i, the diagnostic messages formed by the diagnostic job $diag_j$ at round k can be sent in round k (i.e., $diag_j$ is completed before the sending slot of the node). If the predicate holds for all nodes, all current local syndromes can be immediately written in the interface state (line 7) and the latency of the protocol is reduced. However this global condition may not hold, or it may be impossible to locally evaluate it (e.g. if the node scheduling is dynamic, see Sec. 10). In these cases, send alignment is used to determine the data to be written in the interface variables, which will be later sent. If a job completes its execution before the sending slot of its node, it writes the local syndromes obtained in the previous round; otherwise the current local syndromes are written (lines 8-10).

The *aggregation* phase first reads the values of the local syndromes sent by all diagnostic jobs through the diagnostic messages (line 1). A special error value ε is assigned to local syndromes whose validity bit is 0. Read alignment is used to guarantee that all jobs executed in round k form a diagnostic matrix using local syndromes sent in round $k-1$, which refer to the same diagnosed round (lines 3-6); vector al_dm_j represents the j^{th} row of the matrix. The j^{th} element of the local syndrome sent by node i to node k can assume three possible values: 0, if i was not able to receive the message from node j in the slot of interest; 1, if i was able to receive the message from j; ε, if k was not able to receive the local syndrome from i correctly. For example, Table 1 shows the diagnostic matrix formed in case node 3

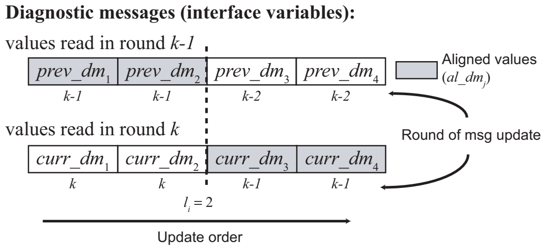

Figure 2: Example of read alignment (round k, $l_i = 2$)

[1]If a diagnostic job $diag_i$ is executed after the last sending slot of a round and can read data from round k from each node, we treat it as it was executed in round $k+1$ and set $l_i = 0$ accordingly.

Table 1: Example diagnostic matrix (3-4 benign faulty)

		Accused node			
Accuser node	**Local syndr.**	*1*	*2*	***3***	***4***
Node 1	al_dm_1	-	1	0	0
Node 2	al_dm_2	1	-	0	0
Node 3	al_dm_3	ε	ε	-	ε
Node 4	al_dm_4	ε	ε	ε	-
	Voted $cons\_hv$	1	1	**0**	**0**

and 4 are two (coincident) benign faulty senders in both the diagnosed round and the dissemination round.

As faults can occur during the dissemination phase of the protocol, the diagnostic matrices can contain incorrect or incomplete information, and different nodes can form different diagnostic matrices due to asymmetric faults. However, a consistent global view on faults in the diagnosed round can be obtained by combining different local views using a hybrid voting function H-maj(V) (Eqn. 1) over the columns V of the matrix. The opinion of a node about itself is considered unreliable and discarded to tolerate asymmetric faults (see Sec. 6). Thus, voting is executed over the $(N-1)$-tuple V of local syndromes representing the opinions of the other nodes (lines 11-13). In order to tolerate benign faults, a hybrid voting function excludes erroneous votes ε from V ($excl(V,\varepsilon)$) before calculating the majority [18] (see example in Table 1).

As for validity bits and local syndromes, the value 0 denotes a faulty node. In case no correct local syndrome is available ($|excl(V,\varepsilon)| = 0$), the voting function can not reach a decision. This can happen only if at least $N-1$ nodes are not able to send their local syndrome and a node is not able to determine whether it was faulty in the diagnosed round. In this case, the protocol cannot do anything else than relying on the outcome of the collision detector in the diagnosed round (line 14). The consistent health vector $cons\_hv$ is the outcome of the hybrid majority voting and contains, at round k, the agreed view on the health of each node at the diagnosed round, i.e., $k-3$ or $k-2$ (see Sec. 6).

$$\text{H-maj}(V) = \begin{cases} \bot & \text{if } |excl(V,\varepsilon)| = 0 \\ v & \text{if } v = maj(excl(V,\varepsilon)) \\ & \quad \text{and } |excl(V,\varepsilon)| \geq 1 \\ 1 & \text{else} \end{cases} \qquad (1)$$

Filtering unhealthy nodes. The consistent health vector is given as an input (Alg. 1, line 15) to the penalty/reward algorithm (Alg. 2), which is used to handle transient faults and discriminate them from intermittent and permanent faults. Each node keeps a penalty and a reward counter for each node in the system in the vectors $penalties$ and $rewards$. They are both initially set to 0. Whenever the node is detected as faulty, the corresponding penalty is increased by each node depending on the criticality of the jobs allocated on the node. Criticality levels for each node are stored in the vector $criticalities$. We discuss the selection

Algorithm 1: Node i diagnostic job $diag_i$

```
begin
    // Phases 1 and 3 - Local detection and Aggregation
    // (read alignment)
1   $\langle curr\_dm_1, \ldots, curr\_dm_N \rangle \leftarrow$
    read_iface($dm_1, \ldots, dm_N$);
2   $curr\_ls \leftarrow$ read_vbits($dm_1, \ldots, dm_N$);
3   for $j \leftarrow 1, \ldots, l_i$ do
4       $al\_dm_j \leftarrow prev\_dm_j$;   $al\_ls[j] \leftarrow prev\_ls[j]$;
5   for $j \leftarrow l_i + 1, \ldots, N$ do
6       $al\_dm_j \leftarrow curr\_dm_j$;   $al\_ls[j] \leftarrow curr\_ls[j]$;
    // Phase 2 - Dissemination
    // (send alignment)
7   if $\forall j$ : $send\_curr\_round_j$ then write_iface($al\_ls$);
8   else if $send\_curr\_round_i$ then
9           write_iface($prev\_al\_ls$);
10      else write_iface($al\_ls$);
    // Phase 4 - Analysis
    // (consistent location of benign faulty senders)
11  for $j \leftarrow 1, \ldots, N$ do
12      $diag \leftarrow$ H-maj$\langle al\_dm_1[j], \ldots, al\_dm_{j-1}[j]$,
        $al\_dm_{j+1}[j], \ldots, al\_dm_N[j] \rangle$;
13      if $diag \neq \bot$ then $cons\_hv[j] \leftarrow diag$;
14      else $cons\_hv[j] \leftarrow$ coll-det(diagnosed_round);
    // Phase 5 - Update counters
    // (decision on node isolation)
15  $active \leftarrow active$ AND pen_rew($cons\_hv$);
    // (buffering for read and send alignment)
16  $\langle prev\_dm_1, \ldots, prev\_dm_N \rangle \leftarrow$
    $\langle curr\_dm_1, \ldots, curr\_dm_N \rangle$;
17  $prev\_ls \leftarrow curr\_ls$;   $prev\_al\_ls \leftarrow al\_ls$;
end
```

of such criticality levels in Sec. 9. If no fault is successively detected the reward is increased by one. The algorithm uses two constants, a *penalty threshold* P and a *reward threshold* R. After a bounded amount of time either of the two thresholds is exceeded, resulting in isolation of the node or reset of the counters respectively.

The two counters, and the corresponding thresholds, represent two different kinds of information: the reward counter (threshold) indicates the (minimum) number of consecutive fault-free slots a node needs to display before the memory of its previous faults is reset; the penalty counter (threshold) indicates the (maximum) number of consecutive faulty slots a node is allowed to display before isolation.

As the health assessment of the system stored in vector $cons\_hv$ is consistently calculated in Alg. 1, the penalty and reward counters are always consistently updated, and isolations are decided in the same round by all obedient nodes.

The vector $active$ contains the status of activity of each node and represents the internal output of the diagnostic protocol (Alg. 1, line 15). Eventual traffic generated by isolated nodes must be ignored by the communication controllers of all other nodes. Upon reintegration of a node, the value of the corresponding element is set back to the initial

Algorithm 2: The p/r algorithm

```
begin
    for $i \leftarrow 1, \ldots, N$ do
        $curr\_act[i] \leftarrow 1$;
        if $cons\_hv[i] = 0$ then
            $penalties[i] \leftarrow penalties[i] + criticalities[i]$;
            $rewards[i] \leftarrow 0$;
            if $penalties[i] \geq P$ then $curr\_act[i] \leftarrow 0$;
        else
            if $penalties[i] > 0$ then
                $rewards[i] \leftarrow rewards[i] + 1$;
                if $rewards[i] \geq R$ then
                    $penalties[i] \leftarrow 0$;    $rewards[i] \leftarrow 0$;
    return $curr\_act$;
end
```

value 1 (up) and the traffic considered again. As the problem of reintegration is outside the scope of this paper, the algorithm only sets activity bits to 0 (isolated).

6. Properties of the diagnostic protocol

In this section we prove the properties of the diagnostic information stored in the consistent health vector $cons\_hv$. We show the experimental evaluation used to practically tune the parameters of the p/r algorithm in Sec. 9.

The properties of the consistent health vector are:

- *Correctness*: a correct sender is never diagnosed as faulty by obedient nodes;
- *Completeness*: a benign faulty sender is always diagnosed as faulty by obedient nodes;
- *Consistency*: diagnosis is agreed by all obedient nodes.

Our protocol does not discriminate between node and link faults. Transient external faults in the communication network are filtered using the p/r algorithm. We remark that in our extended fault model these properties hold for obedient nodes, i.e., both correct nodes *and* nodes encountering omission faults, whereas for classical diagnostic protocols they only hold for correct nodes. These properties imply that an obedient node is able to diagnose itself.

We first prove that the diagnostic matrix used for the hybrid voting consists of validity bits of messages sent in the same round. Next we study the conditions under which the hybrid voting is able to calculate a consistent health vector that provides for the three properties defined above.

Lemma 1 *All local syndromes al_dm_j correctly received and aggregated by the diagnostic jobs executed at round k contain the value of the validity bits of the messages sent in the same previous diagnosed round, which can be either $k-3$ or $k-2$ depending on the schedule of nodes and on the global communication schedule.*

Proof Diagnostic messages are updated at every round. Therefore, each diagnostic job at round k can read values sent either in round k or $k-1$. The read alignment done in the aggregation phase ensures that all local syndromes al_dm_j were sent in round $k-1$. Such local syndromes were formed either in the same round as they were sent or in the previous, i.e., either in round $k-1$ or $k-2$. This depends on the local schedule of diagnostic jobs with respect to the global communication schedule, i.e., on the send alignment. The local detection phase also uses read alignment to consistently form local syndromes of validity bits referring to messages sent in the previous round, i.e., either round $k-3$ or $k-2$. This round is therefore the diagnosed round. □

The aligned local syndromes formed at round k constitute the diagnostic matrix for the diagnosed round $k-3$ or $k-2$. Due to malicious faults during the dissemination, local syndromes can contain incorrect information, and different diagnostic matrices can be formed at different nodes. In the following, we prove that the hybrid voting function H-maj(V) calculates a consistent health vector satisfying correctness, completeness and consistency; a, s and b will represent the number of asymmetric, symmetric malicious and benign faulty nodes over one execution of the protocol.

Lemma 2 *The consistent health vector calculated by each obedient node at round k guarantees* consistency*,* completeness *and* correctness *for faults occurred at round $k-3$ or $k-2$ as long as $N > 2a + 2s + b + 1$ and $a \leq 1$.*

Proof As proved in Lemma 1, the hybrid voting H-maj(V) is executed over the local syndromes of validity bits referring to the same diagnosed round. Let us consider the N-1 tuple of values $V = \langle al\_dm_1[i], \ldots, al\_dm_{i-1}[i], al\_dm_{i+1}[i], \ldots, al\_dm_N[i] \rangle$ used by an obedient node to diagnose i. Among these values, up to b are erroneous (ε), the actual number depending on the amount of benign faulty nodes failing during the dissemination phase. The other $N-b-1$ values are either correct (and therefore symmetric) or malicious/asymmetric.

If the diagnosed sender was correct or benign faulty, all correct votes will carry the same opinion to all obedient nodes. As $N-b-1 > 2(a+s)$, malicious and/or asymmetric votes are a minority and are outvoted by correct values in each obedient node. Thus, a consistent decision is reached that ensures correctness and completeness.

If the diagnosed sender is not correct nor benign faulty in the diagnosed round, the only property required is consistency. If the sender had been symmetric malicious, it was not detected as faulty by any node. Similar to the previous case, there is a consistent majority of correct votes at each obedient node saying that the sender was not faulty, and therefore consistency is guaranteed.

If the diagnosed sender was asymmetric faulty, its vote does not contribute to the diagnosis on its health, since the opinion of a node on itself is ignored and excluded from the vector V. As there can be at most one asymmetric sender over one execution of the protocol, each obedient node receives the same set of votes and reaches a consistent diagnosis, which can assume any value. □

The condition $N > 2a + 2s + b + 1$ requires that, even if there are no malicious faults, $b < N - 1$. In case of long transient bursts in the communication bus, multiple (and possibly all) subsequent slots in the round will be compromised, resulting in a higher number of sender faults. The protocol behaves correctly also under these conditions, as proved by the following Lemma:

Lemma 3 *If there are only benign faulty senders, the consistent health vector calculated by each obedient node at round k guarantees* correct*,* complete *and* consistent *diagnosis of the other nodes for faults occurred at round $k - 3$ or $k - 2$ if $N - 1 \leq b \leq N$. For correct, complete and consistent self-diagnosis, the correctness of the local collision detector is necessary.*

Proof If there are only benign faulty nodes, all local syndromes will be consistent and will reflect the state of the system. An obedient node can thus correctly diagnose other nodes even if it does not receive any external local syndrome. However, when the node has to diagnose itself and no external local syndromes are available, it cannot distinguish whether it was able to correctly send its message or not, unless it queries the local collision detector. Any default decision in this case could be incorrect and inconsistent with the (correct) diagnosis of the other nodes. The correctness of the local collision detector is therefore not only sufficient but also necessary for self-diagnosis. □

Lemmas 1, 2 and 3 imply Theorem 1 as:

Theorem 1 *The consistent health vector calculated by each job at round k guarantees* correctness*,* completeness *and* consistency *for faults occurred at round $k - 3$ or $k - 2$ if: $N > 2a+2s+b+1$ and $a \leq 1$; or there are only benign faults, $N - 1 \leq b \leq N$. In the latter case, local collision detection is necessary for self-diagnosis.* □

7. The membership protocol

A common approach to keep consistency in fault-tolerant distributed systems is to use a *group membership* service. When an asymmetric fault occurs, nodes are partitioned into two sets, also called *cliques*, such that the members of one clique received the message whereas the other did not. In such case a membership service outputs a new *view* consisting of the larger of these cliques. As all the members of a clique have received the same set of messages, they have a consistent state. The properties required for a group membership service are the following:

- *Membership liveness*: A new unique view is formed whenever an obedient node receives a locally detectable faulty message m;
- *View synchrony*: As a new view is formed, all obedient nodes remaining across consecutive views have received the same set of messages prior to, and including, m.

If there is a benign fault, all receivers form a unique clique and Alg. 1 detects sender faulty nodes correctly. In case of asymmetric faults, however, two different cliques of receivers are formed and the diagnostic protocol of Alg. 1 cannot detect them.

A *modified* diagnostic protocol can detect the presence of disjoint cliques and allow the determination of views according to the properties above. In Alg. 1, the analysis phase must be executed before the dissemination phase; after the consistent health vector is calculated, the modified algorithm accuses (as member of the minority clique) the nodes that send local syndromes disagreeing with it. Such accusations, called *minority accusations*, are added in the current aligned local syndrome al_ls and subsequently disseminated. The protocol satisfies the desired properties as shown in Theorem 2:

Theorem 2 *If an obedient correct node receives a locally incorrect message m and $N > 2a+2s+b+1$, $a \leq 1$, a new view is generated after two complete executions of the modified diagnostic protocol* (membership liveness) *containing all nodes never deemed as faulty. Such a view satisfies* view synchrony *for all messages prior to and including m.*

Proof A locally detectable message can be received due to either a benign or an asymmetric fault. If a benign fault occurs, it is detected by the diagnostic protocol as shown in Theorem 1. The sender is the only node which received the message and will be excluded from the view.

If an asymmetric fault occurs during the broadcast of message m, two cliques of obedient nodes are formed. Theorem 1 guarantees that all obedient nodes calculate a consistent health vector, which contains a consistent decision (faulty/non faulty) on message m. During the dissemination phase of the diagnostic protocol, however, obedient nodes of the minority cliques try to send local syndromes disagreeing with the consistent decision. As $a \leq 1$, such nodes can either correctly broadcast it, and be accused by all other obedient nodes (minority accusation), or be benign faulty senders, and thus be accused by the local detection mechanisms of all the other obedient nodes. In both cases they will be consistently accused and diagnosed as faulty in the next execution of the diagnostic protocol. □

8. Validation of the protocols

In this section we present the results of the experimental validation of the diagnostic and membership protocols. We used physical fault injection to validate the properties of the protocol under different scenarios. *We emphasize that all parameters used in the validation (and tuning, see Sec. 9) arise from actual automotive and aerospace applications.*

Prototype setup. The validation setup consists of a set of four nodes consisting of a host computer (Infineon Tricore 1796) and a communication controller (Xilinx Vertex

4 FPGA), which are interconnected via a redundant TT network (layered TTP). Each host computer runs a TT operating system. A diagnostic job runs on each node as an add-on application-level module sending one diagnostic message per round. No constraint was imposed on the internal node scheduling besides executing diagnostic jobs once every round. The static node scheduling defined the constant integers $l_{\{1,..,N\}}$ and the predicates $send\_curr\_round_{\{1,..,N\}}$ used by the protocol for the read and the send alignment operations. Interface variables are automatically updated and the validity bits of a message m can be read using the API call `tt_Receiver_Status`. The bandwidth required for each diagnostic message is $N = 4$ bits.

We also used an additional disturbance node, which is able to emulate hardware faults in the communication network. As the protocol does not discriminate between node and link faults, a fault in a node can be emulated by corrupting or dropping a message it sends.

Injection cases. We selectively injected different classes of physical faults on the bus (electrical spikes, random noise, periods of silence) to simulate faults in a deterministic and reproducible manner. As we know which faults are injected, we can experimentally evaluate whether the diagnostic protocol is able to detect them. Each *experiment class* was repeated 100 times for consistency. A total of 1500 fault injection experiments was conducted.

We injected bursty faults of increasing length: one slot, two slots and two TDMA rounds. The first two cases fall in the hypothesis of Lemma 2, the third in the hypothesis of Lemma 3. In the latter case, all slots of a whole TDMA round are lost, reproducing a communication blackout where no nodes are able to send any messages (and therefore no local syndromes are sent). In each of these three cases, bursts can start in any of the 4 sending slots, thus we considered 12 experiment classes.

Another experiment class aimed at validating the ability of the protocol to correctly update penalty and reward counters for a given node. A fault is injected in the sending slots of the node every second TDMA round for 20 TDMA rounds. Hence, either the penalty or the reward counter should be increased at every round.

The effect of one malicious node sending random local syndromes was also considered. Its presence is not supposed to induce the other nodes to diagnose correct nodes as faulty. As any of the four nodes can be malicious, we considered 4 experiment classes.

To validate the clique detection capabilities of the membership protocol, we placed the disturbance node between Node 1 and the rest of the cluster and disconnected the bus during the sending slot of at least another node to produce (and detect) a minority clique formed by Node 1.

9. Practical tuning of the p/r algorithm

In order to correctly discriminate between healthy and unhealthy nodes, the penalty and reward thresholds have to be tuned together with the criticality levels for each node. We now describe experiences on the tuning of our prototype for realistic automotive and aerospace settings. Table 2 summarizes the results of our tuning.

Characterizing intermittent faults. The first difficulty faced during the practical tuning of the protocol is how to characterize unhealthy nodes. The p/r algorithm resets the penalty and reward counters for a node if it does not fail for R consecutive rounds, where R is the reward threshold. If a fault appears before R is reached, it is considered correlated with the previous fault. Therefore, R should be large enough to correlate intermittent faults. The time to reappearance of intermittent faults, however, depends on the specific frequency of fault activation for each node (i.e., which hardware components of the node are damaged and how often they are stimulated by the software) and is unknown in most practical systems.

While setting R, designers must make a probabilistic tradeoff between the capability of correlating intermittent faults with a large time to reappearance and the avoidance of incorrect correlation of independent and external transient faults. In Figure 3 we show such a tradeoff for our automotive and aerospace settings, where the length of the TDMA round is set to $T = 2.5ms$. Our practical choice was to set $R = 10^6$ to correlate faults whose interarrival time is within $R \times T \cong 42min$, which can be pragmatically considered a reasonable value. After detecting a transient fault, the resulting probability of correlating a second transient fault is less than 1% considering the rates of Fig. 3. It must be noticed that a healthy node will be isolated only if P subsequent transient faults are correlated, where P is the penalty threshold [7]. In all our prototypes the probability of isolation of a healthy node is thus negligible.

Tuning the diagnostic latency. To increase availability and accumulate diagnostic data, the p/r algorithm delays node isolation and increases the *diagnostic latency*. An application can be prevented from correctly exchanging messages if some of its jobs are hosted on a faulty node that is kept operative by the p/r algorithm. In such case the application might experience an outage. Applications with different criticality classes have different requirements on

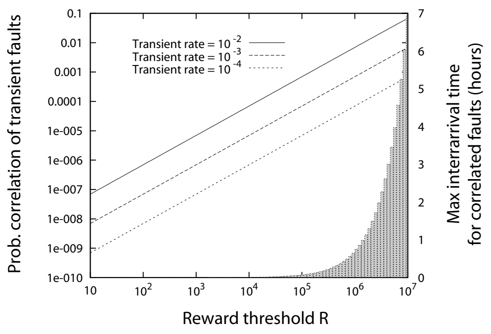

Figure 3: Setting R with rounds of $2.5ms$

Table 2: Results of the experimental tuning of the p/r algorithm

Domain	Criticality class	Example	Tolerated outage	Crit. lvl. (s_i)	P	R	TDMA
Automotive	*Safety Critical (SC)*	X-by-wire	$20-50ms$	40	197	10^6	$2.5ms$
	Safety Relevant (SR)	Stability control	$100-200ms$	6			
	Non Safety Relevant (NSR)	Door control	$500-1000ms$	1			
Aerospace	*Safety Critical (SC)*	High Lift, Landing Gear	$50ms$	1	17	10^6	$2.5ms$

the maximum *tolerated transient outage* time before a recovery action is activated in order to restore the availability of the service or to reach a safe state. Such outage is the sum of the diagnostic latency and the recovery time. Tolerated transient outages for different classes of automotive and aerospace applications are shown in Table 2.

The automotive domain depicts a varied range of criticality classes. *Safety critical* functionalities are necessary for the physical control of the vehicle with strict reactivity constraints, e.g., X-by-wire. Recovery actions must preserve the availability of the (possibly degraded) service. *Safety relevant* functionalities support the driver, e.g., the Electronic Stability Control and the Driver Assistant Systems, such as the collision warning and avoidance system. They are not necessary for the control of the car but the driver must know if they are unavailable. Finally, there are *Non Safety relevant* functionalities such as comfort and entertainment subsystems. In the aerospace domain, only safety critical functionalities are connected to the backbone. The High Lift System adds lift during the flight and is related to the control of flaps. The Landing Gear System controls the retractable wheels used for landing.

Both the diagnostic and the more complex membership service are fast enough to satisfy the requirements of the highest criticality class considered. However, we want to delay the isolation of faulty nodes as much as possible to maximize the availability in presence of transient faults. The diagnostic latency can be tuned by setting the penalty threshold and criticality levels according to the application requirements. Hence, we injected continuous faulty bursts and observed the value of the penalty counter reached when the maximum diagnostic latency for each criticality class was reached. We assumed that once a faulty node is isolated by the diagnostic protocol, each obedient node can instantaneously apply the necessary recovery actions, discounting further delays. Each experiment was repeated 100 times. If classes $c_1, \ldots, c_i$ have corresponding penalties $p_1, \ldots, p_i$, we set $P = max(p_1, \ldots, p_i)$ and the criticality of each class to $s_i = \lceil P/p_i \rceil$. To satisfy the requirements on the diagnostic latency, the criticality increment for a node was set as the maximum s_i of the applications it hosts. Criticality levels are stored in the vector $criticalities$ used by the p/r algorithm. The penalty thresholds and criticality levels for the automotive and aerospace setups are shown in Table 2. We observed in both setups that even for Safety Critical applications it is possible to wait for some round before isolating faulty nodes. This enhances the capability of the system of not overreacting to transient faults.

Table 3: Abnormal transient scenarios

Scenario	Burst	TTReapp.	# Inj.
Auto (blinking light)	$10ms$	$500ms$	50
Aero (lightning bolt)	$40ms$	$160ms$	1
	$40ms$	$290ms$	1
	$40ms$	$500ms$	9

Diagnosis under adverse external conditions. We have shown how we tuned the parameters of the p/r algorithm under *normal* external conditions. The next step was to try to evaluate the capability of the algorithm to guarantee node availability under *adverse* external conditions, characterized by an abnormal rate of transient faults. For this purpose we considered two unfavorable but common scenarios in the automotive and aerospace settings where external faults are highly frequent and will likely be considered as intermittent faults. For the automotive setting we considered a blinking light causing periodic electrical instabilities on the bus due to an open relay, while for aerospace we considered a lighting bolt producing a sequence of instabilities with increasing time to reappearance. Systems are designed and tested to tolerate such transient behaviors without taking specific recovery actions, therefore isolations should be avoided. The length of the faulty bursts, the times to reappearance and the number of instances of the burst are shown in Table 3. We reproduced these scenarios in 100 experiments and observed if and after how much time healthy nodes were incorrectly isolated.

In both cases, different transient burst are considered as correlated by the p/r algorithm. The results for the automotive and aerospace setting are shown in Table 4. The functionalities with lower criticalities can tolerate longer periods of abnormal transient behavior. The use of a p/r algorithm with varied criticality levels gives advantages in terms of availability. In fact, if nodes were immediately isolated after the first fault appearance, a single abnormal transient period would result in the isolation of all the nodes in the system and would entail a restart of the whole system. However, even using our p/r algorithm, the availability of safety critical functionalities can be harmed by relatively short disturbances in both the experimental setting. From this data we

Table 4: Time to incorrect isolation

Setting	Criticality class	Time to isolation
Automotive	*SC / SR / NSR*	0.518 / 4.595 / 24.475sec
Aerospace	*SC*	0.205sec

can conclude that the detection of intermittent faults could be sacrificed for the sake of availability for those nodes implementing safety critical functions. For example, isolated nodes could be kept under observation, collecting rewards if a fault-free behavior is observed and reintegrating the node if a specific reward threshold for reintegration is reached.

10. Portability Issues for Varied TT Platforms

One of our main design drivers was to define a diagnostic/membership protocol that is a tunable and portable add-on application level module, rather than a static and built-in system level feature. Our experience has confirmed that this approach is viable. Our protocol only uses detection capabilities that are provided by any TT platform. The concept of validity bit abstracts a number of platform specific error detection mechanisms, whose outcome can normally be accessed by applications using the basic API provided by the operating system of the host node (see Sec. 8).

Another important issue was not to require interactions or to interfere with other applications. For this reason, local detection of faults is implicitly performed by monitoring the exchange of diagnostic messages among diagnostic jobs. To ease the integration, the bandwidth requirement of the protocol is limited. In our prototype diagnostic messages were as small as N bits.

Finally, we avoided imposing strong constraints on node scheduling. The read and send alignments ensure that all diagnostic jobs use consistent data for any schedule, provided that the diagnostic jobs are executed at every round. To achieve that, they require the application to know some parameters that are directly related to the node scheduling, such as $l_{\{1,..,N\}}$ and $send\_curr\_round_{\{1,..,N\}}$ (see Sec. 5). If a static scheduling policy is used, this information is constant and known at design time. In case of dynamic scheduling we require the OS to provide this information to the application at run-time.

The relaxed constraints on the scheduling of the diagnostic jobs lead to a detection latency, i.e., the time necessary to consistently detect a faulty slot, of four TDMA rounds in the worst case, which is suboptimal. However, if needed, it is possible to trade off flexibility for a shorter latency. By constraining the internal node scheduling, in fact, we can reduce the detection latency down to one round for the diagnostic protocol and two rounds for the membership protocol. In this variant of the protocol, each node keeps sending its local syndrome at each sending slot, but the analysis is executed right after each slot and refers to a single previous slot. After one round all local syndromes necessary to diagnose a slot are collected, and two diagnostic rounds would be sufficient to execute two instances of the modified diagnostic protocol, i.e., one instance of the membership protocol. All the properties of the protocol are preserved in this variant, at the price of making portability more complex.

11. Conclusions

We have presented a generic diagnostic protocol that can be added on as a middleware layer on top of any TT platform. It tolerates multiple benign and malicious faults and aims to maximize node availability by using a p/r algorithm even under abnormal transient disturbances. We have extended it to be usable as a membership protocol without using additional resources. Both variants of the protocol have been experimentally validated. We tuned the p/r algorithm under realistic automotive and aerospace settings, and addressed open issues of characterization of intermittent faults, determination of the severity of faults and diagnosis under adverse external conditions.

References

[1] FlexRay Communication System, Protocol Specification v. 2.1. *http://www.flexray.com/specification_request_v21.php*

[2] H. Kopetz and G. Grunsteidl. TTP - A Protocol for Fault Tolerant Real Time Systems. *IEEE Computer*, 27(1), pp. 14–23, 1994.

[3] K. Hoyme and K. Driscoll. SAFEbus. *IEEE Aerospace and Electronic Systems Magazine*, 8(3), pp. 34-39, 1993.

[4] C. Constantinescu. Impact of Deep Submicron Technology on Dependability of VLSI Circuits. *DSN*, pp. 205–209, 2000.

[5] A. Bondavalli *et al.* Discriminating Fault Rate and Persistency to Improve Fault Treatment. *FTCS*, pp. 354–362, 1997.

[6] A. Bondavalli *et al.* Threshold-Based Mechanisms to Discriminate Transient from Intermittent Faults. *IEEE Trans. on Computers*, 49(3), pp. 230–245, 2000.

[7] M. Serafini *et al.* On-line Diagnosis and Recovery: On the Choice and Impact of Tuning Parameters. *TR-TUD-DEEDS-05-05-2006*, 2006.

[8] F.P. Preparata *at al.* On the Connection Assignment Problem of Diagnosable Systems. *IEEE Trans. on Electronic Computers*, 16(12), pp. 848-854, 1967.

[9] M. Malek. A Comparison Connection Assignment for Diagnosis of Multiprocessor Systems, *ISCA*, pp. 31–36, 1980.

[10] M. Barborak *et.al*, The Consensus Problem in Fault Tolerant Computing, *ACM Surveys*, vol. 25, pp. 171–220, Jun. 1993.

[11] C. Walter *et al.* Formally Verified On-line Diagnosis. *IEEE TSE*, 23(11), pp. 684–721, 1997.

[12] M.A. Hiltunen. Membership and System Diagnosis. *SRDS*, pp. 208-217, 1995.

[13] F. Cristian. Reaching Agreement on Processor-group Membership in Synchronous Distributed Systems. *Distributed Computing*, 4(4), pp. 175–187, 1991.

[14] G. Bauer and M. Paulitsch. An Investigation of Membership and Clique Avoidance in TTP/C. *SRDS*, pp. 118–124, 2000.

[15] P.D. Ezhilchelvan and R. Lemos. A Robust Group Membership Algorithm for Distributed Real Time Systems. *RTSS*, pp. 173-179, 1990.

[16] C. Walter *et al.* Continual On-line Diagnosis of Hybrid Faults. *DCCA*, pp. 150-166, 1994.

[17] A. Ademaj *et al.* Evaluation of Fault Handling of the Time Triggered Architecture with Bus and Star Topology. *DSN*, pp. 123-132, 2003.

[18] P. Lincoln and J. Rushby. A Formally Verified Algorithm for Interactive Consistency under Hybrid Fault Models. *FTCS*, pp. 402-411, 1993.