Application-Level Diagnostic and Membership Protocols for Generic Time-Triggered Systems

Marco Serafini, Péter Bokor, Neeraj Suri *TU Darmstadt, Germany,*
Jonny Vinter *SP, Sweden,* Astrit Ademaj *TU Wien, Austria,* Wolfgang Brandstätter *Audi, Germany,*
Fulvio Tagliabò *Centro Ricerche Fiat, Italy,* Jens Koch *Airbus, Germany*

***Abstract*— We present on-line tunable diagnostic and membership protocols for generic time-triggered (TT) systems to detect crashes, send/receive omission faults and network partitions. Compared to existing diagnostic and membership protocols for TT systems, our protocols do not rely on the single-fault assumption and also tolerate non fail-silent (Byzantine) faults. They run 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 configurations where functions with different criticality levels are integrated. All protocols are formally verified using model checking. Using actual automotive and aerospace parameters, we also experimentally demonstrate the transient fault handling capabilities of the protocols.**

***Index Terms*— Diagnosis, Membership, Time-Triggered Systems, Transient Faults.**

I. Introduction

In both automotive and aerospace X-by-wire applications, TT platforms such as Flexray [15], TTP/C [21], SAFEbus [18] and TT-Ethernet [19] are increasingly being adopted. Some TT platforms disclaim to provide distributed diagnostic and membership services, such as FlexRay, or utilize their specific system-level properties to develop customized solutions, like TTP/C [21] or SAFEbus. Instead, we define on-line diagnostic and membership protocols as add-on application level modules that can be integrated as plug-in middleware modules onto any TT system, without (potentially problematic [31]) interferences with other functionalities or other applications. Our protocols (a) only use network-level error detection information that is made available at the application level by TT platforms, (b) do not impose constraints on the scheduling of the system, and (c) have low bandwidth requirements. The protocols can be tuned to meet customized fault coverage and latency requirements. For TT platforms, such as FlexRay and TT-Ethernet, that do not provide a standardized diagnostic or membership protocol, our add-on protocol represents a viable and flexible solution to provide such add-on functionalities.

The key purpose of a diagnostic protocol, in particular if this is used for safety critical subsystems, is to identify faulty nodes within a small diagnostic delay. Nonetheless, a diagnostic protocol also needs to consider resource availability and to avoid declaring correct components as faulty in case of transient faults, which are becoming increasingly frequent [10]. An "ideal" diagnostic protocol would exclude only nodes with permanent internal faults. In practice, however, these 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 (e.g. sparse message omissions) which, to external observers, appear similar to transient faults.

Our diagnostic protocol uses a *penalty/reward (p/r) algorithm* to distinguish between transient fault and intermittent or permanent faults with stochastically predictable accuracy and coverage [30]. Predictability is provided by a stochastic model that considers faults not only over a single protocol run but over multiple runs (using an *extended* fault model). Different from existing diagnostic approaches which rely on system-specific heuristics for handling transient faults (e.g. [18], [34]) and like the α-count model of [6], [7], our generic p/r model can be applied and tuned to each specific implementation in a well-defined manner. However, different from α-count, our FDIR (Fault Detection, Isolation and Reconfiguration) model does not assume that the maximum duration of transient faults is bounded and known. It admits closed-form analytical solutions which can be easily evaluated by hand without using modeling tools, and it considers systems running multiple applications with varied criticalities. In this paper, we show for the first time how to integrate a p/r algorithm with an on-line distributed diagnostic protocol.

Analogous to diagnosis is the membership problem [17], [2], which consists of identifying the set of nodes (called membership view) that have received the same history of messages. We 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. Similar to diagnosis, membership protocols also need to consider availability. Our membership protocols are the first ones where the ensured consistency degree can be tuned, using the p/r algorithm, to avoid over-reactions to transient faults. The protocol is also extended to detect and tolerate both permanent and transient network partitioning.

An important aspect of our diagnostic and membership protocols is providing consistent diagnostic and membership information to all nodes even in presence of worst-case (Byzantine) faults. A common feature of TT systems is that non fail-silent faults at the network level are turned into fail-silent faults. This ensures that correct nodes can still communicate despite the presence of non fail-silent faulty nodes. SAFEbus, for example, uses double-redundant Bus Interface Units to detect and isolate non fail-silent faults [18], whereas TTP/C can use a star network configuration with redundant bus guardians [1]. For this reason, many previous membership protocols for TT systems assume only benign faults (crashes, send and receive omissions) [21], [3], [14]. Although we assume fail-silence at the network level, this does not rule out the presence of non-detected errors at the application level where

our protocols run. These errors can stem from simple memory corruptions and can result in error propagation at the protocol level if fail-silence is assumed [4]. Our generic protocols are designed to tolerate multiple application-level faults, both silent and non-silent. The number of tolerated faults grows with the total number of nodes in the system.

The properties of all presented protocols are formally verified per hand-proofs and model checking. We have also implemented the diagnostic and membership protocols in a prototype, incorporating actual automotive and aerospace parameters. Using physical fault injection, we experimentally validate the properties of the protocols under several fault scenarios and show how to tune the parameters of the p/r algorithm in a realistic environment.

Overall, our diagnostic and membership protocols are the first with the following key properties:

- Can run on every TT system at application level without imposing scheduling constraints
- Can tolerate Byzantine faults and multiple benign faults
- Are integrated with a tunable p/r algorithm for better transient fault handling
- Have been validated for several fault scenarios using actual automotive and aerospace parameters
- Have been formally verified using a model checker.

The paper is organized as follows. Following the related work in Sec. II, we introduce the system and fault models in Sec. III. The tunable add-on diagnostic protocol and its properties are presented in Sec. IV. The protocol is extended to a membership protocol in Sec. V. The results of formal verification based on model checking are reported in Sec. VI. Sec. VII describes the experimental validation of both protocols. We detail parameter tuning in Sec. VIII. Sec. IX discusses the portability of the middleware to different TT platforms.

II. Related work

The general diagnosis problem was formulated in the PMC model [27], where a set of entities test each other to collect sufficient information to locate the faulty nodes. In on-line, real-time settings the comparison approach is recommended [25], 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. [34] 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 [23], [2], 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 [34] as a middleware service for TT systems, where multiple nodes may 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, and can be adapted to both frame-based and TDMA communication schemes. Different from [34], we explicitly consider the cases of communication blackouts that can arise if particularly long transient bursts corrupt all sending slots in a round. We also show how to modify the protocol to provide membership information even in presence of network partitioning. Finally, our protocol uses the new p/r algorithm to handle transient faults based on the criticality of the applications executing on different nodes.

The problem of group membership is often defined similar to diagnosis [17]. Cristian [11] proposes 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. [21], proved correct in [3], formally verified using theorem proving in [26] and model-checked using parameterized verification in [8]. The fault model is the "single fault assumption": The protocol does not have to tolerate simultaneous faults or non fail-silent nodes. The protocol allows identification of one fault in the communication of a message and also detects the formation of different cliques of nodes which cannot communicate with each other. The latency is two communication slots in the case of sender faults and two rounds in case of receiver faults. The message complexity is $O(N)$ bits per message and $O(N^2)$ bits per round, where N is the number of system nodes. In order to save bandwidth, TTP/C implementations of the algorithm encode the messages in the CRC. If a (possibly transient) faulty node is detected it is generally restarted. However this can generate a window of vulnerability to subsequent failures. If clique detection is not executed, the message complexity of the protocol can be reduced to one bit per message [29]. An extension of the TTP/C protocol was proposed by Ezichelvan and Lemos [14] to tolerate up to half of senders being simultaneously faulty with a latency of three rounds. A more recent protocol also requires a majority of active nodes, handles benign faults, and is extended to work in systems with event-triggered scheduling [5]. Authors of [32] identify a class of non fail-silent faults which are not fully tolerated by the TTP/C membership protocol even if bus guardians are used, and fix the problem under the single fault assumption. Our protocols tolerate multiple coincident non-Byzantine and Byzantine faults and have the same message complexity as existing ones, with the exception of the partitionable membership protocol which requires $O(2N)$ bits per message. Due to their add-on and generic nature, our protocols have a higher latency.

Current TT platforms offer different levels of integration of on-line diagnosis in the system. SAFEbus compares the outcomes of double-redundant Bus Interface Units to detect and isolate arbitrary faults at the network level. An evolution of SAFEbus which achieves lower hardware costs using a braided ring topology is introduced in [16]. The Time Triggered Architecture (TTA) [20] integrates membership with clock synchronization. Experimental evaluation has pointed out some limitations of this integrated approach [31]. The TTA approach of tolerating transient upsets by detecting faulty nodes and by letting them converge to a correct state is discussed in [32]. Such rejuvenation techniques are orthogonal to our work, which only focuses on fault detection, but can be easily integrated with it. The current FlexRay specification [15] allows the use of a Network Management Vector whose functionalities are analogous to membership, and leaves it up to the applications to implement this functionality. Our protocols represent a viable and generic solution in this context.

III. System and Fault model

System Model. We assume a synchronous distributed system model where there are known upper bounds on the execution time of jobs and on communication delays. We assume that each correct node is equipped with a local clock of high precision, i.e., the drift between clock time and physical time is bounded. Clocks of different nodes are synchronized. In the traditional synchronous, or frame-based, computational model [24], [28], nodes execute synchronous rounds in a lock-step manner. During each round the nodes alternate two phases, first send and receive their messages in parallel (communication phase), and only then compute the received messages in parallel (computation phase). Many TT systems, however, do not enforce this strict lock-step model. In order to maximize the utilization of the computational resources of a node, communication and computation within one node are executed in parallel if possible. These systems also allow overlapping computation and communication phases at different nodes in order to use cheaper shared communication buses which require sequential access.

We consider a generic model that encompasses most TT platforms. The system consists of N nodes having unique IDs $\{1, ..., N\}$. We assume that each node is assigned a time window, called *sending slot*, in a larger periodic time window called *round*. Each node has exactly one sending slot per round. We assume without loss of generality that node IDs are assigned consistently with the order of the sending slots in the round: Node j does not complete its sending slot before node i if $i < j$. A round starts with the beginning of its first sending slot. The periodic *global communication schedule* defines when each slot begins and terminates. It is static and defined at design time. For example, in classic frame-based systems all nodes have the same sending slot, whereas sending slots of different nodes never overlap in TDMA systems such as [21] or [15].

Each node has a *communication controller* which executes the global communication schedule. We refer to the subsystem composed by the communication controller and the communication buses as *network level*, while the rest of the system is the *application level*. Network-level communication among jobs running on different nodes is abstracted through a vector of shared variables $\langle v_1, \ldots, v_N \rangle$, called *interface variables*, available at each node. Variable v_i can only be written by jobs running on node i. When the sending slot of node i is reached, the value of v_i at node i is broadcasted by the communication controller of i. All other nodes update their local copy of the variable as soon as the sending slot is completed. Nodes can identify the correct senders of the messages they receive.

The network can not undetectably forge or corrupt messages. Error detection on the value domain is usually ensured by sending redundant data along with the message, for example by adding CRC codes. Also, due to the synchrony of the system, it is always possible to accurately detect missing messages by using timeouts. In all TT platforms, the communication controllers detect errors at network level and signal them to the applications through *validity bits* that are paired to each interface variable. The communication controller of node j sets the validity bit of v_i to 0 iff node j was not able to receive the last message sent by i that was supposed to update v_i, and to 1 otherwise. For $i = j$, the communication controller checks if it can itself read the messages it sends, for example by using a local collision detector.

Besides the global communication schedule, each node has its own internal *node schedule* that determines when jobs are executed in the round. Similar to the global communication schedule, the node schedule is static and defined at design time. The node schedule can have an effect on the "freshness" of the interface variables, which is the round where the values of the interface variables were sent. For example, if a job is executed during a round it may read some values sent in the previous rounds and some values send in the current one. The node schedule also determines the freshness of the data sent on the bus. A job running on node i which is started at round k and writes data onto its interface variable is able to send this data during the same round k only if it is scheduled to start before the beginning of the sending slot of i in k. To increase the portability of our add-on protocol, our protocol is parameterized and does not constrain the scheduling of nodes. Diagnostic jobs running at node i have two parameters, l_i and $send\_curr\_round_i$, to represent different node schedules. The use of these parameters is discussed in detail in Section IV.

Fault Model. Nodes are categorized based on the faults they encounter. *Correct nodes* follow the specification of our protocols, have a correct state, and can correctly communicate with the other correct nodes. *Obedient nodes* follow the specification of our protocols and have a correct state, but they may suffer detectable faults at the network level which prevent them from sending messages to any node. Correct nodes are thus also obedient. *Byzantine nodes* do not follow the specification of our protocols and can have any state. Note that there is no error propagation at the network level. Faulty nodes are fail-silent at the network level and therefore can not disrupt the network-level communication among other nodes.

Faults refer to communication errors as they are observed at the application level. They are partitioned into three classes, based on their severity and symmetry [33]: Symmetric benign, symmetric Byzantine and asymmetric. Symmetric benign (or simply benign) faults produce errors which can be locally detected by all nodes in the system. We say that there is a *local error detection between node i and node j at round r* if j is obedient, i broadcasts a message at round r, and the validity bit of the local copy of the variable v_i held by j is set to 0 after the sending slot of i at round r is completed. We say that *node i suffers a symmetric benign fault*, or a *send omission*, at round r if there is a local error detection between i and all obedient nodes at round r. For example, a crashed node permanently displays benign faults. Obedient nodes can also be benign faulty.

Node i suffers a *symmetric Byzantine fault at round r* if i sends an erroneous message at round r which is not conformal to the system specification, all obedient nodes receive the same erroneous message from i at round r, and there is no local error detection between i and any other obedient node at round r. An example of symmetric Byzantine fault is when an undetected memory error in node i corrupts some value which is then written into v_i and broadcasted to all other nodes. Network-level error detection can not determine that the message is erroneous, so the corrupted value is consistently received by all obedient nodes.

Node i suffers an *asymmetric fault at round r* in all remaining cases where i sends a message and either there is a local detection between i and some obedient node j in round r, or some obedient node j receives in round r an erroneous message from i which is not conformal to the system specification. In the worst case, asymmetric faults can be unconstrained Byzantine faults [22].

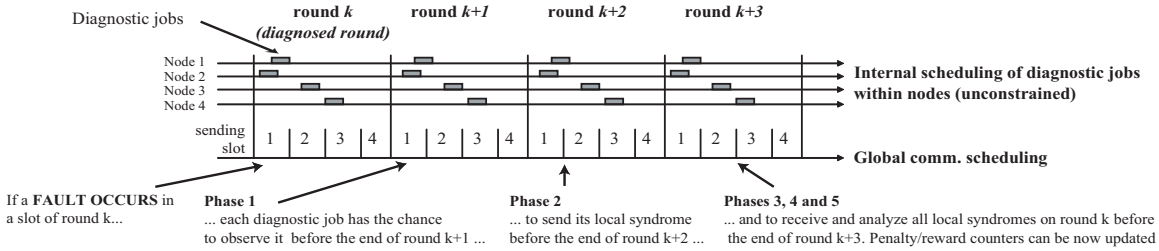

Fig. 1. High level overview of the diagnostic protocol in a system with TDMA access and four nodes

We say that an obedient node j suffers a *receive omission* at round r if some node i is asymmetric faulty and there is a local error detection between i and j. An asymmetric fault can result in any number of receive omissions. Asymmetries in the reception of messages can be an effect, for example, of Slightly-Off-Specification faults (SOS) [1], 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 electro-magnetic interferences disturb only part of the bus.

We classify nodes based on the faults they encounter over one execution of the protocol. If a node displays different type of faults, we consider the most severe one — benign being the least severe and asymmetric the most. Let a, s and b represent the maximum number of asymmetric, symmetric Byzantine and benign faulty nodes over one execution. Our diagnostic and membership protocols of Sections IV and V assume that the number of nodes in the system is $N > 2a + 2s + b + 1$ and that $a \leq 1$ if $s + a > 0$. No assumption on b is needed if $s + a = 0$.

A membership protocol to tolerate permanent and transient network partitions is discussed in Appendix I.

IV. The on-line diagnostic protocol

The purpose of the on-line diagnostic protocol is to detect and exclude faulty nodes from the system at run-time. The protocol outputs at each round the set of *active nodes* which are deemed operational, and requires no inputs from other applications. 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.

At each round, each node i runs 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 at round k is the binary N-tuple containing the validity bits of the messages sent to i at round $k-1$. The diagnostic message dm_i contains the local syndrome broadcasted 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 *broadcasted* using the diagnostic message dm_i.
3) *Aggregation*: Collect 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 [34], [23] over the *columns* of the diagnostic matrix is performed.
5) *Update counters*: Based on the consistent health vector, update the *penalty and reward counters* associated to nodes and possibly eliminate faulty nodes from the active ones.

The phases of the protocol are executed in consecutive 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. For simplicity of presentation the pseudo code assumes that the round number k is known by the algorithm, although this is not necessary in the actual implementation if the values of some variables in the last two rounds are buffered.

Consistent location of faulty senders. Local detection and aggregation entail reading the interface variables and their validity bits. As we do not constrain the scheduling of the diagnostic jobs in a round, we need to consider that diagnostic jobs running on different nodes at the same round can read different values with different freshness from the same interface variables.

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 at round k or $k-1$. Hence there is a locally known integer $l_i \in [1, N]$, determined by the internal schedule of $diag_i$ within node i, such that values of $dm_1, \ldots, dm_{l_i}$ were sent at round k, while values $dm_{l_i+1}, \ldots, dm_N$ were sent at round $k-1$ (see Fig. 2). The same holds for the validity bits of the messages. In Figure 1, for example, $l_1 = l_2 = 0$, $l_3 = 1$ and $l_4 = 2$. Note that from the system model a diagnostic job $diag_i$ which is executed between the completion of the last sending slot of round k and the beginning of the first sending slot of round $k+1$ is treated as it was executed at round k and l_i is thus set to N. Therefore, in a frame-based system with synchronous rounds we have $l_j = N$ for all nodes j.

For all diagnostic jobs executed at round k to consistently

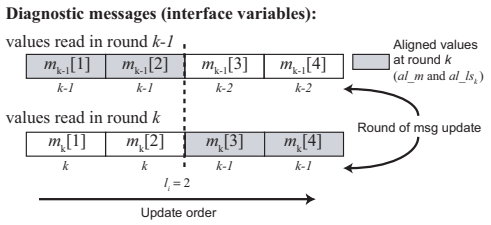

Fig. 2. Example of read alignment (round k, $l_i = 2$)

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 $al\_m[j]$ (resp. $al\_ls_k[j]$) values $m_{k-1}[1,\ldots,l_i]$ (resp. $ls_{k-1}[1,\ldots,l_i]$) read in the previous round $k-1$ and of $m_k[l_{i+1},\ldots,N]$ (resp. $ls_k[l_{i+1},\ldots,N]$) read in the current one. This requires buffering of messages and validity bits of the last two or three rounds, and introduces additional delays in the communication. If all nodes can read the messages sent in the current round, the constant u is set to 0 and no buffering is used. The latency of the protocol is reduced in this case.

For *local detection*, the validity bits are copied into the vector ls_k (line 2) and combined using read alignment (lines 3–6). The vector al_ls_k contains at round k the local syndromes corresponding to the messages sent at round $k-1$.

During the *dissemination* phase a *send alignment* is also needed to ensure that, despite unconstrained node scheduling, all local syndromes sent at round k refer to a same previous diagnosed round, as required by the aggregation phase. We define the predicate $send\_curr\_round_i$ to be true if, according to the internal schedule of node i, the diagnostic messages formed by the diagnostic job $diag_i$ after being started at round k can be sent at round k, that is, $diag_i$ writes onto its interface variable v_i before the sending slot of i in k. In Fig. 1, for example, $send\_curr\_round_1$ does not hold, but it holds for all other nodes. In a frame-based system with synchronous rounds $send\_curr\_round_i$ does not hold for any node. This is because each computational phase is followed by a communication phase in the next round.

As rounds start with the beginning of their first sending slot, $send\_curr\_round_i$ does not hold for at least one diagnostic job. Send alignment is thus used to let all nodes consistently wait one round before sending the local syndromes computed in a round (line 7). If a job completes its execution after the sending slot of its node, it writes its current aligned local syndrome (line 8). This ensures that all local syndromes sent in a round are computed in the same round. In frame-based systems with synchronous rounds $send\_curr\_round_i$ does not hold for any node and all nodes write their current local syndromes for round k. These are all consistently sent it in the next round.

The *aggregation* phase first copies into the vector m_k the values of the local syndromes sent by all diagnostic jobs through the diagnostic messages (line 2). 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 at round k form a diagnostic matrix using local syndromes sent at round $k-1$, which refer to the same diagnosed round (lines 3 – 6); vector $al\_m[j]$ represents the j^{th} row of the matrix. The j^{th} element of the local syndrome sent by node i to node n 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

Algorithm 1: Node i diagnostic job $diag_i$ at round k

```
1  const u = 0 if ∀j : l_j = N; else u = 1;
   // Phases 1 and 3 - Local detection, Aggregation
   // (read alignment)
2  m_k, ls_k ← read_iface&vbits(dm_1, ..., dm_N);
3  for j ← 1, ..., l_i do
4      al_m[j] ← m_{k-u}[j];   al_ls_k[j] ← ls_{k-u}[j];
5  for j ← l_i + 1, ..., N do
6      al_m[j] ← m_k[j];   al_ls_k[j] ← ls_k[j];
   // Phase 2 - Dissemination
   // (send alignment)
7  if send_curr_round_i then write_iface(al_ls_{k-1});
8  else write_iface(al_ls_k);
   // Phase 4 - Analysis
   // (consistent location of benign faulty senders)
9  for j ← 1, ..., N do
10     cons_hv[j] ← H-maj⟨al_m[1][j], ..., al_m[j-1][j],
       al_m[j+1][j], ..., al_m[N][j]⟩;
11 if ∃l : cons_hv[l] = ⊥ then cons_hv ← al_ls_{k-u-1};
   // Phase 5 - Update counters (and isolate)
   // (decision on node isolation)
12 active ← p_r(cons_hv, active);
13 return active;
```

n was not able to receive the local syndrome from i correctly. For example, Table I shows the diagnostic matrix formed in case node 3 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. Thus, voting is executed over the $(N-1)$-tuple V of local syndromes representing the opinions of the other nodes (lines 9 – 11). In order to tolerate benign faults, a hybrid voting function excludes erroneous votes ε from V ($excl(V,\varepsilon)$) before calculating the majority [23].

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

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 d_round, where the value 0 denotes a faulty node. In case no correct vote is available ($|excl(V,\varepsilon)| = 0$), the voting function cannot reach a

TABLE I
EXAMPLE DIAGNOSTIC MATRIX (3-4 BENIGN FAULTY)

		Accused node			
Accuser node	**Local syndr.**	*1*	*2*	***3***	***4***
Node 1	$al\_m[1]$	-	1	0	0
Node 2	$al\_m[2]$	1	-	0	0
Node 3	$al\_m[3]$	ε	ε	-	ε
Node 4	$al\_m[4]$	ε	ε	ε	-
	Voted *cons_hv*	1	1	**0**	**0**

decision. From the fault assumption, this implies that $a + s = 0$ and thus the local syndrome represents a correct and consistent value for the health vector.

Filtering unhealthy nodes. The consistent health vector is given as an input (Alg. 1, line 12) to the p/r algorithm (Alg. 2). The p/r algorithm considers an *extended fault model* where all nodes alternate periods of faulty and correct behavior [30]. A node is *healthy* if it suffers only sporadic and external transient faults, and *unhealthy* if it suffers internal faults which manifest as intermittent or permanent communication faults. The model implicitly assumes that internal faults will manifest at the interface of the node as either permanent sender faults (a long faulty burst), or intermittent faults with a shorter time to reappearance than external transient faults.

The p/r algorithm uses the consistent health vector to eliminate unhealthy nodes from the set of *active* nodes $active$, which is a vector with one entry for each node. Entries of $active$ are initially set to 1 and are later set to 0 when a node is excluded from the set of active nodes. Each node keeps a penalty and a reward counter for each node in the system in the vectors $penalties$ and $rewards$. All counters are initially set to 0. Every time a new consistent health vector is available, penalties and rewards are updated for each node (Alg. 2, line 2). 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 thus exclusions can be consistently decided in the same round by all obedient nodes.

When a node j is deemed faulty (Alg. 2, lines 4 – 7), the corresponding penalty is increased by a value $criticalities[j] \geq 1$. If the penalty value of j exceeds a pre-defined *penalty threshold* $P > 0$, j is eliminated from the set of active nodes. Criticalities higher than 1 are used to speed-up the exclusion of nodes hosting safety-critical applications. The entry of a node j in the vector $criticalities$ is defined based on the highest criticality class among all the applications hosted by j, as described in Section VIII and [30].

If a node j is deemed correct, (Alg. 2, lines 4 – 12), its reward value is increased by one. If the reward value of j exceeds a pre-defined *reward threshold* $R > 0$, both its penalty and reward value are reset to 0.

The penalty and reward thresholds, and the corresponding counters, represent two different kinds of information: the reward 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 threshold bounds the maximum number of consecutive faulty slots a node is allowed to display before isolation. After a bounded amount of time either of thresholds is exceeded, resulting in isolation of the node or reset of the counters respectively. The penalty (resp. reward) threshold must be tuned to maximize (resp. minimize) the probability of excluding unhealthy (resp. healthy) nodes from the set of active nodes. These two objectives require making a complex tradeoff [30]. In Section VIII we show how to do this in a practical system.

Exclusion, reintegration and startup. The vector $active$ contains the status of activity of each node and represents the output of the protocol (Alg. 1, line 12). Any traffic generated by isolated nodes is ignored by the diagnostic jobs. Isolated nodes thus become equivalent to benign faulty nodes. Upon reintegration of an inactive node, the value of the corresponding element is set back to the initial value 1 (up) and its traffic is considered again.

Algorithm 2: The p/r algorithm

```
1  inputs: (cons_hv, active);
2  for j ← 1, ..., N  do
3      if active[j] = 1 then
4          if cons_hv[j] = 0 then
5              penalties[j] ← penalties[j] + criticalities[j];
6              rewards[j] ← 0;
7              if penalties[j] ≥ P then active[j] ← 0 ;
8          else
9              if penalties[j] > 0 then
10                 rewards[j] ← rewards[j] + 1;
11                 if rewards[j] ≥ R then
12                     penalties[j] ← 0;    rewards[j] ← 0 ;
13 return active;
```

A detailed exposition of reintegration techniques are outside the scope of this paper, so the algorithm only sets activity bits to 0 (isolated). In order to be reintegrated into the set by our application-level protocol, a node first has to be correctly reintegrated into the TT platform at the network level. After this is done, reintegration must fulfill three tasks. First, the new node has to know the current set of active nodes. Second, the node has to acquire the current value of the penalty and reward values for all the other nodes in the system. Third, the other nodes must include the reintegrating node into the set. The criticalities and the penalty and reward thresholds for all nodes are statically assigned at design time.

On-line reintegration can entail additional bandwidth and computational costs. Other existing protocols for TT systems [21], [3], [14] let nodes exchange their active sets, or membership views, at each round. This information is used both for detecting and excluding faulty nodes and for on-line reintegration. Our transient fault handling approach decouples the diagnosis of faults from the decision on node exclusions, and requires that these two types of information are sent separately. Sending the active sets stored by the nodes requires additional $O(N)$ bits per message. After joining the TT network, the reintegrating node first tries to find out the set of active nodes. In order to tolerate Byzantine nodes sending incorrect active sets, the reintegrating node must receive active sets from all nodes and execute hybrid voting.

Reintegrating nodes also need to recover the current penalty and reward counters for all the active nodes identified in the previous step. Nodes send counter information of only a single node at each round to reduce bandwidth consumption. Further reduction is achieved by sending the penalty and reward value of a node over three rounds. First the ID of a node is sent, then its current penalty value, and finally its reward value. Reintegrating nodes collect this information from all active nodes and executes hybrid voting to tolerate Byzantine faults.

Penalty and reward values of active nodes are continuously updated during reintegration. Reintegrating nodes receive diagnostic messages, calculate the consistent health vector, and update the penalty and reward counters they have already recovered. However, they do not send protocol messages until they have recovered all the necessary information. Nodes external to the active set can be reintegrated in the set as soon as they begin to send protocol messages and are consistently diagnosed as correct by all other active nodes for the first time. The presence of nodes which are external to the active set can be tolerated by the protocol since they are seen as benign faulty by the other nodes which are already active. The algorithm can easily handle

nodes executing startup as its application-level jobs can participate in the protocol only after their TT communication controller has completed startup. Nodes whose startup is not yet completed are simply seen as benign faulty by the other nodes in the active set. Overall, our protocol can provide fault handling for transient and Byzantine faults at the costs of higher bandwidth requirements for on-line reintegration.

Properties of the diagnostic protocol. In this section we prove the properties of the diagnostic protocol, which are:

- *Correctness*: a correct sender is never diagnosed as faulty in the consistent health vector of any obedient nodes;
- *Completeness*: a benign faulty sender at round $k-2u-1$ is always diagnosed as faulty in the consistent health vector of all obedient nodes at round k;
- *Consistency*: the consistent health vector is agreed by all obedient nodes in each round.
- *Consistent Isolation*: the set of active nodes is agreed by all obedient nodes in each round.

The diagnostic delay for completeness, which is $2u+1$ rounds, depends on the constant $u \in \{0,1\}$ defined in Alg. 1. If read alignment is used then u is equal to 1 and this reflects the additional delay given by buffering messages.

We remark that in our extended fault model these properties hold for obedient nodes, i.e., both correct nodes *and* nodes encountering omission faults. These properties imply that an obedient node is able to exclude itself from the set of active nodes and become silent.

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 four properties defined above.

***Lemma** 1: If node i is obedient and $diag_i$ reads the vector $m_k \neq \varepsilon$ (resp. the vector ls_k) at round k then $m_k[j]$ with $j \in \{1 \dots l_i\}$ contains the diagnostic message dm_j sent by node j to i at (resp. contain the validity bits of the messages sent by all nodes to i at) round k, while $m_k[j]$ with $j \in \{l_i+1 \dots N\}$ contains the diagnostic message dm_j sent by node j at (resp. contain the validity bits of the messages sent by all nodes to i at) round $k-u$.*

***Proof**:* From the definition of l_i, the values of the interface variables with index in $\{1 \dots l_i\}$ read by $diag_i$ at round k (Alg. 1, line 2) were sent in round k, while the values of the other interface variables were sent in round $k-u$. The same holds for the corresponding validity bits. The existence of l_i follows from the system model. Interface variables and validity bits are updated at every round and are immediately updated when the sending slot is completed. Also, node IDs, and thus interface variables too, are given indexes which follow the order of completion of the sending slots of the nodes. □

***Lemma** 2: If node i is obedient and the vector $al\_m \neq \varepsilon$ (resp. al_ls_k) is computed by $diag_i$ at round k then $al\_m[j]$ (resp al_ls_k) contains the diagnostic message dm_j sent by node j to i at (resp. the validity bits of the messages sent by all nodes to i at) round $k-u$.*

***Proof**:* This follows from Lemma 1 and from the fact that read alignment (Alg. 1, lines 3 – 6) copies into the vector $al\_m \neq \varepsilon$ (resp. al_ls_k) the values (resp. the validity bits) of the interface variables with indexes $\{1 \dots l_i\}$ as they are read in round $k-u$, and copies the values (resp. the validity bits) of the remaining interface variables as they are read in round k. □

***Lemma** 3: For all obedient nodes i, if $diag_i$ sends at round k a diagnostic message dm_i, this contains the values of al_ls_{k-1}.*

***Proof**:* The predicate $send\_curr\_round_j$ can not be true for all nodes j since this would imply that at least one job $diag_i$ is executed before the first sending slot of each round k and after the last sending slot of round $k-1$. In this case, by definition, we consider $diag_i$ as executing in round $k-1$ and therefore $send\_curr\_round_i$ is false, a contradiction.

If $send\_curr\_round_j$ is true only for a proper subset S of the nodes then for each obedient node $j \in S$, $diag_j$ writes into v_j at round k the value al_ls_{k-1}, which is sent at round k by definition of $send\_curr\_round_j$ (Alg. 1, line 8). For all other obedient nodes $i \notin S$, $diag_i$ writes into v_i at round $k-1$ the value al_ls_{k-1}, which is sent at round k by definition of $send\_curr\_round_i$ (Alg. 1, line 7). □

***Lemma** 4: If nodes i and j are obedient, the value of $al\_m[j] \neq \varepsilon$ computed by $diag_i$ at round k contains al_ls_{k-u-1} computed by $diag_j$ at round $k-u-1$.*

***Proof**:* From Lemma 3 the diagnostic job $diag_j$ sends at round $k-u$ a diagnostic message dm_j containing the value of al_ls_{k-u-1}. From Lemma 2, the value of dm_j sent by j at round $k-u$ is copied in $al\_m[j]$ by $diag_i$ at round k. □

***Lemma** 5: If there exists a node j and a value v such that for all the obedient nodes i, $diag_i$ computes $al\_ls_{k-u-1}[j] = v$ then for each obedient node i the value of $cons\_hv[j]$ calculated by $diag_i$ at round k is v.*

***Proof**:* Let us assume by contradiction that there exists an obedient node i such that $diag_i$ computes $cons\_hv[j] = v' \neq v$ at round k. The hybrid majority calculated by $diag_i$ at round k (Alg. 1, line 10) either return $\perp$ for some entry value or it does not. If it does, then $v' = al\_ls_{k-u-1}[j] \neq v$ (Alg. 1, line 11), a contradiction. Let us now consider the case when the hybrid majority does not return any $\perp$ entry in $cons\_hv$. It follows from Lemma 4 and from the hypothesis that for all obedient nodes $l \neq j$, the value of $al\_m[l][j]$ computed by $diag_i$ at round k contains either v or ε. Let $b' \leq b$ the number of obedient nodes which suffer send omissions while sending the diagnostic messages used by $diag_i$ to compute the entries of $V = \langle al\_dm_1[j], \dots, al\_dm_{j-1}[j], al\_dm_{j+1}[j], \dots, al\_dm_N[j] \rangle$ at round k (Alg. 1, lines 3 – 6). By definition, there are at least $N-a-s-b'-1$ correct nodes such that V has as many entries equal to v and at least b' entries equal to ε. It follows from the fault assumption $N > 2a+2s+b+1$ that a hybrid majority for v exists in this case as the entries from the correct nodes are a majority among the non-ε entries, that is, $N-a-s-b'-1 > (N-b'-1)/2$. □

***Theorem** 1: The consistent health vector $cons\_hv$ calculated by each obedient node at round k guarantees* correctness*,* completeness *and* consistency.

***Proof**:* By definition, no obedient node sets its validity bit to 0 for a correct node or to 1 for a benign node. Correctness and completeness thus follow from Lemmas 2 and 5.

Let us assume by contradiction that consistency is violated and that two obedient nodes calculate two different $cons\_hv[j]$ entries for the same node j. Since H-maj is a deterministic function, the two nodes must have voted on different vectors $V = \langle al\_m[1][j], \dots, al\_m[j-1][j], al\_m[j+1], [j], \dots, al\_m[N][j]$. This implies that there exists at least one asymmetric faulty node $l \neq j$ such that value of $al\_m[l]$, which is taken from a diagnostic

messages dm_l sent by node l, is received differently by the two obedient nodes. However, as $a \geq 1$ by assumption, the node j is either correct, or benign, or symmetric Byzantine. In all these cases all obedient nodes set their validity bit of the message sent m by j at round $k-2u-1$ to the same value v by definition. From Lemma 2, $al\_ls_{k-u-1}[j]$ is the validity bit for m. From Lemma 5, v is contained in the $cons\_hv[j]$ calculated by all obedient nodes, a contradiction. □

Since the p/r algorithm deterministically updates penalties and rewards according to the consistent health vector, the following theorem follows as a corollary of Theorem 1.

***Theorem** 2: The active vector calculated by each job at round k guarantees* consistent isolation. □

V. The tunable membership protocol

A common approach to fault tolerance in distributed systems is to use a *group membership* service to preserve node consistency and to trigger recovery actions and reconfigurations. Similar to the diagnostic protocol, membership outputs a set of active nodes, called the *view* in membership protocols, and requires no inputs from other applications. Our membership protocol differs from diagnosis as it does not only detect send omissions but also *receive* omissions. When an asymmetric fault occurs, nodes may be partitioned into two sets, termed as *cliques*, such that only members of one clique received the message from the faulty node. The membership protocol is able to detect the formation of cliques and to consistently diagnose them in the consistent health vector.

Membership protocols support replication of deterministic applications by identifying at each time the set of nodes, called *view*, that have received the same history of messages. This guarantees that the deterministic jobs hosted by the nodes have a consistent internal state. Full inter-replica consistency requires, for example, that nodes suffering from even a single receive omission are excluded from the view. In fact, retransmitting lost messages in a timely manner is not always possible and is also complex and resource-consuming. However, a membership protocol that does not overreact to transient faults needs to relax its consistency guarantees. Our membership protocol features a new consistency property, called *tunable view synchrony*, which results from the combination of a clique detection protocol and the p/r algorithm.

Compared to the diagnostic protocol of Section IV, this membership protocol detects a larger range of omission faults. Also, different from existing membership protocols which assume restricted failure models, our protocol keeps a consistent view among all nodes even in presence of asymmetric Byzantine faults. A limitation of the approach is that it is impossible to distinguish between benign receive omissions and Byzantine faults which creates cliques by not sending messages, or by sending inconsistent local syndromes, only to a subset of the nodes in the current view. In worst-case scenarios, these faults can thus in principle impact availability. Our protocol is however practical because it is designed for closed systems where asymmetric Byzantine faults at the application level have a random nature and are sporadic.

A version of the membership protocol which also tolerates partitions is presented in Appendix I.

Location of cliques and relaxed consistency. The pseudo code of a membership job $memb_i$ running on node i is shown in Alg. 3. It is a modification of the diagnostic protocol of Section IV which introduces *minority accusations* to detect and locate minority cliques. In the diagnostic protocol, the analysis phase produces an agreed consistent health vector which uniquely identifies the set of messages that did not suffer from send omissions. In case of receive omissions where only a clique of nodes can receive a certain message, an obedient node j that is member of a minority clique can be identified because its local syndrome disagrees with the consistent health vector on the health of some other node h. In order to accuse j, the new protocol executes the analysis and assigns minority accusations before the formation and dissemination of aligned local syndromes al_ls_k (Alg. 3, lines 10 – 11). Members of the minority clique are then consistently deemed faulty when the analysis of al_ls_k is executed. The reason for disseminating and agreeing on minority accusations rather than using them directly to calculate the new view is to prevent inconsistency on the view if different versions of the local syndromes are disseminated by a Byzantine node j. In this cases only a subset of obedient nodes may assign minority accusations to j.

The outcome of clique detection and location is a consistent health vector $cons\_hv$ which accuses all members of a minority clique. In order not to exclude a node from a view when transient omissions occur, it is possible to delay view changes and thus relax consistency. This is made possible by combining the clique detection protocol and the p/r algorithm. The resulting membership view vector $memb\_view$ identifies the current members of the view and represents the output of the protocol (line 14). Different tunings of the p/r algorithms not only result in a different likelihood of node isolation but also allow to customize the degree of consistency among nodes. If the penalty threshold P is set to 1, nodes are excluded after every single message omission, and full consistency is ensured. When P is greater than 1, the new property of tunable view synchrony ensures that the extent of the divergence among nodes in the same view is bounded. A node in the current view is considered to be fully consistent with the others after it is member of the majority clique for a sufficient number of rounds, regardless of its previous faults.

The relaxed consistency semantics of tunable view synchrony is suitable for all applications of our actual aerospace and automotive setups as detailed in Section VIII. In general, we believe that many applications for TT systems, especially control applications, can benefit from tunable view synchrony to improve node availability in the presence of transient faults.

Properties of tunable membership. In this section we prove that the views identified by the membership vector $memb\_vect$ satisfies two fundamental properties: *tunable membership liveness*, that guarantees the activation of view changes when inconsistencies arise, and *tunable view synchrony*, that ensures that all consistent nodes are included in the view. In order to formally define these properties some auxiliary definitions are first needed.

A *view* is a set of nodes which initially includes all nodes in the system. The membership view vector $memb\_view$ flags the members of the current view with a 1 in the corresponding entry. The aim of the protocol is to include in the view exactly the nodes that are "sufficiently" consistent with each other.

Send and receive omissions on a message m partition the view into two subsets, or cliques, of nodes that have (resp. have not) access to the updated values of some interface variables. These are called majority and minority cliques based on the consistent diagnosis of the sender of m resulting from the consistent health vector. Formally, we say that a node i is *diagnosed as correct*

Algorithm 3: Node i membership job $memb_i$ at round k

```
1  const u = 0 if ∀j : l_j = N; else u = 1;
   // Local detection and Aggregation
2  m_k, ls_k ← read_iface&vbits(dm_1, ..., dm_N);
3  for j ← 1, ..., l_i do
4      al_m[j] ← m_{k-u}[j];   al_ls_k[j] ← ls_{k-u}[j];
5  for j ← l_i + 1, ..., N do
6      al_m[j] ← m_k[j];   al_ls_k[j] ← ls_k[j];
   // Analysis
7  for j ← 1, ..., N do
8      cons_hv[j] ← H-maj⟨al_m[1][j], ..., al_m[j − 1][j],
       al_m[j + 1][j], ..., al_m[N][j]⟩;
9  if ∃l : cons_hv[l] = ⊥ then cons_hv ← al_ls_{k-u-1};
   // Minority accusation
10 for j ← 1, ..., N do
11     if (cons_hv ≠ al_m[j]) then al_ls_k[j] ← 0;
   // Dissemination
12 if send_curr_round_i then write_iface(al_ls_{k-1});
13 else write_iface(al_ls_k);
   // Update counters (and change view)
14 memb_view ← p_r(cons_hv, memb_view);
15 return memb_view;
```

(resp. faulty) by an obedient node j at round r if $memb_j$ computes $cons\_hv[i] = 1$ (resp. $cons\_hv[i] = 0$) at round $r + 2u + 1$. We say that an obedient node i is *member of the minority clique at round* r if i is benign faulty at round r, or there is no local error detection nor minority accusation at round r between i and a node j which is diagnosed as faulty at round r by some obedient node l, or there is a local error detection or a minority accusation at round r between i and a node j which is diagnosed as correct at round r by some obedient node l.

The consistency property of the protocol refers to a suffix of the history of messages sent in the last rounds. Given an obedient node i, we call *divergence set of i with recovery latency d after round k* $div_i(k, d)$ the set of rounds $k' \leq k$ such that i is in a minority clique in k' and there exists no d consecutive rounds $k' < k'' \leq k$ where i is in a majority clique in k''. The cardinality of the divergence set of i multiplied by its criticality level $criticalities[i]$ is called *divergence degree with recovery latency d after round k*. Divergence degree is an important concept because it is used to tune the consistency provided by the membership protocol. In presence of simple message omissions, nodes with a divergence degree equal to zero have received exactly the same history of messages in the last R (or more) rounds.

The membership view vector calculated by the membership protocol Alg. 3 must satisfy the following properties given a penalty threshold $P > 0$ and reward threshold $R > u + 1$:

- *Tunable Membership Liveness*: If after any round $k - 3u - 2$ the divergence degree with recovery latency $R - u - 1$ of any obedient node i in the current view v is greater or equal to $2P$, all obedient nodes agree upon a new unique view $v' \subseteq v \setminus \{i\}$ at round k;
- *Tunable View Synchrony*: If a new view v' is formed after a view v at round k, it includes all obedient nodes in v whose divergence degrees with recovery latency $R+u+1$ in rounds $\leq k - 2u - 1$ are smaller than $\lceil P/2 \rceil$.

Tunable membership liveness imposes that a new view is established after a loss of consistency. The degree of the allowed inconsistency can be tuned using the parameters of the p/r algorithm. Tunable view synchrony ensures that after a new view v' is established, which might take up to five rounds, all nodes in the new view v' have a maximum divergence smaller than P.

Thanks to the similarity between the diagnostic and the membership protocols, the following proofs of correctness can use some results of Section IV. As read and send alignment are the same in Alg. 1 and Alg. 3, Lemmas 1–3 hold in both protocols, with the exception of the result of Lemma 2 for al_ls_k which only holds if al_ls_k is not modified by minority accusations. Furthermore, the hybrid majority function is the same in Alg. 1 and 3, so Lemma 5 holds also for membership. The result of Lemma 2 for al_ls_k is extended to minority accusations by the following Lemma.

***Lemma** 6: If nodes i and j are obedient, the value $al\_ls_k[j]$ computed by $memb_i$ at round k is 0 if and only if* (i) *the validity bit of the message sent by j to i at round $k - u$ is 0 or* (ii) *the diagnostic message dm_j sent by j to i at round $k - u$ is different from $cons\_hv$.*

***Proof**:* We first prove that only if implication. Assume by contradiction that $al\_m[j][l]$ computed by $memb_i$ at round k is 0 but (i) and (ii) are false.

The value of $al\_ls_k[j]$ can be set to 0 either during read alignment (Alg 3, lines 2 – 6) or during minority accusations (Alg 3, lines 10 – 11). In the first case, Lemma 2 holds for Alg. 3 as we do not consider minority accusations and this implies that (i) is true, a contradiction. Therefore the value of $al\_ls_k[j]$ is set to 0 by $memb_i$ through the minority accusation because $cons\_hv \neq al\_m[j]$ at round k. Since (i) is false the value of $al\_m[j]$ computed by $memb_i$ at round k is not ε and, from Lemma 2, it contains the diagnostic message dm_j sent by j to i at round $k - u$. Therefore (ii) is true, a contradiction.

The if implication follows from the fact that obedient nodes only accuse nodes by reading validity bits and assigning minority accusations, so cases (i) and (ii) are the only ones where accusations can be generated. □

***Lemma** 7: All obedient nodes agree on the consistent health vector calculated at each round k.*

***Proof**:* Let us assume by contradiction that consistency is violated and that two obedient nodes calculate two different $cons\_hv[j]$ entries for the same node j. Since H-maj is a deterministic function, the two nodes must have voted on different vectors $V = \langle al\_m[1][j], \ldots, al\_m[j-1][j], al\_m[j+1], [j], \ldots, al\_m[N][j]$. This implies that there exists at least one asymmetric faulty node $l \neq j$ such that value of $al\_m[l]$, which is taken from a diagnostic messages dm_l sent by node l, is received differently by the two obedient nodes. However, as $a \geq 1$ by assumption, the node j is either correct, or benign, or symmetric Byzantine. In all these cases all correct nodes set their validity bit for the message sent by j at round $k - 2u - 1$ to the same value by definition, and all obedient nodes receive the same value of the diagnostic message dm_j sent by j at round $k - 2u - 1$. From Lemma 6, the same value v is contained in $al\_ls_{k-u-1}[j]$ calculated by each obedient node, and thus from Lemma 5 also in $cons\_hv[j]$, a contradiction. □

***Lemma** 8: If an obedient node i is benign faulty at round $k - 2u - 1$, the $cons\_hv$ calculated by each obedient node at round k is such that $cons\_hv[i] = 0$.*

***Proof**:* If i suffer a send omission at round $k - 2u - 1$ then, by definition, the validity bit for the message sent by i to all obedient nodes is set to 0 by all obedient nodes in that round.

From Lemma 6, all obedient node set $al\_ls_{k-u-1}[i] = 0$ at round $k-u-1$. The result thus follows from Lemma 5. □

***Lemma** 9: If an obedient node i is member of a minority clique at round $k-3u-2$, the $cons\_hv$ calculated by all obedient nodes at round $k-u-1$ or at round k is such that $cons\_hv[i] = 0$.*

***Proof**:* If node i is benign faulty at round $k-3u-2$, the result follows from Lemma 8. Else, we only consider the case where there is no local error detection nor minority accusation at round $k-3u-2$ between i and a node j which is diagnosed as correct at round $k-3u-2$ by some obedient node l. The other case is in fact symmetric.

From Lemma 7, for all obedient nodes l, $diag_l$ calculates the same $cons\_hv$ at round $k-u-1$. By hypothesis and from Lemma 6, $diag_i$ computes $al\_ls_{k-2u-2}[j] = 0$ at round $k-2u-2$. From Lemma 3, $diag_i$ sends at round $k-2u-1$ a message dm_i such that $dm_i[j] = 0$. For every obedient node l, either there is a local error detection between i and l at round $k-2u-1$, or l receives dm_j, which is by definition different from $cons\_hv$. In both cases, it follows from Lemmas 6 and 5 that at round k all obedient node computes $cons\_hv[i] = 0$. □.

***Lemma** 10: For each obedient node i, if i is a member of the majority clique at rounds $k-3u-2$ and $k-2u-1$ then the consistent health vector calculated by each obedient node at round k is such that $cons\_hv[i] = 1$.*

***Proof**:* Let us assume by contradiction that i is a member of the majority clique at rounds $k-3u-2$ and $k-2u-1$ and the value of $cons\_hv[i]$ calculated by obedient nodes at round k is 0. From Lemma 5, if all obedient nodes set $al\_ls_{k-u-1}[i] = 1$ at round $k-u-1$, we would have a contradiction. Therefore at least one obedient node j has set $al\_ls_{k-u-1}[i] = 0$. From Lemma 6 and from the fact that i is not benign faulty at round $k-2u-1$, the diagnostic message dm_i sent by i to j at round $k-2u-1$ is different from the $cons\_hv$ calculated at round $k-u-1$. From Lemma 3, dm_i contains the value of al_ls_{k-2u-2} computed by i at round $k-2u-2$. From Lemma 6, the al_ls_{k-2u-2} vector is computed based on the messages received at round $k-3u-2$. The fact that al_ls_{k-2u-2} is different from the $cons\_hv$ calculated at round $k-u-1$ contradict the fact that i is in a majority clique.□

The correctness of clique detection and location is the basis for the activation of necessary view changes when the consistency of some members of the current view becomes too loose.

***Theorem** 3:* Tunable membership liveness *holds*.

***Proof**:* It follows from Lemma 7 and Alg. 3, lines 14 – 15 that the same view is provided by the protocol to all obedient nodes. Let us assume by contradiction that a node i of the current view has at round $k-3u-2$ a divergence degree $|div_i(k-3u-2, R-u-1)| \cdot criticalities[i] \geq 2P$ and that a view v' computed at round k by all obedient nodes is such that $i \in v'$. By definition, in each round $r \in div_i(k-3u-2, R-u-1)$, i was in a minority clique. The penalty value of i is incremented by $criticalities[i]$ at a round k' if $cons\_hv[i]$ is set to 0 by all obedient nodes in that round. From from Lemma 10 and Alg. 3, line 14, a penalty increment at k' for i implies that i is in a minority clique at to at most two rounds $k'-3u-2$ and $k'-2u-1$. This implies that i has got at least $|div_i(k-3u-2, R-u-1)| \cdot criticalities[i]/2 \geq P$ penalties. From Lemma 9, these increments are given during rounds in $[d_m+2u+1, d_M+3u+2]$, where d_m and d_M are $\min(div_i(k-3u-2, R-u-1))$ and $\max(div_i(k-3u-2, R-u-1))$ respectively. The p/r algorithm excludes nodes which have reached the penalty threshold P from the view $memb\_view$ (Alg. 2, line 7). In order not to be excluded from the p/r algorithm, node i must have reset the penalties at least once by accruing rewards in at least R consecutive rounds in $[r', r'+R-1]$, with $r' > d_m+2u+1$ and $r'+R-1 < d_M+3u+2$ (Alg. 2, line 12). The reward value of i is increased in a round if $cons\_hv[i] = 1$. From Lemma 9, if $cons\_hv[i] = 1$ at rounds $r-u-1$ and r then node i is in a majority clique at round $r-3u-2$. Since $cons\_hv[i] = 1$ for $R > u+1$ consecutive rounds, it thus follows that i is in a minority clique in rounds $[r'-2u-1, r'+R-3u-3]$, that is, for $R-u-1$ consecutive rounds. By definition of $div_i(k-3u-2, R-u-1)$, these rounds must either precede d_m or follow d_M. In the first case, $r'+R-3u-3 < d_m$, which contradicts the requirement $r' > d_m+2u+1$ as $R > u+1$. In the second case, $r'-2u-1 > d_M$, which contradicts the requirement $r'+R-1 < d_M+3u+2$ as $R > u+1$. □

Membership liveness could be trivially ensured by a protocol that always returns empty views. Tunable view synchrony rules out these solutions and requires that all nodes having a consistent internal state are included in every new view. Correctness of clique detection is its necessary precondition.

***Theorem** 4:* Tunable view synchrony *holds*.

***Proof**:* Let us assume by contradiction that an obedient node i with a divergence degree $|div_i(k', R+u+1)| \cdot criticalities[i] < \lceil P/2 \rceil$ for all $k' \leq k-2u-1$ is not included from the new view at round k. This can only happen if $cons\_hv[i]$ calculated at some round $r \leq k$ is equal to 0. In this case the penalty of i is incremented and can exceed the penalty threshold P. From Lemma 10, i was in a minority clique at round $r-3u-2$ or $r-2u-1$. This implies that divergence degree $div_i(k', R+u+1)$ after some round $k' \leq k-2u-1$ is not empty. However, $div_i(k', R+u+1)| \cdot criticalities[i] < \lceil P/2 \rceil$ by hypothesis. From Lemma 9, each of the elements in $div_i(k', R+u+1)$ corresponds to at most two penalty increments (at round $k'+3u+2$ or $k'+2u+1$). These result in $2 \cdot |div_i(k', R+u+1)| \cdot criticalities[i] < P$ penalties. If i has been excluded from the view and has reached the penalty threshold P, there must be a round r before or after those in $div_i(k', R+u+1)$ such that $cons\_hv[i] = 0$ at round r. We show that the reward threshold is reached, and the penalty and reward thresholds are reset, before the penalty is increased once again at round r.

Let $d_m = \min(div_i(k', R+u+1))$ and $d_M = \max(div_i(k', R+u+1))$. We first show that r cannot follow d_M. Let r_M be the smallest round $> d_M$ where i is in a minority clique. Note that since $r_M \notin div_i(k', R+u+1)$, $r_M - d_M > R+u+1 \geq u+2$ by definition of divergence degree and $R > 0$. From Lemma 10 and as i is in the majority clique at round d_M+1 and in the immediately following ones (which are at least $u+1$), $cons\_hv[i] = 1$ at round d_M+3u+3 or before. Therefore, i has its reward increased in that round and in the immediately following ones. As i is in a minority clique at round r_M, i has its penalty increased at round $r_M+2u+1 > d_M+R+3u+2$ or later from Lemma 10. In the meanwhile, node i has collected at least $(d_M+R+3u+3)-(d_M+3u+3) = R$ rewards. A similar argument is used to show that r cannot precede d_m. Let r_m be the largest round $< d_m$ where i is in a minority clique. Since $r_m \notin div_i(k', R+u+1)$, $d_m - r_m > R+u+1 \geq u+2$ by definition of divergence degree. As i is in the majority clique at round r_m+1 and in the immediately following rounds (which are at least $u+1$), its reward is increased at round $r_m+3u+3 < d_m-R+2u+2$ or before from Lemma 10. As i is in a minority clique at round d_m,

the penalty of i is again increased at round $d_m + 2u + 1$ or later from Lemma 9. Also in this case the reward threshold is reached before the penalty is increased again. □

VI. Formal verification of the protocols

We formally verified the correctness of the diagnostic and membership protocols presented in the previous sections by using model checking [9]. The hand-proofs of Sections IV and V show that the protocols satisfy their correctness properties. Model checking is an additional, independent technique to prove these correctness claims. We gave as input to the model checker the pseudo-code of our protocols, a description of our fault and system model, and the correctness properties of the protocol expressed in temporal logic. The full inputs for the model checker are in Appendix II. The model checker makes an exhaustive exploration of the generated state space and searches for a run (termed as counterexample) violating one of the properties. If no counterexample is returned, the property is proved true.

Model checkers require a finite representation of the distributed system whose size is small enough to permit exhaustive exploration within reasonable time and memory constraints. The properties of our protocol, however, are proved to hold for systems that are arbitrarily large in terms of the number of nodes. The size of the state space of each node can also grow arbitrarily large as the values of the penalty and reward thresholds P and R are unbounded. The goal of model checking is to verify the most relevant range of finite system behaviors. When computationally feasible, we verified systems with $N \leq 6$, $P \leq 3$, $R \leq 2$ and *criticalities*$[i] = 1$ for every node i. Such choice of parameters allowed the model checker to explore different combinations of faulty node behaviors such as multiple Byzantine faults (when $s = 2$ or $a = s = 1$), receive omissions ($a > 0$), transient faults ($P > 1$, $R > 1$), and partitions ($p > 0$). The overall system size N is realistic for many practical applications. We have also reduced the size of the models by considering frame-based systems where read and send alignment are not used and where $l_i = N$ and $send\_current\_round_i = false$ for each node i. Adding read and send alignment only adds some delays to the protocol, as shown by Lemmas 1 – 4 and 6.

The model checker. We used the Symbolic Analysis Laboratory (SAL) [12] for formal verification because of its powerful model checkers and expressive input language. In particular, we used the *bounded model checker* (BMC) of SAL as it performed better (in terms of execution time) than the symbolic one. The fact that BMC is only able to prove *invariants* was not a limitation, as the properties of both protocols define finite latencies and liveness properties that could be formalized as invariants by storing the requested values over rounds. Although BMC verification is *incomplete* in general [13], we proved all the correctness properties of our protocols.

Properties and verification setting. Table II summarizes the results of the model checking. The verification times ranged from a few minutes with $N = 4$ to a few hours with $N = 6$. The experiments were executed on an Intel Xeon 2GHz machine with 4GB memory running a SAL 3.0 installation on Fedora Core 6. The BMC engine of SAL used the Yices 1.0.3 satisfiability solver. For the diagnostic protocol, we focused on the properties of Correctness, Completeness and Consistency. The model checking verified that the result of Theorem 1 is correct under the considered parametric constraints. The property of Consistent Isolation is a corollary of Theorem 1, so we avoided its verification and set $P = 1$ to reduce the complexity of verification. For the membership protocol, the model checker verified the properties of Tunable Membership Liveness and Tunable View Synchrony and thus the correctness of Theorems 3 and 4. We had to restrict the verification of Theorem 3 to $N \leq 5$ to prevent the size of the model from becoming infeasible.

VII. Validation of the protocols

In this section we present the results of the experimental validation of the diagnostic and tunable membership protocols. We used physical fault injection to validate the correctness claims of Sections IV and V under the most common fault scenarios. We focused on validating the main correctness properties of the protocol, namely those regarding the consistent detection of faults, in the most important fault scenarios. A full-fledged validation of the protocol in every possible fault scenario is outside the scope of this work. We emphasize that all parameters used in the validation (and tuning, see Sec. VIII) arise from actual automotive and aerospace applications.

The validation setup consisted of a set of four nodes consisting of a host computer (Infineon Tricore 1796) and a communication controller (Xilinx Virtex 4 FPGA), which are interconnected via a redundant TT network with shared bus topology (layered TTP). Each host computer run a TT operating system. A diagnostic job run on each node as an add-on application-level module sending one diagnostic message (of 4 bits) 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 were automatically updated and the validity bits of a message m could be read using the API call `tt_Receiver_Status`. 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 wide range of faults in a node could be emulated by corrupting or dropping a message it sends. We observed the internal state of the nodes by using the debugging port of the hosts.

We injected different classes of network-level physical faults on the bus to simulate faults in a deterministic and reproducible manner. Application-level faults were reproduced by modifying the code of one diagnostic job. 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. A total of 1500 fault injection experiments was conducted. The fault injection experiments are summarized in Table III. The table reports the reproduced fault type, expressed in terms of the fault model, the level where the fault is injected, the duration of the injection, the number of instances of the experiment (in each instance different nodes suffer the fault) and the correctness claim which has been validated. Note that in the experiments the values of p/r parameters such as P and R were chosen to validate the correctness of our implementation prior to the system-specific tunings illustrated in the next Section VIII.

VIII. Practical tuning of the p/r algorithm

In the previous sections we have provided evidence for the correctness of the diagnostic and membership protocols by means of

TABLE II

SUMMARY OF THE ANALYSIS OF THE PROTOCOLS AND PROPERTIES WITH THE SAL BMC MODEL CHECKER

Protocol	Property	Parameters	Verified Claim	Result
Diagnosis	Consistency	$N \leq 6$, $P = 1$	Theorem 1	PROVED
	Correctness	$N \leq 6$, $P = 1$	Theorem 1	PROVED
	Completeness	$N \leq 6$, $P = 1$	Theorem 1	PROVED
Membership	Tunable Membership Liveness	$N \leq 5$, $P = 2$, $R = 2$	Theorem 3	PROVED
	Tunable View Synchrony	$N \leq 6$, $P = 3$, $R = 2$	Theorem 4	PROVED

TABLE III

FAULT INJECTION EXPERIMENTS — TDMA SYSTEM WITH $N = 4$ NODES, THRESHOLDS $P = 10$, $R = 20$ AND CRITICALITIES $= 1$

Fault type	Actual fault	Level of injection	Duration (rounds)	# Instances	Validated claim
$b = 1$	Bursty electrical spikes, random noise and silences during the sending slot of the faulty node	Network	1	4	*Diagnosis:* Theorem 1
$b = 2$	Same as the experiments above	Network	1	4	*Diagnosis:* Theorem 1
$b = 4$	Same as the experiments above	Network	2	4	*Diagnosis:* Theorem 1
$b = 1$	Random noise during the sending slot of the faulty node each second round	Network	20	1	*Diagnosis:* Theorems 1 and 2
$s = 1$	Random diagnostic message sent by the faulty node	Application	1	4	*Diagnosis:* Theorem 1
Receive omission ($a = 1$)	Bus disconnect between the faulty node and the other nodes during one sending slot	Network	1	1	*Diagnosis:* Theorem 1 *Membership:* Lemmas 9 and 10

hand proofs (Sections IV and V), formal verification (Section VI) and experimental validation (Section VII). These deterministic correctness properties are necessary but not sufficient for the protocol to correctly discriminate between healthy and unhealthy nodes. This requires a probabilistic tuning of the penalty and reward thresholds and of the criticality levels of each node, which entails doing many tradeoffs [30]. In this section, we describe how we have tuned our prototype in order to respect realistic automotive and aerospace requirements. The obtained tuning is summarized in Table IV.

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 [30]. In all our prototypes the probability of isolation of a healthy node is thus negligible.

Defining the tolerated outage. To increase availability and accumulate diagnostic data, the p/r algorithm delays the exclusion of nodes from the active set. In the period between fault manifestation and system recovery, applications may be prevented from correctly communicating and may suffer an outage. However, all applications used in our aerospace and automotive settings can tolerate bounded periods of continuous outage of a node before a recovery action is activated to restore the availability of the service or to reach a safe state. We define this upper bound on the recovery latency as *tolerated transient outage*. A node which alternates tolerably short periods of continuous faulty behavior with long enough periods of correct behavior can be kept in the set of active nodes.

Applications with different criticality classes have different requirements on the maximum tolerated transient outage. Tolerated transient outages for different classes of automotive and aerospace applications are shown in Table IV. 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)

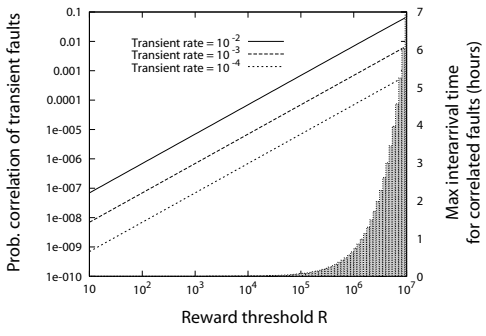

Fig. 3. Setting R with rounds of $2.5ms$

TABLE IV
RESULTS OF THE EXPERIMENTAL TUNING OF THE P/R ALGORITHM FOR DIAGNOSIS

Domain	Criticality class	Example	Tolerated outage	Crit. increment $inc(c_h)$	P	R	TDMA
Automotive	*Safety Critical (SC)* (c_1)	X-by-wire	$20-50ms$	40	197	10^6	$2.5ms$
	Safety Relevant (SR) (c_2)	Stability control	$100-200ms$	6			
	Non Safety Relevant (NSR) (c_3)	Door control	$500-1000ms$	1			
Aerospace	*Safety Critical (SC)* (c_4)	High Lift, Landing Gear	$50ms$	1	17	10^6	$2.5ms$

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. 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.

Tuning the tolerated outage. In order to tune the p/r algorithm according to the required tolerated outages we need first to identify a penalty threshold P and then to define penalty increments for each node, which are stored in the vector $criticalities$. The tolerated outage is the sum of the *detection delay*, between fault manifestation and its first consistent location as reflected in the consistent health vector, the *accumulation delay*, when the fault is continuously recorded by the p/r algorithm before the penalty threshold is reached, and the *recovery delay* required to complete the recovery or reconfiguration actions. In our prototype, the detection delays of both the diagnostic and the more complex membership service are low enough to satisfy the requirements of the highest criticality class considered. Once a faulty node is isolated, each obedient node can instantaneously apply the necessary reconfiguration with no recovery delay.

The tuning of the parameters P and $criticalities$ used by the p/r algorithm can increase the accumulation delay to maximize availability in presence of transient faults. However, our tuning must also ensure that whenever the outage of a node reaches the maximum tolerated outage of its most critical application, the penalty value for the node reaches the penalty threshold and the node is excluded. In order to find such tuning for the diagnostic protocol, we injected continuous faulty bursts into the network so that all nodes are benign faulty for an amount of time equal to the tolerated outage of the different criticality classes. After the burst is finished, we observed the value of the maximum value reached by the penalty counter of any node if all the entries of the vector $citicalities$ are set to 1. Each experiment was repeated 100 times. If classes $c_1, \ldots, c_n$ have corresponding maximum penalty counter $p_1, \ldots, p_n$, we set $P = max(p_1, \ldots, p_n)$. The criticality increment for class c_h, termed as $inc(c_h)$, is set to $inc(c_h) = \lceil P/p_h \rceil$ and is used to define the entries of the vector $criticalities$. If the set of criticality classes of all the applications hosted by node i is C_i, we set $criticalities[i] = \max_{c \in C_i} inc(c)$.

The penalty thresholds and criticality levels for the automotive and aerospace setups are shown in Table IV. 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.

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 V. We reproduced these scenarios in 100 experiments and observed if and after how much time healthy nodes were incorrectly isolated.

TABLE V
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

TABLE VI
TIME TO INCORRECT ISOLATION

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

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 VI. 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 experimental settings. From this data we 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 [30].

IX. Portability Issues for Varied TT Platforms

One of our main design goals 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 APIs provided by the host operating system (see Sec. VII).

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 checking the availability of updated diagnostic messages at the application level. This information is provided by all TT platforms. 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. IV). 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. Note that in case of dynamic scheduling the constant u must be set to 1 because it is impossible to evaluate the global condition needed to set it to 0 and thus to reduce latency.

X. Conclusions

Emerging TT platforms, such as FlexRay, need diagnostic and membership algorithms that are portable, generic, and resistant to the widest possible range of faults. In this paper, we have introduced novel protocols which fulfill these requirements and which can be added-on as a middleware layer on top of any TT platform. Our protocols aim at relaxing the fault assumptions of existing protocols. They can tolerate Byzantine faults and multiple benign faults, and can improve availability even in presence of transient faults by using a p/r algorithm. The main costs for this sophisticated fault handling compared to existing protocols for TT systems are a slightly higher latency and higher bandwidth costs for on-line reintegration. The correctness of the protocols is comprehensively substantiated. We have proved the protocol correctness by hand and through formal verification. Furthermore, the protocol has also been experimentally validated under the most common fault scenarios. Finally, we have discussed common tradeoffs which arise when handling transient faults. We have shown how to tune the p/r algorithm under realistic automotive and aerospace settings, and addressed open issues of characterization of intermittent faults, determination of the criticality of faults and diagnosis under adverse external conditions.

References

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

[2] M. Barborak *et al.*, "The Consensus Problem in Fault Tolerant Computing," *ACM Surveys*, vol. 25, no. 2, pp. 171–220, Jun. 1993.

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

[4] C. Basile *et al.*, "Group Communication Protocols under Errors," *Proc. SRDS*, pp. 35–44, 2003.

[5] C. Bergenhem and J .Karlsson, "A Process Group Membership Service for Active Safety Systems Using TT/ET Communication Scheduling," *Proc. PRDC*, pp. 282–289, 2007.

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

[7] A. Bondavalli *et al.*, "Threshold-Based Mechanisms to Discriminate Transient from Intermittent Faults," *IEEE Trans. on Computers*, vol. 49, no. 3, pp. 230–245, Mar. 2000.

[8] A. Bouajjani, and A. Merceron, "Parametric Verification of a Group Membership Algorithm," *Theory Pract. Log. Program.*, vol. 6, no. 3, pp. 321–353, May 2006.

[9] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, MIT Press, 2000.

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

[11] F. Cristian, "Reaching Agreement on Processor-group Membership in Synchronous Distributed Systems," *Distributed Computing,* vol. 4, no. 4, pp. 175–187, Dec. 1991.

[12] L. de Moura *et al.*, "SAL 2," *Proc. CAV*, pp. 496–500, 2004.

[13] L. de Moura *et al.*, "Bounded Model Checking and Induction: From Refutation to Verification," *Proc. CAV*, pp. 14–26, 2003.

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

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

[16] B. Hall, K. Driscoll, M. Paulitsch and S. Dajani-Brown, "Ringing out fault tolerance. A New Ring Network for Superior Low-Cost Dependability," *Proc. DSN*, pp. 298–307, 2005.

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

[18] K. Hoyme and K. Driscoll, "SAFEbus," *IEEE Aerospace and Electronic Systems Magazine,* vol. 8, no. 3, pp. 34-39, Mar. 1993.

[19] H. Kopetz *et al.*, "The Time-Triggered Ethernet (TTE) Design," *Proc. ISORC,* pp. 22–33, 2005.

[20] H. Kopetz And G. Bauer, "The Time-Triggered Architecture," *Proc. IEEE,* vol. 91, n. 1, pp. 112–126, 2003.

[21] H. Kopetz and G. Grunsteidl, "TTP - A Protocol for Fault Tolerant Real Time Systems," *IEEE Computer,* vol. 27, no. 1, pp. 14–23, Jan. 1994.

[22] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem," *ACM Trans. on Progr. Lang. and Sys.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

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

[24] N. A. Lynch, "*Distributed Algorithms*," Morgan Kaufmann Ed., 1996.

[25] M. Malek, "A Comparison Connection Assignment for Diagnosis of Multiprocessor Systems," *Proc. ASCA*, pp. 31–36, 1980.

[26] H. Pfeifer, "Formal Verification of the TTP Group Membership Algorithm," *Proc. FORTE XIII / PSTV XX 2000*, pp. 3–18, 2000.

[27] F.P. Preparata *et al.*, "On the Connection Assignment Problem of Diagnosable Systems," *IEEE Trans. on Electronic Computers*, vol. 16, no. 12, pp. 848-854, Dec. 1967.

[28] J. Rushby, " Systematic formal verification for fault-tolerant time-triggered algorithms," *IEEE Trans. on Software Eng.*, vol. 25, no. 5, pp. 651–660, Sept. 1999.

[29] S. Katz *et al.*, "Low-Overhed Time-Triggered Group Membership," *Proc. WDAG*, pp. 155–169, 1997.

[30] M. Serafini *et al.*, "Online Diagnosis and Recovery: On the Choice and Impact of Tuning Parameters," *IEEE Trans. on Dependable and Secure Computing*, vol. 4, no. 4, Oct. 2007.

[31] H. Sivecrona *et al.*, "Protocol Membership Agreement in Distributed Communication Systems - A Question of Brittleness," *Proc. SAE*, paper 0108, 2003.

[32] W. Steiner *et al.*, "The TTA's Approach to Resilience after Transient Upsets," *Real-Time Systems*, vol. 32, no. 3, pp. 213–233, 2006.

[33] P. Thambidurai and Y. Park, "Interactive consistency with multiple failure modes," *Proc. SRDS*, pp. 93–100, 1988.

[34] C. Walter *et al.*, "Formally Verified On-line Diagnosis," *IEEE Trans. on Software Eng.*, vol. 23, no. 11, pp. 684–721, Nov. 1997.

APPENDIX I
PARTITIONABLE MEMBERSHIP

Network partitioning represents a typical class of faults that are difficult to prevent, like for example in automotive systems that use basic broadcast-based networks with limited channel redundancy and are installed in a harsh environment. Partitions can be either temporary, due for example to electro-magnetic interferences on the network, or permanent, as in case of a network cut. This results in splitting the system into cliques of nodes that can only communicate within their clique. The membership protocol of Section V detects and locates cliques that are generated by at most *one* asymmetric fault. During a partition, *each* message sent is asymmetric faulty according to the fault model because it only reaches a subset of the nodes.

The diagnostic and membership protocols require that a unique view of the active nodes must be agreed upon by all obedient nodes. This cannot be ensured during partitions because the members of each clique see the other nodes as faulty and agree on different "consistent" health vectors. The resulting divergences on the penalty and reward counters can eventually lead to the creation of different views if a penalty value associated to a node exceeds its threshold on only one clique. In order to avoid that, we designed a partitionable membership protocol which preserves consistency in presence of partitions.

A fundamental problem with partitions is that it is impossible for a node i to distinguish the case where multiple other nodes are benign faulty and i should form a new view from the case where i is partitioned out and it should isolate itself from the view because it has inconsistent local counters. As a decision cannot be postponed indefinitely and we do not want to make assumptions on the duration of the partitions, our protocol needs to assume that not more than half of the nodes are benign faulty in order to take a timely decision. If the assumption is wrong, the system becomes unavailable but not inconsistent.

Fault Model. The partitionable membership protocol has been designed to consider one additional failure model for nodes that are neither benign nor symmetric nor asymmetric faulty. We say that a view is *partitioned* when multiple cliques of otherwise correct nodes exist, called partitions, such that there is always a local error detection between the correct nodes of two different cliques and no local error detection within the clique. We distinguish between a *primary* and possibly multiple *secondary* partitions, such that the first one includes at least a majority of nodes. Partitioning can begin and, if transient, terminate during any sending slot. In the rounds where partitioning begins and ends, some members of a clique may be thus able to correctly communicate with all members of some other clique(s). Partitions are different from asymmetric faults because they result in at least two messages which are only received by the same clique of the sender. A common example of partition is when the network link of a node is broken and the node cannot receive any message from the other nodes. We assume that the number of members of all secondary partitions is at most p, that the number of nodes in the system is $N > 2a + 2s + 2p + 2b + 1$, $a \leq 1$.

View agreement and view majority. Partitionable membership prevents possible divergences in the membership view by carrying out a *view agreement* phase before establishing new views. At each round, after observing that a node has exceeded its penalty threshold, a new view is computed exactly like $memb\_view$ in Alg. 3. However, different from the previous algorithm, this is called *local membership view* ($local\_mv$) and it is not given as output but it is broadcasted for view agreement. After dissemination and aggregation, a *consistent membership view* ($cons\_mv$) is voted and returned as output. The hybrid majority voting function of Eqn. 1, cannot prevent the formation of different views because it only considers votes that have been correctly received by a node, and ignores omitted messages. This can lead to the creation of different views as two different local majorities can be formed in each clique which result in different views. Therefore we use a *simple majority voting* function, which calculates the majority among the elements of a vector V. This ensures that only one view is created at each round if Byzantine and partitioned-out nodes represent a minority. A default value is output if no value reaches a majority.

Algorithm 4: Node i partitionable membership job $part\_memb_i$ at round k

```
// Local detection, aggregation, analysis,
// minority accusation and dissemination
same as Alg. 3, lines 2 - ??
// Update counters
1  local_mv_k ← p_r(cons_hv, local_mv_k);
// Dissemination and aggregation of local views
2  if (∀j : send_curr_round_j) ∨ (¬ send_curr_round_i) then
3      write_iface_field2(local_mv_k);
4  else write_iface_field2(local_mv_{k-1});
5  mv_k ← read_iface_field2(dm_1, ..., dm_N);
6  for j ← 1, ..., l_i do al_mv[j] ← mv_{k-1}[j];
7  for j ← l_i + 1, ..., N do al_mv[j] ← mv_k[j];
// View agreement
8  for j ← 1, ..., N do
9      res ← maj⟨al_mv[1][j], ..., al_mv[N][j]⟩;
10 if ∀l : res[l] ≠ ⊥ then cons_mv[j] ← res;
11 else self-isolate;
12 return cons_mv;
```

The pseudo code of the partitionable membership algorithm for node i is shown in Alg. 4. The only modification with respect to Alg. 3 is the handling of the view change process. After updating the counters, the algorithm does not immediately change the view but it builds a new local membership view $local\_mv_k$, which will then be proposed during view agreement. Compared to the prior algorithms, the diagnostic message contains one additional field of N bits which contains the local view (Alg. 4, line 1). Similar to local syndromes, local membership views are disseminated and aggregated using send and read alignment (Alg. 4, lines 2 - 7).

View agreement is then performed on the aligned local membership views to obtain the consistent membership view (Alg. 4, lines 8 - 11). If no value can get a majority, nodes must be partitioned out. In this case nodes do not create a new view but rather isolate themselves. The special operation *self-isolate* suspends the execution of the protocol and signals to the applications that the node has been excluded from the view.

Properties of partitionable membership. The properties of partitionable membership can be proved in a manner similar to the membership protocol, so we omit similar details in the proofs. Majority and minority cliques are now established by the members of the primary partition, and the local membership view is agreed using the same operations as Alg. 3. The local membership view has thus the same properties as in Alg. 3, but only for the nodes in the primary partition.

***Lemma** 11:* Tunable membership liveness *and* Tunable view synchrony *hold for the local membership views calculated by all obedient nodes of the primary partition.*

***Proof**:* We prove that nodes in the primary partitions can not distinguish a run where a partition takes place from a run where certain tolerated fault scenarios arise. The result thus follows from Theorems 3 and 4, as Alg. 3 and 4 calculate the local membership view exactly in the same way.

From the viewpoint of nodes in the primary partitions, runs with partitions differ from runs without partitions by two facts: (i) during the partition, all otherwise correct nodes in the secondary partitions appear as benign faulty, and (ii) after the partition, some otherwise correct nodes in the secondary partitions appear as symmetric faulty. Case (ii) is given by the fact that nodes in the secondary partition can have a state which appears as incorrect to the nodes in the primary partition, and thus send messages which are apparently arbitrary. To give an example, nodes in the secondary partition may accuse correct nodes in the primary partition just because they have been partitioned out.

From the fault model, there are at most p obedient nodes in the secondary partitions. Let p_b (resp. p_s) be the number of obedient nodes in the secondary partition which only appear as benign faulty (resp. symmetric faulty) to the nodes in the primary partition. By definition, $p \geq p_b + p_s$. Furthermore, let $b' = b + p_b$ (resp. $s' = s + p_s$) be the number of nodes which appear as benign (resp. symmetric) faulty to the nodes in the primary partition. The fault assumption of Theorems 3 and 4 require that $N > 2a + 2s' + b' + 1$ and $a \leq 1$. This holds under the fault assumption of partitionable membership, that is, $N > 2a+2s+2p+2b+1$ and $a \leq 1$. Therefore, the two theorems hold for the nodes in the primary partition. □

***Lemma** 12: If nodes i and j are obedient, the value of $al\_mv[j] \neq \varepsilon$ computed by $part\_memb_i$ at round k contains $local\_mv_{k-u-1}$ computed by $part\_memb_j$ at round $k-u-1$.*

***Proof**:* In Alg. 4, diagnostic messages not only contain the local syndromes but also the local membership views. As read and send alignment in Alg. 4 is executed exactly like in the previous protocols, Lemma 4 holds not only for al_ls but also for the local membership views $local\_mv$. This implies that node i executes the majority voting of Alg. 4, line 9 at round k over vectors al_mv which contain the local membership views $local\_mv_{k-u-1}$ computed by $part\_memb_j$ at round $k-u-1$. □

***Theorem** 5:* Tunable membership liveness *and* tunable view synchrony *hold with an additional delay of $u+1$ rounds.*

***Proof**:* From Lemma 11, all obedient nodes of the primary partition agree on the local membership view at each round. We prove that at each round k each obedient node either executes self-isolate or computes as consistent membership view the local membership view $local\_mv_{k-u-1} = v$ computed by the nodes of the primary partition at round $k-u-1$. Let us assume that some node i exists which returns a consistent membership view v' such that $v'[l] \neq v[l]$ at round k for some l (Alg. 4, line 12). This implies that the vector $\langle al\_mv[1][l], \ldots, al\_mv[N][l] \rangle$ contains more than $N/2$ elements equal to $v'[l]$ (Alg. 4, lines 8 and 11). It follows from the fault assumption that there exists at least one correct node j in the primary partition which has sent a diagnostic message dm_j to i such that $dm_j = al\_mv[j] = v'[j] \neq v[j]$ is computed by $part\_memb_i$ at round k. In fact, the number of nodes which are faulty or which are members of secondary partitions, $a+s+p+b$, is smaller than $N/2$. From Lemma 12, the value $al\_mv[j]$ computed by $part\_memb_i$ at round k is $local\_mv_{k-u-1}$ computed by $part\_memb_j$ at round $k-u-1$. By definition, $al\_mv[j] = v[j]$, a contradiction. □

APPENDIX II
SAL MODELS OF THE DIAGNOSIS AND MEMBERSHIP PROTOCOLS

We first explain the SAL model of the diagnosis protocol which is depicted in Table IX. This model will be the base for the models of the tunable and partitionable membership protocols. Every SAL model was created in full accordance with the pseudo code of the protocol. Additional variables are required, as we will see, to model the environment such as faults or divergence degrees. A short summary of the state variables in the SAL models are shown in Table VII.

Every SAL model is divided into three main parts which are the type and constant declaration part, the module definition part, and the property definition part. These parts can be found in Table IX between lines 2–4, 5–57, and 59–61 respectively. For the sake of clarity, the types and constant declarations for all SAL models are separately listed in Table VIII. The SAL model of diagnosis is parameterized in the number of nodes N (at line 1 in Table IX). SAL modules define state variables and so called transitions to update these variables. Each of our SAL models contains a single module called `system` (line 5) which replicates the variables used in the protocols' pseudo code for each node in the system. This is done by defining an array which is indexed by the ID of the node. For example, `cons_hv[1]` stores the consistent health vector at node 1. One round of the system is modeled through an auxiliary state variable called `pc` (defined at line 11) which ranges from 0 to 3 such that the system's state at the end of the round is reflected by the state variables when `pc` is 0. The ascending values of `pc` correspond to variable updates that are dependent of each other. Now we first explain how the protocol's operations are modeled at every round, then the properties of the protocol are specified.

Diagnosis: Protocol Operations. The state variables are first declared (lines 7–14) and initialized (lines 15–22). The sequence of the SAL transitions that model the execution of the protocol is fully deterministic. These are defined by guarded transitions when `pc` is 0, 1, 2 and 3 at the lines between 25–31, 33–39, 41–47 and 49–59 respectively. Each of these guarded transitions increments the value of `pc` modulo 4. The convention of updating variables in SAL is that a primed (respectively unprimed) version of the same variable denotes the new (old) value of it. In SAL, multiple updates defined within the same transition are executed in parallel, i.e., the sequence of their definition in the code is insignificant.

At the beginning of the round, when `pc=0`, the faults of the current round are generated (lines 26–29) which are stored in the variable `fvec`. For example, `fvec[1]=byz` says the node 1 is Byzantine in the current round. We use SAL's non-deterministic assignment for assigning a value to `fvec`. The syntax `var' IN {v:type|condition}` means that the model checker explores multiple runs such that each different value from the domain `type` where the Boolean `condition` holds is assigned to `var`. In this case, the Boolean condition is determined by the fault model. The pre-defined function constant `fcounter` is used to count the number of faults of each severity class. This function takes the previous and current values of `fvec` as inputs to make

TABLE VII

DESCRIPTION OF THE MAIN STATE VARIABLES IN THE SAL MODELS OF THREE PROTOCOLS

Variable	Description
`fvec[i]`	fault status of node i in the current round
`ls[i][j]`	local error detection between node j and i in the current round
`al_m[i][j]`	diagnostic message received by node i from node j in the current round
`cons_hv[i][j]`	consistent health vector for node j computed by node i in the current round
`penalties[i][j]`	penalty counter for node j stored by node i
`rewards[i][j]`	reward counter for node j stored by node i
`dd_prev[i]`	divergence degree of node i with recovery latency $R+1$ in the previous round
`dd2_prev[i]`	divergence degree of node i with recovery latency $R-1$ in the previous round
`R_dd_prev[i]`	number of the last consecutive rounds $<$ current round where node i was in the majority clique
`al_mv[i][j]`	local membership view received by node i from node j in the current round
`cons_mv[i][j]`	consistent membership for node j computed by i in the current round

TABLE VIII

TYPE AND CONSTANT DECLARATIONS IN THE SAL MODELS OF THE THREE PROTOCOLS

```
nodes: TYPE = [1..N];
fdomain: TYPE = {symm, corr, ben, byz};
valdomain: TYPE =[0..2];
vecdomain: TYPE = ARRAY nodes OF valdomain;
pcdomain: TYPE = [0..3];
int: TYPE = [0..N];
matdomain: TYPE = ARRAY nodes OF vecdomain;

fcounter(v: ARRAY nodes OF fdomain, vprev: ARRAY nodes OF fdomain, f:fdomain,idx:int,cnt:int): int= LET aggr_fvec:
ARRAY nodes OF fdomain = [[n:nodes]
    IF v[n]=byz OR vprev[n]=byz THEN byz
    ELSE IF v[n]=symm OR vprev[n]=symm THEN symm
    ELSE IF v[n]=ben OR vprev[n]=ben THEN ben
    ELSE corr ENDIF ENDIF ENDIF]
IN IF idx = 0 THEN cnt ELSE
    IF aggr_fvec[idx]=f THEN fcounter(v,vprev,f, idx-1, cnt+1) ELSE fcounter(v,vprev,f,idx-1,cnt) ENDIF ENDIF;

pcounter(v: ARRAY nodes OF fdomain, vprev: ARRAY nodes OF fdomain, vprev2: ARRAY nodes OF fdomain, vprev3: ARRAY nodes
OF fdomain, k:[0..N], kprev:[0..N], kprev2:[0..N], kprev3:[0..N], idx:int, cnt:int):int=
    IF idx = 0 THEN cnt ELSE
        IF v[idx]=corr AND vprev[idx]=corr AND vprev2[idx]=corr AND vprev3[idx]=corr AND
            (k=idx OR kprev=idx OR kprev2=idx OR kprev3=idx)
        THEN pcounter(v,vprev,vprev2,vprev3,k,kprev,kprev2,kprev3,idx-1,cnt+1)
        ELSE pcounter(v,vprev,vprev2,vprev3,k,kprev,kprev2,kprev3,idx-1,cnt) ENDIF ENDIF;

h_maj_aux(sv: vecdomain, i: nodes, idx: INTEGER, sum0: INTEGER, sum1: INTEGER): valdomain =
    IF idx = 0 THEN IF sum0 > sum1 THEN 0 ELSE (IF sum1 > sum0 THEN 1 ELSE
        (IF sum1=0 AND sum0=0 THEN 2 ELSE 1 ENDIF) ENDIF) ENDIF
    ELSE h_maj_aux(sv,i,idx-1, sum0 + IF (sv[idx]=0 AND idx /= i) THEN 1 ELSE 0 ENDIF,
                               sum1 + IF (sv[idx]=1 AND idx /=i) THEN 1 ELSE 0 ENDIF)ENDIF;

h_maj(sm: ARRAY nodes OF vecdomain): vecdomain = [[i:nodes] h_maj_aux([[n:nodes] sm[n][i]], i, N, 0, 0)];

maj_aux(sv: vecdomain, idx: INTEGER, sum0: INTEGER, sum1: INTEGER): valdomain =
    IF idx = 0 THEN IF sum0 > N DIV 2 THEN 0 ELSE (IF sum1 > N DIV 2 THEN 1 ELSE 2 ENDIF) ENDIF
    ELSE maj_aux(sv, idx-1, sum0 + IF sv[idx]=0 THEN 1 ELSE 0 ENDIF,
                            sum1 + IF sv[idx]=1 THEN 1 ELSE 0 ENDIF) ENDIF;

maj(sm: ARRAY nodes OF vecdomain): vecdomain = [[i:nodes] maj_aux([[n:nodes] sm[n][i]], N, 0, 0)];

is_prim_part(n:nodes,k:nodes,k_prev:nodes,k_prev2:nodes,k_prev3:nodes):BOOLEAN =
    IF n/=k AND n/=k_prev AND n/=k_prev2 AND n/=k_prev3 THEN true ELSE false ENDIF;
```

sure that the fault model holds over one instance of the diagnosis protocol, which lasts two rounds. The previous value of `fvec` is stored (line 30) and used in the specification of the properties.

When `p=1`, the local syndromes are computed. The Boolean condition of the non-deterministic assignment constrains that correct and symmetric Byzantine faulty senders cannot be locally detected by the receiver node (line 35), a benign faulty sender is always detected (line 36), and that the local syndrome of obedient nodes always assume boolean values (line 37). The local syndromes of the current round are stored (line 38) in order to model the content of the diagnostic messages received in the next round.

As a next step (`p=2`), the diagnostic matrix `al_m` can be computed (line 42). This transition is based on the values of the local syndromes which were computed in the previous round and sent as diagnostic messages. The Boolean condition of the non-deterministic assignment defines that an obedient receiver always correctly receives the local syndromes of a correct sender (line 43), the receiver delivers a vector of ε values (denoted with 2) if the sender is locally detected as faulty (line 44), receive omissions

TABLE IX
SAL CODE OF THE DIAGNOSIS PROTOCOL

```
1  diagnosis{;N: natural}: CONTEXT = BEGIN
2
3  %types and constant declaration comes here
4
5  system: MODULE =
6      BEGIN
7        LOCAL
8      ls: ARRAY nodes OF vecdomain,
9      al_m: ARRAY nodes OF matdomain,
10     cons_hv:ARRAY nodes OF vecdomain,
11     pc: pcdomain,
12     fvec: ARRAY nodes OF fdomain,
13     fvec_prev: ARRAY nodes OF fdomain,
14     ls_prev: ARRAY nodes OF vecdomain
15       INITIALIZATION
16     ls = [[n:nodes] [[m:nodes] 1]];
17     ls_prev = [[n:nodes] [[m:nodes] 1]];
18     al_m = [[n:nodes] [[m:nodes] [[l:nodes] 1]]];
19     cons_hv = [[n:nodes] [[m:nodes] 1]];
20     pc = 0;
21     fvec = [[n:nodes] corr];
22     fvec_prev = [[n:nodes] corr];
23       TRANSITION
24       [
25     pc=0 -->
26         fvec' IN {v:ARRAY nodes OF fdomain|
27             fcounter(v,fvec,byz,N,0)<=1 AND
28             (fcounter(v,fvec,symm,N,0)+fcounter(v,fvec,byz,N,0) > 0
29                 => N > 2*fcounter(v,fvec,symm,N,0) + 2*fcounter(v,fvec,byz,N,0) + fcounter(v,fvec,ben,N,0) + 1)};
30         fvec_prev'=fvec;
31         pc' = 1;
32       []
33     pc=1 -->
34         ls' IN {v:ARRAY nodes OF vecdomain|FORALL(m,n:nodes):
35             (fvec[n]=corr OR fvec[n]=symm => v[m][n]=1) AND
36             (fvec[n]=ben => v[m][n]=0) AND
37             v[m][n]/=2};
38         ls_prev'=ls;
39         pc' = 2;
40       []
41     pc=2 -->
42         al_m' IN {v:ARRAY nodes OF matdomain|FORALL(m,n:nodes):
43             (fvec[n]=corr => v[m][n]=ls_prev[n]) AND
44             (ls[m][n]=0 <=> v[m][n]=[[l:nodes]2]) AND
45             ((EXISTS(l:nodes):v[m][n][l]=2)=>v[m][n]=[[l:nodes]2]) AND
46             (fvec[n]=symm => FORALL(l,k:nodes):v[m][n][l]=v[k][n][l])};
47         pc' = 3;
48       []
49     pc=3 -->
50         cons_hv'=[[n:nodes]
51             IF EXISTS(j:nodes):h_maj(al_m[n])[j]=2 THEN ls_prev[n]
52             ELSE h_maj(al_m[n]) ENDIF];
53         pc' = 0;
54       []
55           ELSE -->
56       ]
57     END;
58
59   consistency: THEOREM system |- G(FORALL(l,m,n:nodes):pc=0 => cons_hv[l][m]=cons_hv[n][m]);
60   completeness: THEOREM system |- G(FORALL(n:nodes):fvec_prev[n]=ben AND pc=0 => FORALL(m:nodes):cons_hv[m][n]=0);
61   correctness: THEOREM system |- G(FORALL(n:nodes):fvec_prev[n]=corr AND pc=0 => FORALL(m:nodes):cons_hv[m][n]=1);
62 END
```

always invalidate the whole diagnostic message so that all entries assume the value ε (line 45), and symmetric Byzantine nodes send the same syndrome to every node in the system (line 46).

The last step of the round (`p=3`) is the computation of the consistent health vector (line 50), which is deterministic, and uses the outcome of the hybrid majority unless the voting function cannot reach a decision and outputs $\perp$ (lines 51–52). In the latter case, which is modeled via the voting function returning 2, the local syndrome of the previous round is used. After that, `pc` is reset to 0 and the modeling of the new round follows similarly.

Diagnosis: Protocol Properties. The properties of the diagnosis protocol are specified as invariants. The syntax `G(condition)` specifies that `condition` which is a Boolean formula over the values of the state variables is an invariant, i.e., it holds in each reachable state of the system. The properties are specified such that they must only hold at the end of each round, i.e., when `pc=0`. The property `consistency` specifies that the consistent health vector entry for node m is the same for every pair of nodes l and n (line 59). The reason why we do not distinguish between obedient and non-obedient nodes is

that a faulty node is modeled via the messages it sends but its internal state is modeled as it was an obedient one. Therefore, the properties must hold for them as well. Completeness is specified via the predicate `completeness`, which specifies that if a node n is benign faulty in the previous round it must be diagnosed as faulty in the consistent health vector of all obedient nodes m (line 60). Correctness follows from the predicate `correctness`, which states that a correct node n will not be diagnosed at the next round as faulty by any other obedient node m, by simple induction over subsequent protocol instances.

Tunable Membership: Protocol Operations. The SAL model of the tunable membership protocol is shown in Tables X and XI. In addition to N, this model has the penalty and reward thresholds P and R as parameters (line 1). The transitions guarded by `pc=0`, `pc=1`, `pc=2` and `pc=3` are almost identical with those of the diagnosis protocol. The only difference is that `fcounter` now takes as inputs the fault vectors of the current and the last *two* rounds because one execution of tunable membership lasts three rounds.[1] Furthermore, the boolean variables `obedient` store for each node if it has never been Byzantine or symmetric Byzantine (line 44).

In this model a new transition is added (lines 61–103) which models minority accusation to a node n (lines 62–63), the update of counters (lines 65–74) and divergence degrees (lines 76–101). A node is active (respectively inactive) if `penalties` $< P$ (`penalties=P`).

In order to express the properties of tunable membership liveness and tunable view synchrony in a more compact manner, the model maintains two variables `dd_prev` (line 84) and `dd2_prev` (line 76) which are the divergence degrees of each node at the previous round with recovery latency $R+1$ and $R-1$ respectively. These are needed for the synchrony and liveness properties of tunable membership. As the domain of each state variable needs to be finite in model checking, we defined the ranges of `dd_prev` and `dd_prev2` large enough such that the properties of the protocol could be specified. The divergence degrees are only incremented if their values stay within the domain (lines 85 and 77). This is a trivial abstraction which preserves all properties of the protocol as the behavior of the protocol is not specified for the values of the divergence degrees that are not explicitly modeled. A new variable is needed to count the number of consecutive rounds where a node is in a majority clique (`R_dd_prev`, line 91). As the model in Table XI assumes that $R = 2$, `dd2_prev` is reset if a node is in the majority clique of a round (line 80).

Since the BMC model checker does not support the definition of temporal properties, the model has to remember for synchrony that a node's divergence degree has never reached $\lceil P/2 \rceil$. This is done by `dd_synchr_prev[i]` (line 95) which is true at round k if and only if `dd_prev[i]` had not reached $\lceil P/2 \rceil$ at round $(k-1)$ or before. As the new value of `dd_prev` is only available in the next step when `pc=0`, we check that $\lceil P/2 \rceil$ was not reached at round $(k-1)$ by using the conditions of being in the majority clique (lines 98–99). Such a modeling was needed to minimize the number of transitions in our SAL models because this can significantly affect the complexity of BMC model checking.

Tunable Membership: Protocol Properties. The properties of tunable membership are specified as `synchrony` (line 110) and `liveness` (line 111). The first property mandates that an obedient node n with a low divergence degree should never be isolated by an obedient node m, that is, its penalty must always be smaller than P. The second property mandates isolations if the divergence degree of n is too large. We require these properties to hold only if the node is marked as obedient. The divergence degree for Byzantine nodes is not defined as their local syndrome is unconstrained.

Partitionable Membership: Protocol Operations. The SAL model of partitionable membership is shown in Tables XII and XIII. For the feasibility of model checking, the models assume that $N \leq 4$. For all other parameters we use the same values as for the tunable membership protocol. The first transition is now a parameterized by `k_curr` and `ps` (line 37). This macro means that SAL generates one instance of the transition for every pair of possible values of these two parameters. Each of these transitions selects for each round at most one partitioned node with ID `k_curr` and the number of the first slot `ps` when the system is partitioned. The system is not partitioned in the current round if the first parameter is 0. Note that the assumption $N \leq 4$ implies that at most one node is partitioned out during each execution of the protocol. The assignment of `fvec` (line 39) is adjusted to the fault model of partitionable membership. The function `pcounter` (see Table VIII) is used to count the number of partitioned correct nodes during one protocol execution (line 42). Nodes that self-isolate will act as benign faults in the future (line 43).

The local syndromes are now computed such that partitions are also considered (line 55). The assignment of `ls` is the same as for tunable membership if either there is no partition (line 56), the sender and the receiver are in the primary partition (line 57), the sender is the receiver (line 58) or the partition has not started yet (line 59). Otherwise, the sender and the receiver are in different partitions and the receiver locally detects the sender as faulty (line 62). The assignment of the diagnostic matrix defines that a partitioned node, even if otherwise correct, cannot send its local syndrome to a receiver in another partition (line 72). A new variable `al_mv` has been added to store the local membership views that are collected in the last round of the protocol to reach agreement about the set of active nodes. The assignment of this variable (line 75) only differs from `al_m` in that the local membership view instead of the local syndrome is sent by each node (line 77). Simple majority voting is used to compute the global membership view. This is done by the next transition and the result is stored in `cons_mv` (line 85). The definition of the majority voting function can be found in Table VIII.

The penalty and reward counters and the divergence degrees are updated similar to tunable membership. The only difference is the minority and majority cliques are established based on the primary partition. This means that the local syndrome of a node will be compared with that of an active node in the primary partition (see lines 103,111,117 and 125). A simple macro called `is_prim_part` defines a node as part of the primary partition if it was never part of the secondary partition during the last execution of the protocol. The array `inactive` keeps track of the nodes that executed self-isolation when they registered that no agreement over the global membership view can be reached (line 130).

Partitionable Membership: Protocol Properties. The synchrony and liveness properties of the partitionable membership protocol are specified as `psynchrony` and `pliveness` (lines

[1]Table VIII only contains the definition of `fcounter` and `pc` for the diagnosis protocol.

TABLE X
SAL CODE OF THE TUNABLE MEMBERSHIP PROTOCOL (PART 1)

```
membership{;N: natural,P:natural,R:natural}: CONTEXT = BEGIN

%types and constant declaration comes here

system: MODULE =
    BEGIN
      LOCAL
    %only variables new to tunable membership are listed here
    obedient: ARRAY nodes OF BOOLEAN,
    penalties: ARRAY nodes OF ARRAY nodes OF [0..P],
    rewards: ARRAY nodes OF ARRAY nodes OF [0..R-1],
    dd_prev: ARRAY nodes OF [0..P],
    dd2_prev: ARRAY nodes OF [0..2*P],
    dd2_prev2: ARRAY nodes OF [0..2*P],
    R_dd_prev: ARRAY nodes OF [0..R],
    dd_synchr_prev: ARRAY nodes OF BOOLEAN
      INITIALIZATION
    %only variables new to tunable membership are listed here
    obedient = [[n:nodes]true];
    penalties = [[n:nodes] [[m:nodes] 0]];
    rewards= [[n:nodes] [[m:nodes] 0]];
    dd_prev = [[n:nodes] 0];
    dd2_prev = [[n:nodes] 0];
    dd2_prev2 = [[n:nodes] 0];
    R_dd_prev = [[n:nodes] 0];
    dd_synchr_prev = [[n:nodes]true]
      TRANSITION
      [
    pc=0 -->
        fvec' IN {v:ARRAY nodes OF fdomain|
            fcounter(v,fvec,fvec_prev,byz,N,0)<=1 AND
            (fcounter(v,fvec,fvec_prev,symm,N,0)+fcounter(v,fvec,fvec_prev,byz,N,0) > 0
                => N > 2*fcounter(v,fvec,fvec_prev,symm,N,0) + 2*fcounter(v,fvec,fvec_prev,byz,N,0) +
                    fcounter(v,fvec,fvec_prev,ben,N,0) + 1)};
        fvec_prev'=fvec;
        pc' = 1;
      []
    pc=1 -->
        ls' IN {v:ARRAY nodes OF vecdomain|FORALL(m,n:nodes):
            (fvec[n]=corr OR fvec[n]=symm => v[m][n]=1) AND
            (fvec[n]=ben => v[m][n]=0) AND
            v[m][n]/=2};
        ls_prev'=ls;
        obedient'=[[n:nodes]IF fvec[n]=byz OR fvec[n]=symm THEN false ELSE obedient[n] ENDIF];
        pc' = 2;
      []
    pc=2 -->
        al_m' IN {v:ARRAY nodes OF matdomain|FORALL(m,n:nodes):
            (fvec[n]=corr => v[m][n]=ls_prev[n]) AND
            (ls[m][n]=0 <=> v[m][n]=[[1:nodes]2]) AND
            ((EXISTS(l:nodes):v[m][n][1]=2)=>v[m][n]=[[1:nodes]2]) AND
            (fvec[n]=symm => FORALL(l,k:nodes):v[m][n][l]=v[k][n][l])};
        pc' = 3;
      []
```

136 and 138). The delays of this protocol in the frame-based case are increased by one round with respect to the tunable membership protocol. Therefore, we use the divergence degrees from one round earlier. In addition, synchrony and liveness must hold for active nodes only. SAL's BMC model checker has proven both properties for the chosen parameters within a time of up to a few hours.

TABLE XI

SAL CODE OF THE TUNABLE MEMBERSHIP PROTOCOL (PART 2)

```
55      pc=3 -->
56          cons_hv'=[[n:nodes]
57              IF EXISTS(j:nodes):h_maj(al_m[n])[j]=2 THEN ls_prev[n]
58              ELSE h_maj(al_m[n]) ENDIF];
59          pc' = 4;
60        []
61      pc=4 -->
62          ls'=[[m:nodes] [[n:nodes]
63              IF cons_hv[m]/=al_m[m][n] THEN 0 ELSE ls[m][n] ENDIF]];
64
65          penalties' = [[m:nodes][[n:nodes]
66              IF penalties[m][n]<P THEN
67                  IF cons_hv[m][n]=0 THEN penalties[m][n]+1
68                  ELSE IF rewards[m][n]=R-1 THEN 0 ELSE penalties[m][n] ENDIF
69                  ENDIF
70              ELSE penalties[m][n] ENDIF]];
71
72          rewards' = [[m:nodes][[n:nodes]
73              IF cons_hv[m][n]=0 THEN 0
74              ELSE IF rewards[m][n]=R-1 THEN 0 ELSE rewards[m][n]+1 ENDIF ENDIF]];
75
76          dd2_prev' = [[i:nodes]
77              IF FORALL(j:nodes):penalties[j][i]<P THEN
78                  IF  (fvec_prev[i]=ben OR EXISTS(j,l:nodes):ls_prev[i][j]/=cons_hv[l][j])
79                      THEN IF dd2_prev[i]<2*P THEN dd2_prev[i]+1 ELSE dd2_prev[i] ENDIF
80                  ELSE 0 ENDIF
81              ELSE dd2_prev[i] ENDIF];
82          dd2_prev2' = dd2_prev;
83
84          dd_prev' = [[i:nodes]
85              IF FORALL(j:nodes):penalties[j][i]<P THEN
86                  IF  (fvec_prev[i]=ben OR EXISTS(j,l:nodes):ls_prev[i][j]/=cons_hv[l][j])
87                      THEN IF dd_prev[i]<P THEN dd_prev[i]+1 ELSE dd_prev[i] ENDIF
88                  ELSE IF R_dd_prev[i]=R THEN 0 ELSE dd_prev[i] ENDIF ENDIF
89              ELSE dd_prev[i] ENDIF];
90
91          R_dd_prev'=[[i:nodes]
92              IF fvec_prev[i]=ben OR EXISTS(j,l:nodes):ls_prev[i][j]/=cons_hv[l][j] THEN 0
93              ELSE IF R_dd_prev[i]<R THEN R_dd_prev[i]+1 ELSE R_dd_prev[i] ENDIF ENDIF];
94
95          dd_synchr_prev'=[[i:nodes]
96              IF  dd_synchr_prev[i] AND
97                  dd_prev[i]<(P DIV 2)+(P MOD 2) AND
98                  (dd_prev[i]=(P DIV 2)+(P MOD 2)-1 =>
99                      fvec_prev[i]/=ben AND FORALL(j,l:nodes):ls_prev[i][j]=cons_hv[l][j])
100                 THEN true
101             ELSE false ENDIF];
102
103         pc' = 0;
104       []
105           ELSE -->
106       ]
107   END;
108
109
110   synchrony: THEOREM system |- G(FORALL(m,n:nodes):pc=0 AND dd_synchr_prev[n] AND obedient[n] => penalties[m][n]<P);
111   liveness: THEOREM system |- G(FORALL(m,n:nodes):pc=0 AND dd2_prev2[n]=2*P AND obedient[n] => penalties[m][n]=P);
112 END
```

TABLE XII

SAL CODE OF THE PARTITIONABLE MEMBERSHIP PROTOCOL (PART 1)

```
pmembership{;N: natural,P:natural,R:natural}: CONTEXT = BEGIN

%types and constant declaration comes here

system: MODULE =
    BEGIN
      LOCAL
    %only variables new to partitionable membership are listed here
    al_mv: ARRAY nodes OF matdomain,
    cons_mv:ARRAY nodes OF vecdomain,
    fvec_prev2: ARRAY nodes OF fdomain,
    fvec_prev3: ARRAY nodes OF fdomain,
    p_start: [1..N],
    k: [0..N],
    k_prev: [0..N],
    k_prev2: [0..N],
    k_prev3: [0..N],
    inactive: ARRAY nodes OF BOOLEAN,
    dd2_prev3: ARRAY nodes OF [0..2*P],
    dd_synchr_prev2: ARRAY nodes OF BOOLEAN
      INITIALIZATION
    %only variables new to partitionable membership are listed here
    al_mv = [[n:nodes] [[m:nodes] [[1:nodes] 1]]];
    cons_mv = [[n:nodes] [[m:nodes] 1]];
    fvec_prev2 = [[n:nodes] corr];
    fvec_prev3 = [[n:nodes] corr];
    p_start = 1;
    k = 0;
    k_prev = 0;
    k_prev2 = 0;
    k_prev3 = 0;
    inactive = [[n:nodes]false];
    dd2_prev3 = [[n:nodes] 0];
    dd_synchr_prev2 = [[n:nodes]true]
      TRANSITION
      [
    ([](k_curr:[0..N]):([](ps:[1..N]):
    pc=0 -->
        fvec' IN {v:ARRAY nodes OF fdomain|fcounter(v,fvec,fvec_prev,fvec_prev2,byz,N,0)<=1 AND
            N > 2*fcounter(v,fvec,fvec_prev,fvec_prev2,symm,N,0) + 2*fcounter(v,fvec,fvec_prev,fvec_prev2,byz,N,0) +
            2*fcounter(v,fvec,fvec_prev,fvec_prev2,ben,N,0) +
            2*pcounter(v,fvec,fvec_prev,fvec_prev2,k_curr,k,k_prev,k_prev2,N,0) +1
            AND FORALL(n:nodes):inactive[n] => v[n]=ben};
        fvec_prev'=fvec;
        fvec_prev2'=fvec_prev;
        fvec_prev3'=fvec_prev2;
        k' = k_curr;
        k_prev' = k;
        k_prev2' = k_prev;
        k_prev3' = k_prev2;
        p_start'=ps;
        pc' = 1;))
      []
    pc=1 -->
        ls' IN {v:ARRAY nodes OF vecdomain|FORALL(m,n:nodes):
            (k=0 OR
            (n/=k AND m/=k) OR
            n=m OR
            n<p_start =>
                (fvec[n] =corr OR fvec[n]=symm => v[m][n]=1) AND
                (fvec[n]=ben  => v[m][n]=0)) AND
            (NOT(k=0 OR (n/=k AND m/=k) OR n=m OR n<p_start) => v[m][n]=0) AND
            v[m][n]/=2};
        ls_prev'=ls;
        obedient'=[[n:nodes]IF fvec[n]=byz OR fvec[n]=symm THEN false ELSE obedient[n] ENDIF];
        pc' = 2;
      []
    pc=2 -->
        al_m' IN {v:ARRAY nodes OF matdomain|FORALL(m,n:nodes):
            (fvec[n]=corr AND ls[m][n]/=0 => v[m][n]=ls_prev[n]) AND
            (ls[m][n]=0 <=> v[m][n]=[[1:nodes]2]) AND
                    ((EXISTS(1:nodes):v[m][n][1]=2)=>v[m][n]=[[1:nodes]2]) AND
            (fvec[n]=symm => FORALL(1,k:nodes):v[m][n][1]=v[k][n][1])};

        al_mv' IN {v:ARRAY nodes OF matdomain|FORALL(m,n:nodes):
            (fvec[n]=corr AND ls[m][n]/=0 =>
                v[m][n]=[[1:nodes]IF penalties[n][1]=P THEN 0 ELSE 1 ENDIF]) AND
            (ls[m][n]=0 <=> v[m][n]=[[1:nodes]2]) AND
                    ((EXISTS(1:nodes):v[m][n][1]=2)=>v[m][n]=[[1:nodes]2]) AND
            (fvec[n]=symm => FORALL(1,k:nodes):v[m][n][1]=v[k][n][1])};
        pc' = 3;
      []
```

TABLE XIII

SAL CODE OF THE PARTITIONABLE MEMBERSHIP PROTOCOL (PART 2)

```
83      pc=3 -->
84          cons_hv'=[[n:nodes] IF EXISTS(j:nodes):h_maj(al_m[n])[j]=2 THEN ls_prev[n] ELSE h_maj(al_m[n]) ENDIF];
85          cons_mv'=[[n:nodes] maj(al_mv[n])];
86          pc' = 4;
87        []
88      pc=4 -->
89          ls'=[[m:nodes] [[n:nodes] IF cons_hv[m]/=al_m[m][n] THEN 0 ELSE ls[m][n] ENDIF]];
90
91          penalties' = [[m:nodes][[n:nodes]
92              IF penalties[m][n]<P THEN
93                  IF cons_hv[m][n]=0 THEN penalties[m][n]+1
94                  ELSE IF rewards[m][n]=R-1 THEN 0 ELSE penalties[m][n] ENDIF ENDIF
95              ELSE penalties[m][n] ENDIF]];
96
97          rewards' = [[m:nodes][[n:nodes]
98              IF cons_hv[m][n]=0 THEN 0
99              ELSE IF rewards[m][n]=R-1 THEN 0 ELSE rewards[m][n]+1 ENDIF ENDIF]];
100
101         dd2_prev' = [[i:nodes]
102             IF  (fvec_prev[i]=ben OR EXISTS(j,m:nodes):
103                 is_prim_part(m,k,k_prev,k_prev2,k_prev3) AND (NOT inactive[m]) AND ls_prev[i][j]/=cons_hv[m][j])
104                 THEN IF dd2_prev[i]<2*P THEN dd2_prev[i]+1 ELSE dd2_prev[i] ENDIF
105             ELSE 0 ENDIF];
106         dd2_prev2' = dd2_prev;
107         dd2_prev3' = dd2_prev2;
108
109         dd_prev' = [[i:nodes]
110             IF  (fvec_prev[i]=ben OR EXISTS(j,m:nodes):
111                 is_prim_part(m,k,k_prev,k_prev2,k_prev3) AND (NOT inactive[m]) AND ls_prev[i][j]/=cons_hv[m][j])
112                 THEN IF dd_prev[i]<P THEN dd_prev[i]+1 ELSE dd_prev[i] ENDIF
113             ELSE IF R_dd_prev[i]=R THEN 0 ELSE dd_prev[i] ENDIF ENDIF];
114
115         R_dd_prev'=[[i:nodes]
116             IF  fvec_prev[i]=ben OR EXISTS(j,m:nodes):
117                 is_prim_part(m,k,k_prev,k_prev2,k_prev3) AND (NOT inactive[m]) AND ls_prev[i][j]/=cons_hv[m][j] THEN 0
118             ELSE IF R_dd_prev[i]<R THEN R_dd_prev[i]+1 ELSE R_dd_prev[i] ENDIF ENDIF];
119
120         dd_synchr_prev'=[[i:nodes]
121             IF  dd_synchr_prev[i] AND
122                 dd_prev[i]<(P DIV 2)+(P MOD 2) AND
123                 (dd_prev[i]=(P DIV 2)+(P MOD 2)-1 =>
124                     fvec_prev[i]/=ben AND (FORALL(j,m:nodes):
125                     is_prim_part(m,k,k_prev,k_prev2,k_prev3) AND (NOT inactive[m]) => ls_prev[i][j]=cons_hv[m][j]))
126                 THEN true
127             ELSE false ENDIF];
128         dd_synchr_prev2'=dd_synchr_prev;
129
130         inactive' = [[n:nodes] IF inactive[n] OR EXISTS(m:nodes):cons_mv[n][m]=2 THEN true ELSE false ENDIF];
131         pc' = 0;
132       []
133           ELSE -->
134       ]
135   END;
136 psynchrony: THEOREM system |- G(FORALL(n,m:nodes):pc=0 AND dd_synchr_prev2[n] AND obedient[n] AND (NOT inactive[m])
137                           => cons_mv[m][n]=1);
138 pliveness: THEOREM system |- G(FORALL(m,n:nodes):pc=0 AND dd2_prev3[n]=2*P AND obedient[n] AND (NOT inactive[m])
139                           => cons_mv[m][n]=0);
140 END
```