A Group Membership Protocol for Communication Systems with both Static and Dynamic Scheduling

Valério Rosset, Pedro F. Souto and Francisco Vasques
Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias s/n
4200-465 Porto, Portugal
vrosset@fe.up.pt, pfs@fe.up.pt, vasques@fe.up.pt

Abstract

We present a group membership protocol specially designed for next generation communication systems for real-time safety-critical applications such as FlexRay and FTT-CAN. The proposed protocol imposes an overhead of two bits per processor per communication cycle, when the system is in a quiescent state, and is able to tolerate benign failures of up to half of the group members between consecutive executions. Additionally, it removes a faulty processor within two communication cycles in the worst case and reintegrates a processor at the latest two communication cycles after it recovers. Compared with protocols developed for similar systems, it is as tolerant as the most robust protocol with a traffic overhead slightly higher than the most efficient protocol, which is much less robust.

1. Introduction

Group membership is considered an important abstraction to facilitate the provision of fault tolerance in systems in general [1] and in safety-critical applications in particular [2]. In this paper, we describe a group membership protocol for real-time safety-critical applications, specially designed for communication systems that support both static and dynamic communication scheduling in a communication cycle such as FlexRay [3] and FTT-CAN [4]. This is the first protocol of this kind that we are aware of. Both FlexRay and FTT-CAN [5] provide a basic set of fault-tolerant communication services, but no group membership.

A group membership protocol comprises two fundamental operations: failure detection and agreement. Failure detection is performed locally by a processor, by monitoring the messages it receives, or it does not receive, from other processors. Agreement is achieved through the exchange among the different processors of the perceived operating state of processors in the system.

Currently, many communications systems for safety critical real-time applications [2] are synchronous with only static communications scheduling implemented on top of a protocol based on Time-Division Multiple Access (TDMA). I.e., in those protocols it is assumed that in each TDMA cycle, each processor can send a fixed amount of traffic, which must be sufficient to satisfy the worst case traffic requirements of all the applications in the processor. Therefore, virtually all group membership protocols for this class of systems described in the literature use the messages sent in a TDMA cycle to detect failure of a remote processor. Furthermore, to achieve agreement, they exchange processor state information in every TDMA cycle and strive to minimize this information. Essentially, the differences among the published protocols depend on the failure assumptions; usually, the stronger these assumptions, the more efficient the protocol. For example, the group membership protocol [6, 7] executed in every cycle of the TTP/C [8], requires the sending of only one additional bit per cycle per processor [9]. However, it can tolerate at most one failure within any interval of two TDMA cycles. Actually, the implementation of the protocol takes advantage of this property, and achieves group membership without sending any additional bit. Given that the protocol interprets a communication failure as a processor failure, this assumption is too strong for the operation environment of many safety-critical applications. Therefore, in the TTP/C, if a processor detects a potential violation of this assumption, it switches to the blackout operating mode, in which system operation is severely degraded. On the other hand, the protocol of Ezhilchelvan and Lemos [10] tolerates the failure of up to half of non-faulty processors in three consecutive TDMA cycles, but requires that every processor broadcast a vector with the state of every other processor in every TDMA cycle. Given that buses for safety-critical applications may have several tens of processors [11], this protocol may lead to TDMA cycles with a duration longer than required by the real-time applications that execute in the system.

The proposed protocol relies on the observation that group membership does not change in every TDMA cycle, and takes advantage of next generation communication protocols for safety-critical applications such as FTT-CAN and FlexRay. In these communication protocols

the TDMA cycle is divided in two segments: a statically scheduled segment essentially to support periodic traffic, and a dynamically scheduled segment essentially to support aperiodic traffic or traffic required for non-safety-critical applications. Therefore, the basic idea of the protocol is to exchange information on the processors state in the dynamically scheduled segment only and when there is a change to the membership. In a quiescent state, when there are no membership changes, the protocol requires only an overhead of two bits per processor per cycle that are sent in the static segment. Note that even in the case of membership changes , the protocol requires the exchange of less state information than the protocol proposed by Ezhilchelvan and Lemos, while tolerating a higher fault arrival rate than their protocol.

The remainder of the paper is organized as follows. In the next section we present the system model, including the fault assumptions. In Section 3 we provide the specification of the group membership protocol. A protocol satisfying this specification is presented in Section 4. The protocol is developed gradually and we provide informal arguments for the correctness of each version. In Section 5 we outline its implementation on top of FlexRay. Related work is discussed in Section 6. Finally we summarize the results and present some directions for future work in Section 7.

2. Model

We assume a synchronous system that is composed of a fixed set of processors P that are connected via a broadcast network, in which a processor receives every message it broadcasts.

The execution model is based on the one presented in [12]. Each processor begins its execution in some start state and then repeatedly executes, in lock-step with the other processors, a *phase* that has two steps:

Communication step, in which each processor generates a *message*, if any, that depends on the processor's *state*, broadcasts it to the network, and receives messages broadcasted in this step;

Processing step, in which each processor generates the new state, by processing the messages received in the communication step.

Note that a state is comprised of the values of a set of state-variables and the set of states may not be finite. Some states are *halting states*, i.e. a processor in such a state will not send any message or modify its state.

By using this model, we abstract away some details that are not essential, thus simplifying the presentation of the protocol. For example, we do not consider the exact time when a processor broadcasts its message. We assume that some protocol ensures that every processor is granted access to the network to broadcast its message and that every processor knows when no more messages broadcasted in the communication step will be received, so that the processing step can be executed.

Processors can fail by experiencing one of three types of faults: a *crash fault*, i.e. a processor enters an halting state and takes no further action; a *receive fault*, i.e. a fault on reception, that prevents a processor from receiving a message broadcasted by another processor in that step; a *send fault*, i.e. a fault on broadcasting, that prevents a processor from broadcasting a message to the network. Note that receive and send faults do not need to be persistent, e.g. a processor may have a receive fault in a phase, but be able to receive a message in a later phase. We say that a processor is non-faulty if it has not experienced any fault since the beginning of the execution.

Finally, we assume that the network provides a reliable broadcast service. I.e. a message broadcasted by a non-faulty processor will be correctly received by all other non-faulty processors.

3. Group Membership Specification

We state the Group Membership problem in terms of the set of group members (M-SET) maintained by every processor.

We consider essentially two properties:

Agreement: All non-faulty group members compute the same M-SET.

Validity:

1. A faulty processor will be removed from the M-SET of a non-faulty group member in a bounded time interval;
2. A non-faulty processor attempting to be reintegrated will be added to the M-SET of a non-faulty group member in a bounded time interval.

In addition to these properties, Cristian [1] defines stability properties of the group membership to ensure that a group's membership does not change for no reason. The proposed protocols satisfy such properties, but we will not provide the arguments for that, because of lack of space.

4. Group Membership Protocol

The protocol has two phases that are repeated in every cycle: Failure Detection phase (FD-phase) and the Group Membership phase (GM-phase). Every group member is required to broadcast one message in the FD-phase, so that failures can be detected with bounded delay. In the GM-phase a protocol may be executed to achieve agreement on the group membership.

We present first a very basic protocol that does not support processor reintegration and that requires the sending of messages in both phases of every cycle. Then we add support for processor reintegration and finally present a

```
Set majSet(Set S, SetofSet R, int n)
begin
    Set M to the ∅
    for every p in S do
        if p is an element of ⌈n/2⌉ or more sets in R
        then add p to M
        else if p is not an element of ⌈n/2⌉ or more sets in R
        then continue
        else return undefined
        end
    end
    return M
end
```

Figure 1. majSet function used by all versions of the protocol.

version that not only supports processor reintegration, but also does not require sending messages in the GM-phase of every cycle. Note that the goal of presenting several versions with increasing complexity is to facilitate the understanding of the final protocol. The intermediate protocols are not intended to be the most efficient of their kind.

All three versions use the same majority function, majSet, that computes the set of elements that are members of at least a majority of n sets. A special value, *undefined*, is returned when no majority is found. Figure 1 shows the pseudo-code for the majSet function. Note that undefined is returned only when the number of sets in R is smaller than n and there is no agreement among the sets in R. E.g., consider the invocation $\mathrm{majSet}(\{a, b\}, \{\{a, b\}, \{a\}\}, 3)$. In this case, although a is a member of two sets in R, a majority for $n = 3$, there is no majority of sets agreeing on the membership of b. Thus this invocation would return undefined.

4.1. Base Protocol

In addition to the M-SET, in the base protocol every processor maintains the M-set, the set of candidate members, and an integer, u, an upper bound on the size of the group. Initially, both sets are set to P, the set of processors, and u is set to the size of P.

During the FD-phase, every group member broadcasts a heartbeat message and receives the heartbeats broadcasted by group members in this phase. Then it removes from its M-set the processors from which no heartbeat message was received in this phase.

During the GM-phase a group member broadcasts the M-set it computed in the FD-phase, and receives the corresponding sets broadcasted by group members. It then computes the set of candidate members proposed by a majority of all the group members, the Maj-set, by applying the majSet to the M-sets it received from group members, i.e. processors in the M-SET. If this set is *undefined* or different from the processor's M-set or the processor is not in the Maj-set, then it halts. Otherwise, it uses the Maj-set and the set of all M-sets to compute the new group membership, i.e. the M-SET.

Group Membership Protocol (Base Version)

State

P the set of all processors

M-SET the set of group members, initially set to P

M-set the set of candidate group members, initially set to P

u upper bound of the group's size, initially set to |P|

FD-phase

Communication step: Broadcast heartbeat message.

Processing step: Remove from the M-set every processor from which no heartbeat message was received.

GM-phase

Communication step: Broadcast the M-set.

Processing step:

1. Let Maj-set be the result of applying the majSet function to P, the set of all the M-set's received from processors in the M-SET and u.
2. If the Maj-set
   (a) is undefined, or
   (b) is different from the M-set the processor broadcasted, or
   (c) does not contain the processor;
   then halt.
3. Remove from the M-set every processor from which an M-set different from the Maj-set was received.
4. Set u to the size of the M-set.
5. Remove from the M-set every processor from which no message was received in this phase.
6. Set the M-SET to the M-set.

Properties of the Base Protocol We now argue that this protocol satisfies the Agreement and the first of the Validity properties stated above.

Agreement A rigorous proof of Agreement can be done by induction on the number of cycles. Here we provide informally the arguments that can be used in such a proof. First, note that given the assumptions on the communication subsystem, every non-faulty processor receives the same set of messages in every phase. Because the computation of the M-SET is based on a deterministic algorithm and, assuming that every non-faulty group member computed the same values for u and for M-SET in the previous cycle, every non-faulty group member uses the same inputs, therefore every non-faulty group member will compute the same values for u and for the M-SET.

Note that the base protocol can be seen as yet another instance of the state machine approach to fault-tolerance [13, 14], and the arguments provided in the previous paragraph are the standard arguments of such an approach and independent of the problem at hand. The assumptions on the communication subsystem make the arguments easier, because they imply interactive consistency with the fault model considered, i.e. the communication subsystem provides a reliable broadcast service.

Validity It can be shown that the base protocol ensures that a faulty group member will be removed from the M-SET of a non-faulty group member in no later than two cycles. I.e., if a group member has a fault in a cycle, then it will be removed from the M-SET of non-faulty group members by the end of the following cycle, in the worst case, as long as a majority of group members remain non-faulty.

The proof can be done by case analysis, considering each of the three possible processor faults in each of the two protocol phases. Here we just outline the main arguments used in that case analysis. Send faults are easily detected by a non-faulty group member because it will not receive the message the faulty processor was supposed to send. If the fault occurs in the FD-phase, then the processor will be removed from the M-set of non-faulty group members, and because they are a majority it will not be a member of the Maj-set and consequently of the M-SET by the end of that cycle. If the fault occurs in the GM-phase, then non-faulty group members will not receive the M-set from the faulty processor and will remove it from their M-SET by the end of that cycle. Processor crashes are detected by other processors when the crashed processor does not send a message it was supposed to send. Therefore this kind of fault is similar to a send fault, except that there may be an extra delay between the occurrence of the fault and its detection. This delay can be as long as one phase when a processor crashes immediately after sending the message in a phase. In any case, a crashed processor will be removed from the M-SET of non-faulty group members at the latest by the end of the following cycle. Receive faults are detected by comparing the M-set received from the faulty processor with the Maj-set computed in the GM-phase. If a processor has a receive fault in the FD-phase, then it will erroneously remove the sender of the missing message from its M-set whereas non-faulty group members will not. Therefore, the M-set broadcasted in the GM-phase by the faulty processor will differ from the Maj-set computed by non-faulty group members in at least the sender of the missed message, and the faulty processor will be removed from the M-SET of non-faulty group members. Receive faults in the GM-phase will result in the sender of the corresponding message being removed from the M-SET of the faulty processor because it did not receive the former's M-set. This will be detected in the following cycle, because the M-set broadcasted by the faulty processor will not include the sender of the missed message, whereas that of non-faulty group members will.

Note that we do not claim that we have just given a proof of correctness of the base protocol. Actually, the last sentence of the previous paragraph will be true only if the processor that sent the missed message does not fail to send its message in the FD-phase of the following cycle. Therefore, with rigor, the base protocol satisfies Validity only if we strengthen the fault assumptions by excluding send faults in the FD-phase by the senders of messages on which other processors had a receive fault in the GM-phase of the previous cycle. It is possible to modify the protocol to eliminate this exception, but it requires sending extra information in the GM-phase message.

Faulty Processor Behavior Usually, it is easier to build fault-tolerant systems if faulty processors fail silently. Because of that, the protocol requires a processor to halt whenever its state indicates that the processor may be faulty, given the assumptions made. As a result:

Proposition A faulty processor will halt at the latest by the end of the cycle after which it has a fault.

Again, this proposition can be proven by induction and by case analysis using essentially the arguments used above to argue for Validity; the caveat regarding receive faults in the GM-phase still applies. But the key to the proof is that the value u computed is an upper bound of the group size. As a result, under our fault assumptions, faulty processors will never be able to compute a Maj-set different from that computed by non-faulty processors. Note that if u were computed at the end of the processing step of the GM-phase rather than in point 4, a faulty processor might compute a value lower than the actual group size. Therefore, it could compute a different non-undetermined Maj-set, even though there were a majority of non-faulty group members, and do not halt.

4.2. Processor Reintegration

The base protocol does not handle processor reintegration. From a pratical point of view processor reintegration is very important as it allows reintegrating not only repaired processors but also non-faulty processors that were excluded because of transient communication faults.

Handling processor reintegration requires a very simple modification to the base protocol. Essentially, a processor that wishes to join (or rejoin) the group needs to send a *join request* message instead of the heartbeat message in the FD-phase and then it has to participate in the GM-phase that follows, as if it were a member of the group. However, because its state may not be in agreement with that of group members, its messages have to be handled in a special way in the GM-phase. Furthermore, every group member has to broadcast not only its M-set but also its group size upper-bound, so that a joining processor can compute the Maj-set.

The new version of the protocol is presented next and changes with respect to the base protocol are highlighted in bold.

Group Membership Protocol (Reintegration Version)

State

P the set of all processors

M-SET the set of group members, initially set to P

M-set the set of candidate group members, initially set to P

u upper bound of the group's size, initially set to $|P|$

FD-phase

Communication step:
If processor is group member
Then broadcast hearbeat message
Else if wishing to join group
Then set the M-set and the M-SET to P, u to the size of P, and broadcast join-req message.

Processing step:

1. Remove from the M-set every processor from which no heartbeat message was received.
2. **For every join-req message received add its sender to the M-set.**

GM-phase

Communication step:
Broadcast message with the M-set **and the group's size upperbound, u.**

Processing step:

1. Let Maj-set be the result of applying the majSet function to P, the set of all the M-set's received from processors in the M-SET **and the minimum of all u's received in the same messages**.
2. If the Maj-set
   (a) is undefined, or
   (b) is different from the M-set the processor broadcasted **and the processor is a member of the group,** or
   (c) **is not a subset of the M-set the processor broadcasted and the processor is joining the group, or**
   (d) does not contain the processor
   then halt.
3. Remove from the M-set
   (a) every group member from which an M-set different from the Maj-set was received;
   (b) **every joining processor whose M-set is not a superset of the Maj-set;**
4. Set u to the size of the M-set.
5. Remove from the M-set every processor from which no message was received in this phase.
6. Set the M-SET to the M-set.

Properties of the Protocol It can be shown that this protocol satisfies both the Agreement and the Validity properties, with the same strengthening of the fault assumption as for the base protocol, as long as more than half of the group members remain non-faulty from one group to the next. The delay for removing faulty processors or to add joining processors is, in the worst case, two cycles.

As mentioned above, Agreement is straightforward as the protocol uses the state machine approach to fault-tolerance. Note that although the state of joining processors at the beginning of a cycle may not be equal to that of group members, the computation of the M-SET does not use directly that state, but rather information derived from the state that is broadcasted to all processors, including the sender itself.

With respect to Validity, the reasoning that faulty processors are removed from the M-SET of non-faulty group members still applies. However, correctness also depends on the upper bound of non-faulty group members being the minimum of all the upper bounds. As already mentioned, this can be shown by induction on the number of cycles, and relies on faulty processors halting no later than the cycle when they have a fault, which is assured by the protocol. The same arguments can be used to show that faulty joining processors either will not be added to the M-SET or will be removed from the M-SET in the cycle immediately after being added.

The addition of non-faulty joining processors to the M-SET of non-faulty group members can be argued based on the fact that Join request messages are received by all the non-faulty group members. Therefore every non-faulty joining processor is added to the M-set of every non-faulty group member by the end of the FD-phase, and the M-set broadcasted by non-faulty group members in the GM-phase will include every non-faulty joining processor. Furthermore, even though a joining processor may not send an M-set equal to the Maj-set, because it initializes its M-set to the set of all processors, P, its M-set will be a superset of the Maj-set, and therefore a non-faulty joining processor will be a member of the M-SET of non-faulty group members by the end of the cycle. The delay for joining is in the best case one cycle and in the worst case two cycles, because a joining processor must execute a full FD-phase.

4.3. Protocol without GM-phase Messages

The versions of the group membership protocol described so far require sending messages in every GM-phase, and therefore are not particularly advantageous with respect to other protocols that were developed for communications systems with a static schedule only, e.g. the protocol by Echilzelvan and Lemos [10].

However, the decomposition of the protocol in two-phases, FD-phase and GM-phase, with failure detection occurring in the FD-phase and agreement in the second phase, allows for the non-execution of the latter, if there are no events that trigger the change of the group membership. This way, group membership is maintained with virtually no messages when the system is in a quiescent state, and the dynamically scheduled segment will be available for other traffic.

In the new and final version of the protocol, a processor sends a GM-phase message when it detects a fault that may lead to modifying the group membership. Furthermore, in the FD-phase, processors may explicitly request the sending of GM-phase messages. This is used after a GM-phase execution in which a processor modified its M-SET in a way that other processors may not be aware of.

Because the GM-phase does not have to be executed in every cycle, there is the possibility that a faulty group member will not detect changes in the group membership, and later create havoc. To prevent that, every group member keeps a group id, a monotonically increasing integer variable, that is incremented by one every time there is a group membership change. The group id is sent together

with the M-set and the group size upper bound in the GM-phase. The correctness of the final version relies on the fact that the group id of non-faulty group members is the maximum of all the group ids of all processors.

The final version of the GM protocol is presented next and changes with respect to the version supporting reintegration are highlighted in bold.

Group Membership Protocol (Final Version)

State

P the set of all processors

M-SET the set of group members, initially set to P

M-set the set of candidate group members, initially set to P

u upper bound of the group's size, initially set to $|\mathrm{P}|$

group-id integer with group id, initially set to 0

GM-request boolean indicating whether execution of the GM-phase should be performed, initially set to false

FD-phase

Communication step:
If processor is group member
Then broadcast hearbeat message **with GM-request as determined in the previous GM-phase**
Else if wishing to join group
Then set the M-set and the M-SET to P,
u to the size of P, **the group-id to zero**
and broadcast a join-req messsage.

Processing step:

1. Remove from the M-set every processor from which no heartbeat message was received.
2. For every join-req message received add its sender to the M-set.
3. **If received a message with GM-request or modified the M-set in 1 or 2 Then set GM-request.**

GM-phase

If GM-request is set, then

Communication step:
Broadcast message with the M-set, the group's size upperbound, u, **and the group-id**.

Processing step:

1. **Let max-id be the maximum of the group ids received.**
2. **If the processor is joining Then set the group-id to max-id Else if its group-id is different from max-id Then halt**
3. Let Maj-set be the result of applying the majSet function to P, the set of all the M-set's received from processors **with a group-id equal to max-id,** and to the minimum of all u's received in the same messages.
4. If the Maj-set
   (a) is undefined, or
   (b) is different from the M-set the processor broadcasted and the processor is a member of the group, or
   (c) is not a subset of the M-set the processor broadcasted and the processor is joining the group, or
   (d) does not contain the processor

   then halt.
5. Remove from the M-set
   (a) every group member from which an M-set different from the Maj-set was received;
   (b) every joining processor whose M-set is not a superset of the Maj-set;
6. Set u to the size of the M-set.
7. Remove from the M-set every processor from which no message was received in this phase.
8. **If removed some processor from M-set in 7 Then set the GM-request Else reset the GM-request.**
9. Set the M-SET to the M-set **and increment the group-id.**

Note that it is essential to execute a GM-phase after another GM-phase in which a processor removes from the M-set processors from which it did not receive their M-set. This will allow detection of faults in the GM-phase. These faults are detected locally by each receiving processor, but by itself the receiving processor is unable to determine whether it had a receive fault or the sending processor had a send fault. To resolve this dilemma, a processor needs to know what is the perception of other group members.

Protocol Properties Again, it can be shown that the final version of the protocol supports the Agreement and the Validity properties with the same fault assumption strengthening as for the previous versions, i.e. no send fault in the FD-phase by a processor that sent a GM-phase message in the previous cycle on which another processor had a receive fault, and assuming that at least a majority of group members remain non-faulty. As in the other versions of the protocol, with these assumptions, a faulty processor will be removed from the M-SET of non-faulty group members at the latest by the end of the cycle following the one in which it has a fault. Likewise, a non-faulty joining processor will be added to the M-SET of the non-faulty group members at the latest by the end of the cycle following the one in which it decides to join the group. It can also be shown, that under these fault assumptions, a faulty processor will halt the latest by the end of the cycle following the one in which it has a fault.

The arguments presented for the version of the protocol supporting processor reintegration still apply, as long as the group id of non-faulty group members is the maximum of all the ids. Again, a rigorous proof can be done by induction on the number of cycles. Informally, note that

if, in a GM-phase execution, a faulty group-member increments its group id, then it must have been able to compute a Maj-set with a value different from *undefined.* Because, faulty group members are a minority, then it must be the case that non-faulty group members also executed the GM-phase and will, therefore, increase their group id.

Regarding the upper bounds of delays of the Validity property, note that faults in the FD-phase will be detected as before in the same cycle, although in the case of a receive fault the faulty processor may be removed from the M-SET only in the following cycle, as a result of the processor halting in the GM-phase of that cycle. Faults in the GM-phase are detected as before, indeed if a processor has a receive fault it will request the execution of the GM-phase in the following cycle. Crash faults will originate send faults at the latest in the FD-phase of the following cycle, therefore the faulty processor will be removed from the M-SET at the latest the following cycle. With respect to the upper bound of the delay for a processor to join a group, the protocol execution is essentially identical to that of the version supporting processor reintegration only, and the arguments used there still apply. Thus, a non-faulty processor wishing to join the group will be added to the M-SET of non-faulty group members the latest by the end of the following cycle.

Note that just as non-faulty processors detect failure of a processor no later than the end of the following cycle, faulty processors also detect their own faults by the same time. Indeed, it can be shown by case analysis that for any fault different from a crash, a faulty processor will execute the GM-phase protocol either in the same cycle or in the following cycle, and in both cases it will halt.

5. Implementation Outline

The group membership protocol was designed to take advantage of the support for both static and dynamic scheduling in the next generation of communication systems for safety-critical applications. The FD-phase, which must be run in every cycle to ensure timely fault detection, can be scheduled statically, whereas the GM-phase, which may not need to be executed in every cycle, can be scheduled dynamically. In this section, we provide further details on a possible implementation of this protocol on top of the FlexRay protocol.

FlexRay FlexRay is a TDMA-based medium access control protocol that supports both static and dynamic communications scheduling. The FlexRay protocol is rather complex and below we omit many details to simplify its description. These omissions make the protocol less flexible than it really is, but they do not make it behave in a way that violates its specifications.

In FlexRay every TDMA cycle has a fixed length and it comprises two segments, a static segment and a dynamic segment, each of which has a fixed length. Both segments are divided in slots, however, whereas in the static segment all slots in a cycle have the same fixed length, which does not change between cycles, in the dynamic segment, the length of the slots may vary both within a cycle and between cycles. More precisely, in the dynamic segment, time is divided in mini-slots and each slot has an integer number of mini-slots that may vary, depending on the amount of data to transmit.

Each slot is assigned in every cycle to the same processor, implicitly determining the processor that has access to the medium. However, whereas in the static segment, every processor transmits a frame in its slots, even if it has no data to send, in the dynamic segment, a processor may not transmit a frame in a slot assigned to it, in which case the slot takes only one mini-slot.

Access control is implemented using a slot counter per processor, which identifies the frame that is transmitted in that slot, if any, and that is reset to one in the beginning of every TDMA cycle. The FlexRay protocol ensures that the slot counters of all non-faulty processors are synchronized. In the dynamic segment, the slot number effectively determines the priority of the frames: depending on the configuration, it may be possible for a set of frames to prevent other frames with a higher id from being sent to the medium.

Implementation Outline The group membership protocol puts some requirements on the communication system. First, it requires that every processor broadcast a heartbeat message in the FD-phase and that the communication system be able to detect the absence of messages. Second, it requires that, if necessary, GM-phase messages be sent in the next GM-phase. Third, it requires that processors be able to signal their wish to join the group or to request the execution of the GM-phase. All these requirements can be satisfied by FlexRay as we argue next.

Regarding the first requirement, we note that in FlexRay's static segment a processor will always transmit in its slots, even if it has no application data to send. Therefore, by assigning a slot of the static segment to each processor, we can use the frames broadcasted in the static segment as heartbeat messages of the FD-phase. The loss of one of these messages is detected by FlexRay, which provides several status bits that can be used to distinguish a send omission, caused by a crash of the sender, for example, from a message loss caused by communication faults, for example.

With respect to the GM-phase messages, they are broadcasted in the dynamic segment, because we expect that events that may lead to group membership changes do not occur that often. But when they do occur, it is important to broadcast these messages as soon as possible. An implementation satisfying this requirement is to reserve the first slots of the dynamic segment for the exchange of the GM-phase messages, by assigning one of these slots to each of the processors and binding the corresponding frame to the GM-phase message. This way, it is

ensured that GM messages will not be preempted by other messages of the dynamic segment. If a processor has no GM-phase message to send, the corresponding slot will take only one mini-slot. All the mini-slots remaining in the dynamic segment after the broadcast of GM-messages can be used to exchange application data that presumably does not have hard deadlines for transmission. Note that, if the maximum time available for the dynamically scheduled traffic is not enough for a complete execution of the GM-phase protocol, the GM-phase may have to be spread over several dynamic segments, i.e. TDMA cycles, leading to higher latency in the group membership change. Fault-rate assumptions will be affected accordingly.

Finally, there is a need for some mechanism allowing processors to signal their wish to join the group or to request the execution of the GM-phase. In our protocol, this is done in the FD-phase, i.e. in the static segment. Thus a possibility is to reserve two bits in the message sent by each processor in the static segment, a join-bit and a group-bit that, if set, indicate a join request and a GM execution request, respectively. The main problem with this implementation is that these bits will use some of the space reserved for application data. An alternative that avoids this overhead is to use the Network Management (NM) Vector, which is an optional field of the payload of a frame in the static segment. This solution has the problem that the NM Vector is an optional feature, and as such may not be implemented by every FlexRay communications controller.

Assumptions In addition to these requirements on the communication system, we made some assumptions regarding the capabilities of the communication system and the types of faults.

One important assumption is that the communication system supports reliable broadcast. This is certainly not true for FlexRay and, occasionally, a message broadcasted by a non-faulty processor may be corrupted by communication faults. We can make such scenarios less likely, by using FlexRay's support of duplicated communication channels: as long as a message is received correctly in one of the channels, it will be considered correct. Note that whereas FlexRay provides an abstraction of a single channel in the static segment, it does not provide such an abstraction in the dynamic segment. Thus, it is up to the application to build such an abstraction on top of FlexRay.

Whether or not a duplicated communication channel is used, messages will still be lost. In most cases, this will be detected by all non-faulty processors, but in some rare cases, an asymmetrical fault may result and some processors will discard the message whereas others will not. By assuming that the communication system is reliable, we assign the responsibility for these faults to processors. Essentially, we convert communication faults into processor faults. The symmetric case is not very problematic, if the group has several members, because it leads to the exclusion of only one processor. On the other hand, the case of asymmetric faults is not very well tolerated, as a single communication fault may lead to the exclusion of several processors. In an extreme case, if other faults occur, all processors may halt because there is no majority.

Another type of fault that may seriously affect the reliability of the protocol is the *babbling idiot*, a processor that transmits in slots other than its own. Our assumptions ruled out this type of fault, and therefore we should show that FlexRay prevents or masks these faults. Indeed, FlexRay provides a bus guardian to protect the communication channels in the static segment, by cutting off of the network the offending processor. However, communication in the dynamic segment is not similarly protected, and therefore such a processor might prevent the exchange of GM-phase messages. This is a limitation of the implementation of our protocol in FlexRay.

GM-phase Messages and Overhead A key issue regarding GM-phase messages is the encoding of the group id. In the protocol, the group id is an integer that is incremented in every execution, and therefore it will increase without bounds. In order to minimize the traffic overhead caused by the protocol, the group id must be encoded with the minimum number of bits. Because the protocol ensures that a faulty processor halts no later than the TDMA cycle after which it has a fault, two bits are enough for encoding the group id. The encoding of the M-set and of the group size upper bound present no difficulty. The M-set can be encoded as a bit-vector and the group size upper bound as an integer. Thus, assuming a system with 64 processors, each GM-phase message could be 9 bytes long, only: 8 bytes for the M-set and one byte packing the group size upper bound and the group id.

6. Related Work

The group membership problem has been the focus of a lot of research on both asynchronous and synchronous ditributed systems. In this section we concentrate on previous work for synchronous systems. For a discussion on the work for asynchronous systems we suggest the reading of the comprehensive survey [15] by Chockler, Keidar and Vitenberg.

In [1], Cristian provides a specification of the group membership problem for synchronous systems very similar to the one we use. Furthermore it proposes three protocols that solve that specification. There are however two important model assumptions that make the protocols described by Cristian and ours not directly comparable. On one hand, Cristian assumes a general communication system, whereas we consider a broadcast network with a TDMA medium access protocol where each processor broadcasts at least once every cycle. On the other hand, the fault assumptions made by Cristian are stronger than the ones we make: Cristian assumes that processors can fail only by crashing.

Much closer to our research is the extensive work on group membership that has been developed in the scope of the Time-Triggered Architecture and its protocol, TTP/C. The group membership protocol used in TTP/C is essentially the one described by Kopetz and Gruensteidl in [6], and an optimized version of this protocol with respect to failure detection latency is proposed in [7]. The system model used in these works is very similar to ours. In all cases, the network suports broadcast and uses a TDMA-based medium access protocol. The fault model assumed is also identical, in particular the network is supposed to be reliable, and processors are assumed to fail by crashing, or by failing to receive or to send messages. A key assumption of the TTP/C protocol is that there is at most one fault within an interval of two TDMA cycles. The protocol explores this assumption to ensure agreement on group membership among non-faulty processors without any communication overhead. This is achieved by sending with every message a CRC that covers not only the message but also part of the processor state that includes the group membership. Although the support of a duplicated network medium by TTP/C reduces the likelihood of communication faults, the occurrence of more than one communication fault in one TDMA cycle caused by electromagnetic interference, e.g., cannot be ruled out. Because of this, in TTP/C, processors will switch to the blackout operating mode if they detect that the fault assumption is violated. Operation in this mode is severely degraded. By contrast, our protocol supports the failure of up to half of the group members in one TDMA cycle, although at the cost of some communication overhead and by assuming an extended TDMA protocol.

More recently, with the adoption of a redundant star topology [16], the TTP/C group membership protocol is able to tolerate an arbitrary fault per TDMA cycle. It should be noted that this is possible because the bus guardian is now placed at the star coupler and it transforms an arbitrary fault at the processor into a fault that is tolerated by the group membership protocol. Failure of one star coupler can be masked by the other star coupler, but faults in this component cannot be arbitrary.

Ezhilchelvan and Lemos [10] proposed a group membership protocol also designed for broadcast networks using a TDMA medium access protocol. The fault model is similar to the one assumed in this paper: processors can experience send and receive faults and can fail by crashing, whereas the communications system is assumed to provide a reliable broadcast service. This protocol has some resemblance to ours, but there are significant differences that lead to very different performance. In [10], every processor maintains information on the group membership using a Membership Status Vector (MSV vector), which it broadcasts in every cycle. In contrast, in our protocol a processor only broadcasts group membership information whenever it detects a group membership change event, because the communication system supports both static and dynamic scheduling. In addition, the group membership information sent by each processor in our protocol is just half of the information sent in the protocol of [10]. Indeed, whereas in our protocol, the perceived state of a processor can be coded with 1 bit, each element of the MSV vector can have 3 values and therefore requires at least two bits. This higher efficiency has a cost in terms of the fault tolerance. Whereas the protocol of [10] is able to detect every processor fault type of the failure model assumed, in our protocol receive faults may be masked by send faults. On the other hand, our protocol tolerates the failure of up to half of the total number of non-faulty group members between two consecutive executions, whereas the protocol of [10] tolerates that many failures but in three consecutive cycles.

Walter, Lincoln and Suri [17] proposed a sequence of protocols for distributed on-line diagnosis, which is equivalent to the group membership problem, that are tolerant to increasing weaker fault assumptions. There are some similarities between these protocols and ours, in particular we base agreement on the majority function also used in [17], but there are also some major differences. First, [17] focus on fault diagnosis and is not concerned with processor reintegration. Second, their protocols require the exchange of diagnosis information in every cycle, whereas our protocol does not. This is possible because we consider only faults that can be locally detected by a non-faulty processor. On the other hand, [17] presents protocols that tolerate byzantine, or arbitrary, faults, which cannot be locally detected. The use of a stronger fault model has the additional advantage of requiring the exchange of less diagnosis information. There is clearly a trade-off between fault tolerance and efficiency that can be solved only by considering the requirements of the application.

The SPIDER group membership protocol [18] is an optimized version of the DD protocol presented in [17] that is used by the Reliable Optical Bus (ROBUS) communication system, which has a redundant active star topology. Again there are some resemblance between both protocols, but there are also major differences, as they were designed with two different communication systems in mind, each of which with its own emphasis. Whereas SPIDER was designed to improve the reliability of ROBUS and is designed to tolerate hybrid faults [19], including asymmetric or arbitray faults, our protocol was designed to reduce the traffic required by group membership in communication protocols with both static and dynamic scheduling. Therefore, the SPIDER group membership protocol tolerates more severe failures, but it depends strongly on the ROBUS communication system and requires that the number of communication channels be at least three. On the other hand, whereas the SPIDER protocol requires each ROBUS processor, including the star hubs, to broadcast the presumed state of all other processors periodically, ours requires only two bits per processor per cycle in the quiescent state. Thus the remaining bandwidth is available for application traffic.

7. Conclusion

We presented a new group membership protocol specially designed for next generation communication systems intended to support real-time safety-critical applications. By taking advantage of static and dynamic communication scheduling supported by these TDMA-based protocols, it has an overhead of only two bits per processor per cycle in a quiescent state, i.e. when there are no group membership changes. In addition, it tolerates benign failures of up to half of the group members between consecutive executions. Even when there are group membership changes, the overhead of the protocol is lower than that of other protocols that provide the same level of fault tolerance. The protocol is also very responsive, removing faulty processors no later than two TDMA cycles after they fail, and (re)integrating them no later than two TDMA cycles after they decide to join. These numbers compare favorably with protocols designed for systems based on TDMA with static scheduling only: the protocol is as fault tolerant as the most fault tolerant protocols [10], and has a slightly higher overhead than the most efficient [7], which assumes only one fault per TDMA cycle.

To show the feasibility of the group membership protocol, we have also outlined an implementation on top of the FlexRay protocol. We have found no major difficulty. However, the protection against the *babbling idiot* fault in the dynamic segment might be useful.

One of the main concerns in our design was to keep the overhead of the protocol as low as possible, so as not to affect the timeliness of the system. This required a strengthening of the fault assumptions. We plan to carry out a reliability study of this protocol to better assess the worthiness of this design decision, and to analyse the coverage of our failure assumptions.

Acknowledgments

The authors would like to thank to Joaquim Ferreira for his review of a previous version of this paper, and the anonymous referees for their comments and suggestions.

References

[1] F. Cristian, "Reaching Agreement on Processor-Group Membership in Synchronous Distributed Systems", *Distributed Computing*, vol. 4, no. 4, pp. 175–188, 1991.

[2] J. M. Rushby, "Bus Architectures for Safety-Critical Embedded Systems", in *Proceedings of the 1st International Workshop on Embedded Software*, 2001, pp. 306–323.

[3] *FlexRay Communications System Protocol Specification Version 2.1*, FlexRay Consortium, 2005.

[4] L. Almeida, P. Pedreiras, and J. Fonseca, "The FTT-CAN protocol: why and how", *IEEE Transactions on Industrial Electronics*, vol. 49, pp. 1189– 1201, 2002.

[5] J. Ferreira, L. Almeida, J. Fonseca, G. Rodriguez-Navas, and J. Proenza, "Enforcing Consistency of Communication Requirements Updates in FTT-CAN", in *Proceedings of Workshop on Dependable Embedded Systems*, October 2003, Florence, Italy.

[6] H. Kopetz, G. Grünsteidl, and J. Reisinger, "Fault-Tolerant Membership Service in a Distributed Real-Time System", in *IFIP WG10.4 Int'l Working Conference on Dependable Computing for Critical Applications*, August 1989, pp. 167–174.

[7] K. H. Kim, H. Kopetz, K. Mori, E. H. Shokri, and G. Grünsteidl, "An efficient decentralized approach to processor-group membership maintenance in real-time LAN systems: the PRHB/ED scheme", in *Proceedings of the 11th Symposium on Reliable Distributed Systems*, 1992.

[8] H. Kopetz and G. Bauer, "The Time-Triggered Architecture", *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, Oct. 2001.

[9] S. Katz, P. Lincoln, and J. M. Rushby, "Low-Overhead Time-Triggered Group Membership", in *Proceedings of the 11th International Workshop on Distributed Algorithms*, 1997, pp. 155–169.

[10] P. D. Ezhilchelvan and R. de Lemos, "A robust group membership algorithm for distributed real-time systems", *Proceedings of the 11th Real-Time Systems Symposium*, , pp. 173 – 179, 1990.

[11] A. Albert, "Comparison of event-triggered and time-triggered concepts with regards to distributed control systems", in *Embedded World Conf*, 2004, Germany.

[12] N. A. Lynch, *Distributed Algorithms*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[13] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: a tutorial", *ACM Comput. Surv.*, vol. 22, no. 4, pp. 299–319, 1990.

[14] L. Lamport, "Using Time Instead of Timeout for Fault-Tolerant Distributed Systems.", *ACM Trans. Program. Lang. Syst.*, vol. 6, no. 2, pp. 254–280, 1984.

[15] G. V. Chockler, I. Keidar, and R. Vitenberg, "Group communication specifications: a comprehensive study", *ACM Comput. Surv.*, vol. 33, no. 4, pp. 427–469, 2001.

[16] G. Bauer, H. Kopetz, and W. Steiner, "The Central Guardian Approach to Enforce Fault Isolation in the Timed-Triggered Architecture", in *Proceedings of the 6th International Symposium on Autonomous Decentralized Systems*, 2003, pp. 37–44.

[17] C. J. Walter, P. Lincoln, and N. Suri, "Formally Verified On-Line Diagnosis", *IEEE Transactions on Software Engineering*, vol. 23, no. 11, pp. 684–721, 1997.

[18] A. Geser and P. S. Miner, "A New On-Line Diagnosis Protocol for the SPIDER Family of Byzantine Fault Tolerant Architectures", Technical Report NIA 2003-07/ NASA TM-2004-212432, National Institute of Aerospace and NASA, 2004.

[19] P. Thambidurai and Y.-K. Park, "Interactive consistency with multiple failure modes", in *Proceedings of the 7th Symposium on Reliable Distributed Systems*, 1988, pp. 93–100.