Appropriate consistent replicated voting for increased reliability in a node replication scheme over FTT

Sinisa Derasevic, Manuel Barranco, Julián Proenza
DMI, Universitat de les Illes Balears, Spain
sinishadj@gmail.com, manuel.barranco@uib.es, julian.proenza@uib.es

***Abstract*—In the context of critical applications there is an increasing interest in having Distributed Embedded Systems (DESs) that are able to operate in dynamic environments, while at the same time reaching a high reliability. The Flexible Time-Triggered communication paradigm (FTT) is designed to support the QoS and real-time requirements of the traffic of these systems. However, FTT does not provide fault tolerance. This paper explains our on-going work towards designing a consistent and highly-reliable voting protocol which supports node replication on DESs that use FTT switched Ethernet. In particular, we propose a protocol for the node replicas to vote consistently on messages exchanged through an FTT Ethernet network that uses time redundancy, while trying to maximize the reliability that can be achieved thanks to the redundancy of the nodes and the communication subsystem itself.**

I. Introduction

This work is a part of the ongoing FT4FTT project. The aim of the project is to provide fault tolerance mechanisms for adaptive DES that use the Flexible Time Triggered (FTT) communication paradigm [1] in order to increase reliability, i.e. the ability of the system to provide continuous operation.

A widely accepted technique for dealing with permanent faults in the nodes is node replication. Our design uses *active replication* [2] where each node replica executes the same code, thus tolerating hardware faults in the nodes, but not software ones. Yet, we will generally follow the terminology of N-Version Programming (NVP) [3], despite NVP assuming replicas with design diversity. Each replica, after some partial execution, called *segment*, produces an output, called *cc-vector*. Cc-vectors are then exchanged among replicas and the voting procedure in each node replica obtains the result, called *consensus cc-vector*, which each node uses in the computations of the next segment.

The voting procedure in each node replica needs to have the same input values (cc-vectors), so each voting produces the same result. The problem of providing the same input values to all replicas is called *external replica determinism enforcement* [4]. One possibility to achieve external replica determinism for cc-vectors in FTT is to enforce *full consistency* when exchanging them, i.e. to enforce that each cc-vector sent from one of the replicas to the rest, either is received by all of them or by none. A protocol well suited for this purpose is the one proposed in [5].

However, although external replica determinism could be achieved in that way, to require full consistency is too restrictive, since for voting it is enough that a majority number of replicas have consistently exchanged a sufficient number of cc-vectors. In fact, to require full consistency can unnecessarily hinder the voting procedure and even provoke a quick attrition of the available node redundancy. For instance, a fault preventing a single replica from receiving cc-vectors will be enough to impede the completion of the voting from then on, as full consistency will not be achieved any longer for any cc-vector. Moreover, since [5] was not specifically designed to support a voting mechanism, it presents some features that makes it more difficult to fully exploit node redundancy in this context.

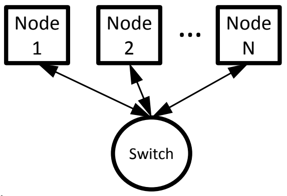


Fig. 1. System Architecture.

Thus, the objective of this paper is to propose a protocol for voting in FTT that fully exploits the available node redundancy thus increasing the system reliability. This work takes [5] as a starting point and, then, modifies and extends it in several aspects. First, it changes [5] to improve the reliability with which messages are exchanged. Second, it eliminates the requirement of achieving full consistency for accepting a message (cc-vector) for voting. Third, it proposes what we call the *Voting Set-Up Algorithm* (VSUA), i.e. an algorithm for the replicas to consistently decide which cc-vectors should be accepted and which replicas should vote using them.

The paper emphasizes the criteria that VSUA uses to select the cc-vectors and to select the voting replicas, and the way in which it finds an appropriate trade-off between these two criteria. This is an important issue as it is not always possible to simultaneously maximize both, the number of cc-vectors and replicas that vote on them. On the one hand, to maximize the number of cc-vectors used for voting increases the quality of the voting itself. On the other hand, to maximize the number of replicas is a key point for preventing redundancy attrition, as only the replicas that vote in a given segment can continue voting in the subsequent one.

Section II presents the architecture of the system, its fault model, its fault tolerance mechanisms, and the protocol [5]. Section III proposes a sub(protocol) that modifies and extends [5], and describes the algorithm for choosing the cc-vectors and the replicas for voting. Finally, Section IV concludes the paper and points out future work.

II. System & initial consistency protocol basics

A. System architecture

The system consists of N nodes, some of which are replicas of the same node, interconnected by a switched Ethernet network (Fig. 1). The protocol used is a version of FTT [1] called *Hard Real Time Ethernet Switch* (HaRTES) [6]. FTT is a master/multi-slave protocol where a single node, called *FTT Master*, controls the communication of multiple slaves. In HaRTES the master is integrated within the switch.

The communication is divided into fixed-size cycles called *Elementary Cycles* (ECs). Each cycle starts by the FTT Master broadcasting a special control message called *Trigger Message* (TM) which conveys information about which messages should be transmitted, and by whom in the current EC. The remainder of the EC is divided into two consecutive windows. The first one, called synchronous window, is used for the transmission of periodic messages, and the second one, called asynchronous window, is used for the transmission of aperiodic ones.

B. Fault model and fault tolerance mechanisms

Although the HaRTES switch already includes some error-containment mechanisms [7], in the context of the FT4FTT project the switch and the rest of the system is provided with additional mechanisms to tolerate permanent and transient faults in the nodes and in the network.

Nodes can fail in arbitrary manners. To prevent error propagation from a faulty node to the rest of the system an enhanced HaRTES switch is used. The enhanced switch contains the errors coming from the nodes by policing the traffic coming to its ports and dropping any message it deems incorrect. This enforces incorrect computational failure semantics of each node from the point of view of the other nodes as needed by the voting procedure. Since this switch is crucial in our system, it is enforced to exhibit restricted (crash) failure semantics by duplicating and comparing its internal circuitry.

We have designed FT4FTT for tolerating both permanent and transient node faults by means of active node replication [2] with majority voting, which corresponds to an *error compensation* approach. Permanent faults affecting the network itself, i.e. the switch and the links, are tolerated by replicating the switch and the links as a part of an ongoing work [8]. Finally, in order to tolerate transient faults affecting the links, in FT4FTT the unused bandwidth is exploited for temporal redundancy, i.e. critical messages are sent multiple times in the same (or in different ECs) in order to increase the probability with which they are eventually received.

C. Initial consistency protocol

As said in Section I, we extended and modified the consistency protocol proposed in [5] to make it suitable for the voting protocol proposed here. Thus the basic aspects of [5] are explained next.

[5] is based on the publisher-subscriber model and consists of 4 phases, whose position within the EC can be seen in Fig. 2. The *Schedule* phase starts with the FTT master broadcasting a TM containing the traffic schedule, which is the list of messages that the publishers have to transmit in the next synchronous window. Next, in the *Broadcast* phase, each publisher transmits in the synchronous window the messages indicated by the TM. In the *Acknowledge* phase each subscriber sends in the asynchronous window a notification (ACK) for each message that it has correctly received. Finally, in the *Accept* phase, each message is either accepted or rejected in a consistent manner depending on whether or not *all* the ACKs for that message have been received or not.

This last phase has two critical points. The first one, called *Accept Point*, occurs at the end of the asynchronous window. At that point the FTT master has all the information, i.e. all the ACKs, and makes a decision on whether each specific message must be accepted or rejected. Then, it generates a vector, the EC-*Status* vector, that includes the information about each

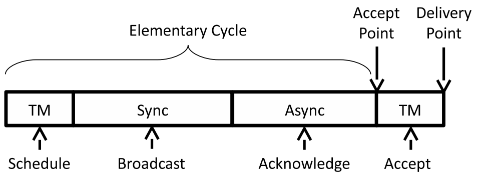


Fig. 2. EC Schedule.

message delivery. Specifically, this vector includes 1 bit per message indicating whether that message must be accepted (the bit is set to true), or it must be rejected (the bit is set to false). This vector is a part of the TM and is populated by the FTT master at the beginning of the incoming EC. The second point, called *Delivery Point*, happens after the TM contained within the EC-Status vector has been received and decoded by all of the slaves. The TM is sent multiple times at the beginning of the EC to ensure that all nodes receive it.

In order to make this consistency protocol resilient to the lost of messages and ACKs, a retransmission of each message that has not been acknowledged is scheduled by the FTT master, at least, in the next EC [5].

III. Consistent majority voting protocol

A. Cc-vector exchange protocol

The (sub)protocol we propose for exchanging cc-vectors modifies and extends [5]. The modifications are devoted to improve the reliability with which those messages are exchanged.

a) [5] does not specify how many times an inconsistent message should be retransmitted. Thus, we propose to dedicate an *exchange round*, constituted by k ECs, to try to consistently exchange the cc-vectors. In each one of the round's ECs the switch orders the retransmission of the cc-vectors (which then triggers the retransmission of the ACKs) that have not been consistently received yet. Moreover, each replica sends n copies of each cc-vector it is requested to re/transmit. Also, in case of using a duplicated star topology (as proposed in [8]), we advocate increasing reliability further by forcing replicas and switches to send each copy of every message through each replicated link. Finally, to tolerate transient faults and distinguish them from permanent ones, k and n must be selected to ensure that a message is not consistently exchanged in a given round only if a permanent fault occurred.

b) [5] does not either specify who should retransmit the messages. We consider that to rely on the publishers for retransmitting is not reliable, since a faulty publisher could retransmit a message different from the one that is expected, thereby violating replica determinism. In contrast, the switch cannot carry out incorrect retransmissions as it presents crash failure semantics. Thus, we propose that the switch retransmits any message it has correctly received and that has not been consistently acknowledged. The only messages that publishers should retransmit are those that the switch has never received. Note that if a publisher retransmits an incorrect version of a message the switch never received, replica determinism will not be violated as neither the switch nor any of the subscribers would have received the original message.

c) [5] specifies that, in each (re)transmission trial of a message, the subscribers that correctly receive the message have to discard it if the message is not consistently received by all the subscribers. This has the disadvantage of decreasing

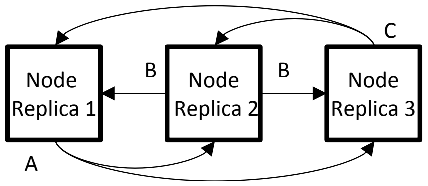


Fig. 3. Message exchange between 3 replicas.

the chances of a message being eventually received by all subscribers, as correct but inconsistent receptions are wasted in each trial. Thus, we propose that the subscribers keep during all the ECs of the round each message they correctly receive.

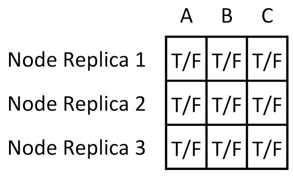
d) Finally, [5] establishes that a message can only be accepted if all replicas have consistently acknowledged it (full consistency). Nevertheless, as explained in Section I, to apply this strategy for deciding when a cc-vector must be accepted makes the voting mechanism very sensible to transient or permanent faults. Thus, we propose to change the acceptance criterion for whether a cc-vector is accepted.

Since this criterion is part of the *Voting Set-Up Algorithm* (VSUA), it will be explained in the next section. However, it is important to highlight here that to support this algorithm we needed to substitute the EC-Status vector of [5] by what we call the *Messages Status* vector (MS-vector). Since the VSUA allows voting as long as a majority of cc-vectors has been consistently exchanged by a majority of replicas, the MS-vector includes information about which replicas sent its cc-vector and which replicas acknowledged each one of them.

The master populates the MS-vector and includes it in the TM. For instance, consider the system constituted by 3 replicas depicted in Fig. 3, in which each replica sends a cc-vector to the others. Fig. 4 shows a matrix that represents the content of the corresponding MS-vector. The first row specifies if replica 1 has sent its message, A, and received the other two, B and C. More specifically, the first cell $(1, A)$ is set to *true* if the switch has received message A sent from replica 1. The other cells of this row are respectively set to true if (1) replica 1 has sent ACKs for messages B and C previously received by the switch from replica 2 and 3, and (2) the switch has received these ACKs. The meaning of the other rows is analogous.

Although nodes can fail in arbitrary manners, the switch drops all messages that are incorrect in order to enforce incorrect computational failure semantics for the nodes. In particular, the faults that affect the messages used to build up the MS-vector are treated as follows. An out-of-time cc-vector/ACK will be dropped by the switch and will hence manifest as a *false* within the corresponding cell/s of the MS-vector. A cc-vector conveying an erroneous value will be tolerated by the voting mechanism itself. An ACK with an incorrect value will normally manifest as a false within the MS-vector. Nevertheless, if the ACK's content refers to a wrong but existing publisher, it will manifest as a false *true* (impersonation) in the vector. This last case can be tolerated by forcing subscribers to piggyback within the ACK some kind of redundant code based on the content of the original cc-vector, so that the switch can check the correspondence between the ACK and the original message.

Anyway, a false within the MS-vector compels the FTT-master to schedule in the next round's EC the retransmission of



	A	B	C
Node Replica 1	T/F	T/F	T/F
Node Replica 2	T/F	T/F	T/F
Node Replica 3	T/F	T/F	T/F

Fig. 4. Messages Status vector.

the corresponding cc-vector. Note that a *false* within the MS-vector can change to *true* after a successful retransmission; whereas a value *true* remains *true* once it is set.

B. Voting Set-Up Algorithm (VSUA)

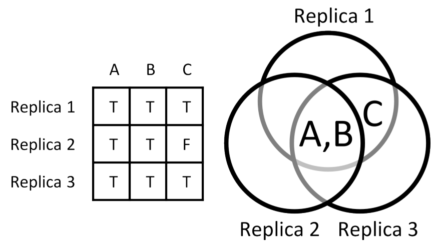
The Voting Set-Up Algorithm (VSUA) is devoted to deciding which cc-vectors should be used for voting and which replicas should vote on them. On the one hand, it must provide the voting of the current segment with at least a majority of cc-vectors. On the other hand, it has to ensure that the voting of the current segment takes place in at least a majority of node replicas, which will thereby be able to produce sufficient number of inputs (cc-vectors) for the voting of the next segment. Note that $\textit{majority} = \lfloor X/2 \rfloor + 1$, where X is the number of cc-vectors/nodes replicas. Recovery of nodes once they are excluded from voting is still an on-going work.

Transient faults that provoke that messages are not transmitted/received are tolerated by the spatial and temporal redundancy explained in Section III-A. Thus, the VSUA is designed to tolerate transient faults that provoke that a node produces an erroneous cc-vector value for the whole round and, also, permanent faults that provoke so or that prevent messages from being transmitted/received. The guaranteed number of faults the VSUA can tolerate is $X - \textit{majority}$, but in some particular cases, as will be shown later, even more. The VSUA is designed for any number of replicas. However, for the sake of clarity, in this section we illustrate it for just 3; which is in fact the number of replicas in most highly-reliable systems due to cost restrictions.

Recall that any false in the MS-vector can only be the consequence of a permanent fault preventing a replica from re/transmitting its cc-vector or ACK. Also note that the VSUA assumes that no more than one permanent fault occurs during a round of the cc-vector exchange protocol.

The criteria for selecting the cc-vectors and replicas could be to maximize the number of both of them. However, as already said, there can be a conflict when trying to do so and, thus, the VSUA must find a compromise. To illustrate this conflict consider that the final MS-vector at the end of a round in the system of Fig. 3 is as shown in Fig. 5. Replica 2 has not acknowledged the cc-vector sent from replica 3, C. In such a case the VSUA has two options. The first one is to maximize the number of cc-vectors and, then, use the three messages (A, B, C) for voting in replicas 1 and 3. The second one is to maximize the number of replicas, so that all replicas should vote using the messages A and B.

The key point to choose the most reliable option is to elucidate which replica is affected by the permanent fault, 3 or 2. To diagnose this, note that replica 3 succeeded in transmitting C to the switch, i.e. $(3, C) = \textit{true}$. Thus, even if replica 3 had permanently failed after the switch received C, replica 2 should have acknowledged C, as the switch is the responsible for retransmitting any message it receives and,



	A	B	C
Replica 1	T	T	T
Replica 2	T	T	F
Replica 3	T	T	T

Fig. 5. An example of Messages Status Vector.

hence, from the point of view of replica 2, the switch has enforced that C has been retransmitted $k \cdot n$ times. Therefore, replica 2 is the one affected by the permanent fault.

In fact, independently of the number of replicas, N, it can be concluded that the VSUA has to prioritize the number of cc-vectors over the number of replicas. This is because if a subscriber does not acknowledge a cc-vector the switch has successfully received, then the subscriber is certainly affected by a permanent fault and it should be excluded from the voting. Moreover, as shown in the example, this criteria allows tolerating more than the guaranteed limit of number of faults. Specifically, the VSUA tolerates 1 permanent fault (node 2) and can tolerate an additional transient fault that corrupts the value of a cc-vector since it uses 3 cc-vectors to vote.

This criteria is reflected in Algorithm 1, which outlines the pseudo JAVA code of the VSUA. First, the MS-vector is populated by random values (line 2). Then if some cell in the diagonal, $(\mathrm{i}, \mathit{Message}_i)$, is false the VSUA populates the whole column $\mathit{Message}_i$ with false values (line 3), since no subscriber can acknowledge a cc-vector the switch has not correctly received. Then, the VSUA inspects whether a majority of cc-vectors have been consistently acknowledged by a majority of replicas, i.e. it checks if a majority of columns are set to true in a majority of rows (lines 4-15). In order to prioritize cc-vectors over replicas, the VSUA starts by considering that the majority of cc-vectors equals the total number of cc-vectors, $M = S$ (line 1). Then, if it does not find a majority of replicas for this majority of cc-vectors, it decreases by one the number of cc-vectors that are considered as a majority, i.e. $M - 1$ (line 4). The VSUA does this iteratively until it finds a majority of replicas that has consistently acknowledged the majority of cc-vectors, or until the number of cc-vectors that is considered as a majority is $M < \lfloor S/2 \rfloor + 1$ (line 4). Note that for a given majority of cc-vectors there are several permutations of cc-vectors that could be consistently acknowledged by a majority of replicas, e.g. with $N = 3$ and $M = 2$ the possible permutations of majority cc-vectors are AB, AC and BC. However, as only one permanent fault can occur per round, the replicas are able to consistently acknowledge the cc-vectors of just one permutation, i.e. the one that VSUA eventually finds.

IV. Conclusion and Future Work

Although FTT switched Ethernet is specially well suited to support QoS and real-time flexible communication in DESs in dynamic environments, it lacks of fault tolerance mechanisms. This paper presents our work towards providing a highly-reliable voting protocol supporting node replication in FTT switched Ethernet. It includes a (sub)protocol for the replicas to exchange the messages that are needed for voting, and which is more reliable than an approach previously proposed for consistently exchanging messages in FTT. On the other hand, it includes an algorithm that does not require full consistency, but that allows replicas to vote as long as a majority of them has consistently exchanged a majority of the messages needed for voting. This strategy allows to better exploit the available spatial and time redundancy.

Algorithm 1: Pseudo JAVA code for simulating the VSUA

```
1  M = S = N = number of messages = number of nodes;
2  msgStVect[N][S] = populateRandValues();
3  inpectAndPopulateDiagonalFalse(msgStVect);
4  while M-- ≥ S/2+1 do
5      nodeList = empty;
6      for i=0; i<N; i++ do
7          nodeHasAllMessages=true;
8          for j=0; j< S; j++ do
9              if not msgStVect[i][j] then
10                 nodeHasAllMessages=false;
11                 break;
12         if nodeHasAllMessages then
13             nodeList.add(i);
14     if size(nodeList) ≥ N/2+1 then
15         Exit and vote;
```

The paper explains the algorithm for 3 replicas, but was implemented in JAVA and successfully simulated for up to 7 nodes and an arbitrary number of faults. We plan to formally verify the correctness of the algorithm and quantitatively evaluate the system reliability that can be achieved when using node replication and the proposed protocol.

V. Acknowledgements

This work was supported by project DPI2011-22992 (Spanish *Ministerio de Economía y Competividad*), by FEDER funding, and by the Portuguese government through FCT grant Serv-CPS PTDC/EEA-AUT/122362/2010. Sinisa Derasevic was supported by a scholarship of the EUROWEB Project (http://www.mrtc.mdh.se/euroweb), which is funded by the Erasmus Mundus Action II programme of the European Commission.

References

[1] P. Pedreiras and A. L, "The Flexible Time-Triggered (FTT) paradigm: An approach to qos management in distributed real-time systems," in *IPDPS, Proceedings.* IEEE, 2003.

[2] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding replication in databases and distributed systems," in *Distributed Computing Sys., Proc. 20th Int. Conf.*, 2000, pp. 464–474.

[3] A. Avizienis *et al.*, "The n-version approach to fault-tolerant software," *IEEE Trans. Software Eng.*, vol. 11, no. 12, pp. 1491–1501, 1985.

[4] S. Poledna, *Fault-tolerant real-time systems: The problem of replica determinism.* Springer, 1996.

[5] G. Rodriguez-Navas and J. Proenza, "A proposal for flexible, real-time and consistent multicast in FTT/HaRTES switched ethernet," in *Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference.*

[6] R. Santos, "Enhanced Ethernet Switching Tecnology for Adaptibe Hard Real-Time Applications," Ph.D. dissertation, Universidade Aveiro, 2010.

[7] A. Ballesteros, D. Gessner, J. Proenza, M. Barranco, and P. Pedreiras, "Towards preventing error propagation in a real-time ethernet switch," in *Emerging Technologies Factory Automation, 2013 IEEE 18th Conf.*

[8] D. Gessner, J. Proenza, M. Barranco, and L. Almeida, "Towards a flexible time-triggered replicated star for ethernet," in *Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference on*, 2013.