Towards a Flexible Time-Triggered Replicated Star for Ethernet

David Gessner, Julián Proenza, Manuel Barranco*
DMI, Universitat de les Illes Balears, Spain
davidges@gmail.com

Luís Almeida
IT, Universidade do Porto, Portugal
lda@fe.up.pt

Abstract

Distributed embedded systems have traditionally been designed using static approaches, i.e., assuming a static environment. Such approaches, however, cannot guarantee continuous operation under dynamic environments that impose new requirements upon a system as time passes. As a solution, flexible approaches have been proposed. One such approach that allows a system to adapt to changing real-time requirements is the Flexible Time-Triggered (FTT) communication paradigm. Nevertheless, if continuous operation under dynamic environments is desired, then flexibility is not enough. Indeed, it is also crucial for the system to be sufficiently reliable. In this paper we therefore explore some design ideas to make FTT highly reliable through fault tolerance by using replication. As a starting point we will use the switch of the Hard Real-Time Ethernet Switching (HaRTES) implementation of FTT.

1. Introduction

Traditionally, distributed embedded systems (DES) have been designed to operate in environments that do not change over time. This has led to static approaches that are inadequate for continuous and correct operation under dynamic environments. The alternative are flexible approaches. However, flexibility alone is not enough to guarantee continuous operation: reliability is also essential.

The goal of the project titled *Fault Tolerance for Flexible Time-Triggered communication* (FT4FTT) is to demonstrate that it is possible to build a highly reliable DES that can change its real-time operation upon changing requirements imposed by a dynamic environment. For this it takes as a basis a master/multi-slave communication paradigm known as Flexible Time-Triggered communication (FTT) [9], which is a bandwidth efficient approach to achieve flexibility with high reactivity.

Previous work has been done to increase FTT's reliability. However, it focused on FTT master replication without using any channel replication [7] or using replicated buses only [4], even though buses have significant disadvantages in terms of reliability when compared with replicated star topologies [2]. Moreover, these approaches were partial in the sense that they only focused on the communication channel. Having a reliable channel alone, however, does not necessarily improve the reliability of the whole system by much. In fact, it has been shown that without being able to tolerate the failures of nodes, introducing channel redundancy in an attempt to improve the system's reliability, at least in the case of replicated stars, only improves the overall reliability to a small degree [3]. The FT4FTT project therefore focuses on the whole system, including both the communication channel and the nodes.

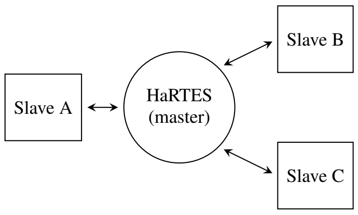

Figure 1. HaRTES architecture.

In this paper we present some design ideas for FTTRS, a *Flexible Time-Triggered Replicated Star*, which will provide the communication channel for FT4FTT.

FTTRS uses as its basis the switch of the Hard Real-Time Ethernet Switching (HaRTES) architecture, which is an implementation of FTT for Ethernet [6]. This switch allows the communication to be flexible, but it has not been designed with high reliability in mind, such as it is required in highly reliable DES. To make a system more reliable, a classic approach is to use fault tolerance. FTTRS will therefore replicate the HaRTES switch, which includes the FTT master it embeds, to increase the reliability of the communication channel. This paper describes some ideas on how to do this replication.

The paper is organized as follows. Section 2 gives an overview of HaRTES; Section 3 presents the basic architecture for FTTRS; sections 4 and 5 discuss some specific details of channel and master replication, respectively; and Section 6 gives the conclusions and mentions future work.

2. HaRTES: an overview

As shown in Figure 1, HaRTES implements a simplex (not replicated) microsegmented star topology, with the

*©2013 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. doi:10.1109/ETFA.2013.6648137

HaRTES switch as a central element that provides the most relevant functions of FTT. In particular, the switch embeds an FTT master. This master grants access to the network following a centralized master/multi-slave scheme. This means that a single message from the master triggers the transmission of messages in several slaves. Specifically, the master divides the communication time into rounds called *Elementary Cycles* (ECs), which are synchronized among and have the same constant duration in all simultaneous communications that may occur across the switch. The ECs are comprised of a *synchronous window* followed by an *asynchronous window*. The FTT master initiates each EC with the transmission of a *Trigger Message* (TM) that is flooded to all slaves. This message not only marks the beginning of a new EC, inciting the slaves to transmit their synchronous messages followed by their asynchronous messages, but it also dictates the schedule for the next synchronous window, i.e., it tells the slaves which synchronous messages they should transmit during that window. The schedule is calculated by the master based on the contents of a *System Requirements Database* (SRDB). This database specifies the communication requirements for different message streams, which are sequences of messages related to the same entity (e.g., a sequence of readings of the same sensor) and are analogous to a task in processor scheduling. Example requirements are deadlines and periods, which are both expressed as multiples of the EC length. Slaves may request changes to the SRDB, but these requests are subject to an online admission control performed by the master. The admission control basically ensures that the SRDB is only updated with the requested change if the system will still be schedulable afterwards.

Regarding the slaves of the protocol, they are each connected to the switch by means of full-duplex links. This allows multiple simultaneous communications: while one set of slaves is intercommunicating, another disjoint set of slaves may intercommunicate at the same time.

3. FTTRS architecture

As stated in the introduction, our goal is to use fault-tolerance to obtain highly reliable communication for FT4FTT, while maintaining the flexibility provided by the FTT communication paradigm.

To achieve high reliability, we need to eliminate all single points of failure. This is achieved through some form of redundancy.

In the case of HaRTES, the single points of failure are the switch and the master it embeds: if they fail, the whole system will suffer a global failure. This is in contrast to the slaves and the links. Their failure must not necessarily imply a global failure because they might not be critical for the given application, or, if they are, the node replication provided by FT4FTT may tolerate their failure. In any case, the architecture for FTTRS must replicate the switch and the master. Furthermore, since it can not be generally assumed that there is failure independence between a switch and the master embedded in that switch, the master replicas should not be within the same switch, but be distributed among the switch replicas. Additionally, for the slaves to be able to make use of both switch replicas, we also require links from each switch to the slaves. Regarding the number of replicas, duplication should already grant a significant increase in reliability without the often prohibitive cost of higher levels of replication (to know for certain, a quantitative reliability evaluation is required). Finally, to tolerate the failure of one of the replicas, any replication scheme needs to ensure that the remaining replicas can continue to provide a correct service and that the transition to a state with less correct replicas does not interfere with the correct functioning of the whole system. For this, it is necessary to ensure replica determinism [10]. Replica determinism usually requires some form of communication between the replicas. The only exception is under the assumption that (i) the replicas behave internally in a completely deterministic way whenever they receive the same inputs and (ii) their inputs are guaranteed under all circumstances to be the same. Since the switch and master replicas receive their inputs through different links, we do not want to depend on this assumption. We therefore require a communication path between the replicas. Specifically, for FTTRS, this means that we need a communication path between the switches and between their embedded masters. Moreover, this communication path is replicated as well to further increase the reliability of FTTRS.

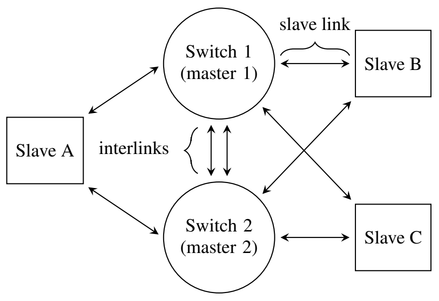

Figure 2. FTTRS architecture.

The above discussion leads us to the switch and master replication scheme shown in Figure 2. The replicated communication subsystem comprises two HaRTES switches, each with its own master. The slaves are connected to both switches by means of *slave links* and there are several redundant links between the switches called *interlinks*. The latter serve as a communication channel between the masters, which is used to provide them with replica determinism. Furthermore, if the interlinks are also used to forward slave messages from one switch to another, we benefit from a further redundant path between any pair of slaves, reducing the probability of the slaves being divided into subsets that cannot intercommunicate [8].

Another important point to consider when replication is used is failure independence among replicas. For this, it is necessary to prevent error propagation, i.e., it is necessary to prevent an error generated in one replica to cause new errors in other replicas.

To prevent the propagation of errors between replicas, they need to be prepared to cope with the errors generated in other replicas. The difficulty of this depends on how the errors generated can manifest i.e., it depends on the failure semantics of the replicas. In the most general case, replicas may present byzantine behavior, which is the most difficult to deal with. Error propagation can be prevented much more easily if we restrict the failure semantics of the replicas. For this reason, we ensure that the switch and the master replicas have more benign failure semantics. Specifically, we ensure that they have crash failure semantics, i.e., they either function correctly or they are silent. This can be achieved by using an internal duplication and comparison mechanism. Both the switches and the masters are thus prevented from generating spurious frames.

Note that ensuring crash failure semantics for the switch and master replicas does not ensure failure independence between a given switch and the master it contains. This is acceptable since we view a switch/master combination as a single entity whose components in any case depend on common resources such as a common power source.

Finally, there is one design decision of the architecture related to the service that FTTRS can provide to the node replication in FT4FTT. Since FT4FTT will use node replication, it will also be necessary in FT4FTT to provide failure independence among slave replicas. This can be simplified by restricting the failure semantics locally at each of the slaves. However, this would usually imply complicating the internal design of the slaves. Another solution is to add error containment capabilities to the ports of the central elements of a star topology [11]. We take this approach for FTTRS. In other words, the switches of FTTRS provide error containment mechanisms at their ports to prevent certain errors from spreading from one link to another. Part of this error containment can be provided by taking advantage of the Frame Check Sequence (FCS) in the Ethernet frames transmitted over the links. For this, the FTTRS switches drop any frame that fails the FCS test. That is, the switches in FTTRS are store-and-forward as opposed to cut-through. In addition to this Ethernet-generic error containment, the switches have FTT-specific error containment capabilities. Specifically, any unexpected frame received at a switch port is dropped. This means that unscheduled synchronous messages are dropped, as well as duplicate synchronous messages received within the same EC.

4. Channel replication

Next, we give some specific design ideas on the channel replication in FTTRS: we discuss how the channel replication can be used for the TM and for synchronous messages. Asynchronous messages are out of the scope of this paper.

Concerning the TM, both slaves and masters must have a common time base for the FTT communication to function correctly. Since the TM is used as a means to synchronize the nodes, its transmission must satisfy the following requirements. (**R1**) The TM of each EC must be received by all slaves that have at least one non-faulty link attached to a non-faulty switch. (**R2**) The TM must not be delayed by more than a pre-specified amount of time. (**R3**) All correct slaves and masters must agree on when each EC starts.

To satisfy R1 there are several possibilities. One option is to use a total-order broadcast protocol [5]. However, such an approach may make it difficult to also satisfy R2 because such protocols usually have a non-negligible time overhead. Another approach is to use both spatial and temporal redundancy for every TM such that at the beginning of each EC the TM is flooded through both switches and this is done more than once.

As to R3, if R1 and R2 hold, the masters can agree on when an EC starts by enforcing replica determinism for them (see Section 5). The slaves, on the other hand, can agree on when an EC starts, independently of which replica of the TM they receive, by, for instance, distinguishing TM replicas using sequence numbers, which would be reset in each EC, and setting a timer at each receiving slave. The timeout would be set according to the sequence number in the last TM received and according to the propagation delay from the master to the corresponding slave. When it expires, the slave would consider the EC to start. With such a mechanism the expiration times of the timers in all nodes could be made to line up such that they would all consider the EC to start at the same time.

Regarding synchronous messages from slaves, the replicated channel should also be used to tolerate faults affecting them. Two possible options are the following: (i) transmit a given message through only one of the links and, if this permanently fails, use the other, or (ii) always send each message through both links. The second option might lead receiving stations to receive duplicate frames in a given EC. However, they can safely discard the second copy because in FTT the master will never schedule two identical frames within the same EC (an EC is atomic from the scheduler's point of view).

The advantage of the first option is that if a given message is only transmitted through one link, then another message can simultaneously be transmitted through the other link, increasing the rate with which different messages can be transmitted. However, such an approach would generally take longer to recover from errors than the second option. This is so because in the second option a retransmission might only be necessary if both simultaneously transmitted copies of a frame are corrupted; whereas in the first approach the corruption of the single copy might already require a retransmission. On the other hand, the second option would not allow an increase in the channel capacity.

In either case, each switch could forward received frames to the other using the interlinks to increase the likelihood of the frames reaching all slaves—bandwidth constraints could be alleviated by using higher bandwidth links for the interlinks. Whether a given switch must then forward the frames to the locally attached slaves depends on whether those slaves already received the frames through the other switch. Only if not, the forwarding is necessary. However, a given switch cannot check by itself whether an attached slave already received a given frame. Only the respective slave can know that for sure. Thus, the easiest solution seems to be letting the switches always forward all frames received through the interlinks and let the slaves handle any duplicate frames.

5. FTT master replication

To tolerate the failure of one of the masters, they need to be replica determinate [10]—otherwise inconsistencies between them may lead to the broadcast of conflicting TMs, preventing the correct operation of the FTT protocol.

To enforce replica determinism for any set of components, we need to identify the service they provide. For the FTT masters specifically, the services they provide that must be replica determinate are (**S1**) to dictate when each EC begins and (**S2**) to dictate what synchronous messages should be transmitted in each EC.

S1 is provided by the transmission instants of the TMs and S2 by the schedules transmitted in the TMs. To have the masters replica determinate we therefore need to ensure that they each transmit their TMs at the same time, allowing only a small imprecision, and that they transmit the same schedule in the TMs corresponding to the same EC.

Ensuring that the masters transmit their TMs at the same time can be achieved by executing a clock synchronization protocol using the interlinks. As a basis, the IEEE 1588 protocol [1] could be used.

Regarding the TM schedules, if they differ, this is due to inconsistent SRDBs between the masters when the next schedule is calculated. To determine the cause of these inconsistencies, note the following assumptions regarding the masters. (**A1**) We assume that they have internal replica determinism [10]. This means that the scheduling algorithm and the admission control executed by the masters is deterministic. That is, given the same contents in the SRDB, the scheduler will always produce the same sequence of schedules. Likewise, given the same SRDB contents and the same sequence of update requests, the admission control will always accept the same requests. (**A2**) Second, we assume that at system startup time the SRDBs of the masters are consistent, i.e., that they have identical contents. (**A3**) Third, only SRDB update requests from slaves may lead to changes in the SRDBs. Thus, under assumptions A1-A3, the only cause for SRDB inconsistencies are inconsistent receptions of SRDB update requests.

There are two ways to enforce replica determinism for service S2: ensure that the masters apply the same SRDB updates in the same order before the next schedule is calculated or, alternatively, allow inconsistencies between the SRDBs of the masters temporarily, but ensure that they are resolved before the next schedule is calculated.

For the first option, the problem is basically to enforce total-order multicast [5] for the SRDB update requests, but with real-time constraints. For the second option, enforcing replica determinism becomes the problem of ensuring replicated database consistency with real-time constraints. In both cases the deadline is the time at which the next schedule is calculated.

6. Conclusions and future work

In this paper we presented some first ideas for the design of FTTRS, a Flexible Time-Triggered Replicated Star for Ethernet, which takes as its starting point the HaRTES implementation of the FTT communication paradigm and adds fault tolerance by using replication. We proposed an architecture for FTTRS and then discussed some ideas related to the replication of the channel and the FTT master.

These ideas will now be further developed. For instance, we will explore how to handle asynchronous traffic in order to maximize the reliability of the system. Furthermore, the different design ideas will be evaluated quantitatively in order to estimate the reliability achievable with each, and how their reliability compares with the original HaRTES. This will give us an objective criteria to make a choice among the different design options. The chosen design should then be formally verified, e.g., by using model checking, to ensure its correctness. Finally, a prototype should also be developed in order to prove the feasibility of the design.

Acknowledgements

This work was supported by project DPI2011-22992 and grant BES-2012-052040 (Spanish *Ministerio de economía y competividad*), by FEDER funding, and by the Portuguese government through FCT grant Serv-CPS PTDC/EEA-AUT/122362/2010.

References

[1] IEEE Std 1588-2008, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, 2008.

[2] M. Barranco, J. Proenza, and L. Almeida. Boosting the Robustness of Controller Area Networks: CANcentrate and ReCANcentrate. *Computer*, 42(5):66–73, May 2009.

[3] M. Barranco, J. Proenza, and L. Almeida. Reliability improvement achievable in CAN-based systems by means of the ReCANcentrate replicated star topology. In *8th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 99–108, May 2010.

[4] V. da Silva. *Flexible Redundancy and Bandwidth Management in Fieldbuses*. PhD thesis, Universidade de Aveiro, 2010.

[5] X. Défago, A. Schiper, and P. Urbán. Total Order Broadcast and Multicast Algorithms: Taxonomy And Survey. *ACM Computing Surveys*, 36:2004, 2003.

[6] R. G. V. dos Santos. *Enhanced Ethernet Switching Technology for Adaptive Hard Real-Time Applications*. PhD thesis, Universidade de Aveiro, 2010.

[7] J. Ferreira. *Fault-Tolerance in Flexible Real-Time Communication Systems*. PhD thesis, Univ. de Aveiro, 2005.

[8] D. Gessner, M. Barranco, J. Proenza, and M. Short. A first qualitative evaluation of star replication schemes for FTT-CAN. In *IEEE 17th Int. Conf. on Emerging Technologies & Factory Automation*. IEEE, Sept. 2012.

[9] P. Pedreiras and L. Almeida. The flexible time-triggered (FTT) paradigm: an approach to QoS management in distributed real-time systems. In *Proc. International Parallel and Distributed Processing Symposium*, page 9. IEEE Comput. Soc, 2001.

[10] S. Poledna. *Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism*. Kluwer Academic Publishers, 1996.

[11] J. Proenza, M. Barranco, J. Llodra, and L. Almeida. Using FTT and stars to simplify node replication in CAN-based systems. In *17th IEEE Int. Conf. on Emerging Technologies & Factory Automation*. IEEE, Sept. 2012.