Towards an Experimental Assessment of the Slave Elementary Cycle Synchronization in the Flexible Time-Triggered Replicated Star for Ethernet

David Gessner, Inés Álvarez, Alberto Ballesteros, Manuel Barranco, Julián Proenza
DMI, Universitat de les Illes Balears, Spain
{davidges, ines.alvarez.91}@gmail.com {a.ballesteros, manuel.barranco, julian.proenza}@uib.es

***Abstract*—The communication subsystem of distributed embedded systems (DES) that must operate continuously and satisfy unpredictable requirement changes must be reliable and flexible. Recently the Flexible Time-Triggered Replicated Star for Ethernet (FTTRS) has been proposed as a communication subsystem that satisfies these two attributes. It is based on the master/multi-slave Flexible-Time Triggered (FTT) communication paradigm and relies on two custom switches, each with its own embedded FTT master. Both masters are active simultaneously and provide the same service. Specifically, they simultaneously and periodically broadcast so-called trigger messages (TMs) in a redundant manner to make them robust to transient channel faults. One of the functions of these TMs is to divide the communication time into rounds called elementary cycles (ECs). For the correct operation of FTTRS, it is important that all slaves agree when each EC starts and ends. A mechanism to achieve this has been recently proposed. This paper presents a first implementation of this mechanism and a series of experimental tests that constitute a first step towards building a prototype of an FTTRS network.**

I. Introduction

A distributed embedded system (DES), to operate continuously while satisfying unpredictable requirement changes, must be both highly reliable and flexible. To achieve this it requires a communication channel that satisfies those attributes as well. The goal of the *Flexible Time-Triggered Replicated Star for Ethernet* (FTTRS) [1] is to provide such a channel for a project called *Fault Tolerance for Flexible Time-Triggered Ethernet-based systems* (FT4FTT), which aims to provide high reliability and flexibility to all crucial parts of a DES.

FTTRS is based on a switched Ethernet implementation of the Flexible Time Triggered (FTT) communication paradigm [2], a paradigm that provides master/multi-slave communication in a way that allows the communication to adapt to changing real-time requirements. FTTRS attempts to make such communication highly reliable for switched ethernet by using fault tolerance. Its architecture is shown in Figure 1. The main components are two interconnected custom ethernet switches, each of which embeds an FTT master, and a set of FTT slaves connected to both of them.

The embedded masters broadcast a periodic message called *trigger message* (TM), which divides the communication time into rounds of fixed duration called *elementary cycles* (ECs). Specifically, each EC begins with a *trigger message window* (TM window) in which each one of the two embedded masters broadcasts several redundant TMs to the slaves while no other traffic is exchanged on the network. The number of TMs

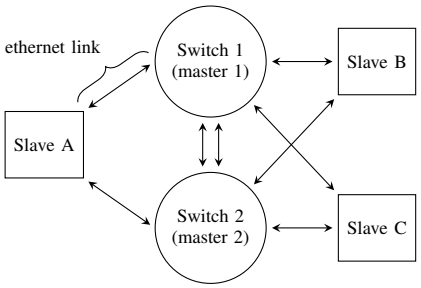

Fig. 1. FTTRS architecture.

broadcast by each master in each EC is given by a parameter k, which is a function of the bit error rate of the channel. Moreover, the broadcasts are synchronized such that when one master transmits its nth TM of a given TM window, the other transmits its nth TM of the same TM window quasi-simultaneously [3]. In other words, the TM transmissions of the two masters occur in lockstep.

For FTTRS to function correctly, the slaves must agree when each EC begins and ends. Since we want FTTRS to be highly reliable, we have recently proposed a mechanism to achieve this even if due to channel faults each slave fails to receive all but one TM per TM window [4]. In this paper we present a first implementation of this mechanism and a series of tests to check that the implementation is correct. Moreover, we provide some first results regarding the viability of achieving a precise EC synchronization in practice with the mechanism.

The remainder of the paper proceeds as follows. Section II summarizes the EC synchronization mechanism used by the slaves. Section III describes our implementation of the EC synchronization mechanism. Section IV describes the tests we performed and the results we obtained. Finally, Section V concludes the paper and points to future work.

II. The slave EC synchronization mechanism

This section summarizes the slave EC synchronization mechanism that was first presented in a previous paper [4].

As mentioned in the introduction, of the k TM replicas broadcast by each master, the corresponding slave might receive all k replicas or only a subset of them due to transient faults. Regardless of which specific replicas each slave receives on each of its links, the time instants when the slaves consider each EC to start and end must align. This can be achieved by

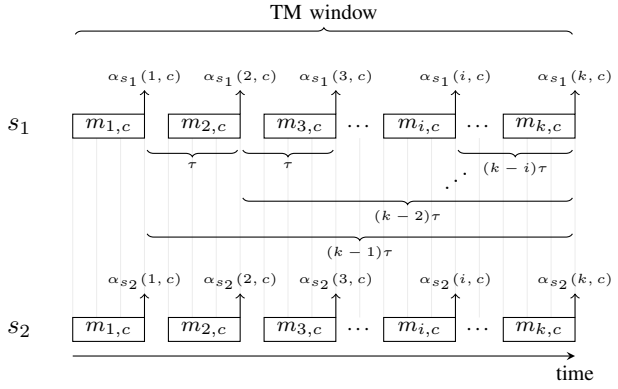

Fig. 2. Alignment of TM arrival times.

the recently proposed EC synchronization mechanism under certain conditions, which we will call *EC synchronization requirements*. These conditions can be summarized as follows: (a) each of the slaves to be synchronized receives at least one TM per TM window; (b) the TMs of a TM window are broadcast such that all slaves that receive the same TM do so at the same time through each of their links; (c) the TMs are broadcast with the same fixed intertransmission time τ; and (d) the amount by which the clocks of the slaves drift apart during one EC is negligible. Under these conditions, the following EC synchronization mechanism can be used for the slaves. (Note that because of condition (b), we do not need to distinguish between the links of each slave.)

Let S_c denote the set of slaves in a given FTTRS network that remain non-faulty at the end of EC c. Moreover, let $m_{i,c}$ denote the TM with sequence number i in EC c, where $i \in \mathbb{N}$ and $1 \leq i \leq k$. Also, let $M_{s,c}$ be the set of TMs that a slave $s \in S_c$ receives during the TM window of an EC c. Furthermore, let $\alpha_s(i,c)$ denote the expected arrival time of $m_{i,c} \in M_{s,c}$ at slave $s \in S_c$. Figure 2 illustrates these expected TM arrival times during the TM window of an EC c for two slaves $s_1, s_2 \in S_c$.

The expected arrival time of $m_{k,c}$ at a slave s is

$$\alpha_s(k,c) = \alpha_s(i,c) + (k-i)\tau, \tag{1}$$

which coincides with the end of the TM window. This is also illustrated in Figure 2 for slaves s_1 and s_2.

If s received at least one TM replica $m_{i,c}$ in c, then (1) can be calculated for all $s \in S_c$ for each EC c. Because of condition (b), in addition we have that

$$\alpha_{s_1}(k,c) = \alpha_{s_2}(k,c), \quad \text{where } s_1, s_2 \in S_c. \tag{2}$$

The EC synchronization mechanism for the slaves can therefore synchronize the end of the TM window of EC c among s_1 and s_2 by using the TM arrival time $\alpha_{s_1}(k,c) = \alpha_{s_2}(k,c)$ as the synchronization event.

III. Implementing the slave EC synchronization

To simplify a first experimental assessment of the EC synchronization mechanism, in our current implementation we abstracted away the presence of two switches and masters. This is a reasonable abstraction for an initial experimental evaluation of the EC synchronization mechanism because according to the FTTRS design the two masters are replica determinate and thus provide identical service from the slaves point of view [3]. The advantage of this abstraction is that it allowed us to test the EC synchronization among slaves without first having to implement the enforcement of master replica determinism, which is required for two masters to transmit their TMs in lockstep. Moreover, it also allowed us to obtain some first results without having to implement how the slaves manage the replication of the TMs caused by having two masters generating sets of k TM replicas.

Regarding the specifics of our implementation, we took as our starting point a software implementation of FTT-SE [5], [6], which is a non-fault-tolerant switched Ethernet implementation of FTT. In this implementation the slaves and masters are implemented as processes to be executed in user space on top of an x86-based computer running a GNU/Linux operating system. In a typical FTT-SE network based on this implementation, several computers running GNU/Linux are interconnected by means of a single commercial off-the-shelf (COTS) Ethernet switch. One of these computers executes the process for the FTT-SE master, and each one of the others executes an FTT-SE slave process.

For our implementation we made several changes to the FTT-SE codebase. First, we modified the code for the master such that it implements the behaviors that are relevant for the FTTRS slave EC synchronization mechanism. Specifically, we made changes for the master to transmit a pre-specified number k of TMs per EC, and to do that with a fixed period τ within each EC. Both k and τ, as well as the desired EC duration, can be passed as arguments to the executable for the master. Moreover, we modified the code for the master to encapsulate k and τ in all TMs, and to add sequence numbers to them. Regarding the EC duration, it was already encapsulated in the TMs in the original FTT-SE codebase.

With respect to the code for the slaves, we modified it to implement the EC synchronization mechanism using a busy wait. When a TM $m_{i,c}$ is received by a slave s at time $\alpha_s(i,c)$, it calculates the time remaining to the end of the current TM window. This time is then added to $\alpha_s(i,c)$ in order to set the absolute time when the TM window in EC c must end. We call this time the *TM window expiration time* of EC c and it coincides with $\alpha_s(k,c)$ of equation (1). If $m_{i,c}$ was the first TM received in c, a thread is created which will, by means of a busy wait, indicate the TM window expiration time. Note that the advantage of using a busy wait is that we avoid sleeping the slave process. This provides better behavior than timers because with timers processes are sent to sleep and must be awoken by the OS, which may cause nondeterministic delays. Finally, as explained in the next section, we added some additional code to the slaves to evaluate the implementation of the mechanism.

IV. Testing the implementation

To test the implementation of the EC synchronization mechanism we used software implemented fault injection (SWIFI) [7]. This means that each slave process executes additional code that forces them to ignore certain TMs. In this way, we can test whether the implementation is robust to TM

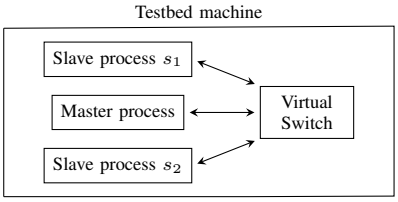

Fig. 3. Virtualized prototype architecture.

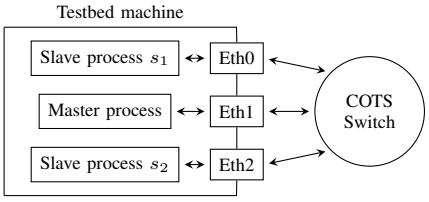

Fig. 4. Physical switch prototype architecture.

losses as foreseen by the design of the EC synchronization mechanism. In particular, the added SWIFI code chooses which TMs must be ignored depending on their sequence number. This is done such that all possible combinations of missing up to $k-1$ TMs on a slave's link are tested, which are all TM loss scenarios under which the designed mechanism can synchronize the slaves. The number of these TM loss combinations is given by

$$\left(\sum_{e=0}^{k-1}\binom{k}{e}\right)^{n}, \tag{3}$$

where k is the number of TMs per EC, e is the number of lost TMs, and n is the number of slaves attached to the network.

In addition to the SWIFI code, we also added instrumentation code to the slaves to timestamp the TM window expiration time of each EC. This allows us to evaluate the precision with which we are able to synchronize the ECs among the slaves.

A. Test setup

To test the slave EC synchronization we used two different setups, but with several commonalities. First, in both setups we used a single master and two slaves. This is the minimum number of slaves and masters required for a first experimental evaluation of the slave EC synchronization. Second, in both setups the process for the master and the two processes for the slaves were executed on the same machine and under the same GNU/Linux OS instance. This made it possible for the slaves to share the same clock, providing a common timebase for their timestamping. Moreover, it simplified initializing the communication between the master and the slaves. Finally, in both setups the master process and slave processes are attached to a single 100 Mbps Ethernet switch. Next, we highlight the differences between the two setups.

1) Virtualized network setup: In this setup the single switch is a virtual one and the processes are attached to it through virtual Ethernet interfaces. Specifically, a *virtual distributed Ethernet switch* is used [8]. This setup allowed us to check the performance of the synchronization without taking into account physical Ethernet interfaces, propagation delay, and switching delay. A diagram of this setup can be seen in Figure 3.

2) Shared machine with physical switch: In this setup the single switch is a COTS Ethernet switch and each process is attached to it by means of a different physical Ethernet interface of the testbed machine. A diagram of this setup is shown in Figure 4.

TABLE I. EXPERIMENT PARAMETERS.

	k	τ (μs)	EC length (μs)	# Test runs
Virtual Switch	4	100	1000	1000
COTS switch	4	100	1000	1000

B. Test parameters and results

To evaluate the results we define the *measured EC offset* between two slaves s_1 and s_2 in cycle c as $|t_{s_1}(c) - t_{s_2}(c)|$, where $t_{s_1}(c)$ and $t_{s_2}(c)$ indicate the recorded timestamp in EC c by slave s_1 and s_2, respectively. In other words, we measure by how much the TM window expiration times of the slaves deviate in each EC. Note that since these times correspond to $\alpha_{s_1}(k,c)$ and $\alpha_{s_2}(k,c)$, respectively, the measured EC offset should be zero according to equality (2) of Section II. Under our real experimental conditions, however, the EC synchronization requirements are not perfectly satisfied and the execution time of the slave processes is not deterministic, e.g., the process might be preempted by the OS. Thus, the measured EC offset is a measure of the precision with which the EC synchronization among two slaves is achieved in practice.

Table I shows the parameters we used in our tests. Specifically, the EC length has been set to 1 ms. This is a suitable value for providing timely communications for typical control applications. Regarding the intertransmission time τ with which the master process sends the TMs to the slaves, it must be greater than the transmission time of a TM, including the Ethernet interframe gap (96 bit times). Since during the TM window only TMs are exchanged on the network, this prevents the TMs from being queued in the switch output ports. This helps ensure that the TMs are not only transmitted with an intertransmission time τ, but also reach the slaves with the necessary interarrival time τ. Note that queuing at a switch output port could prevent this by introducing significant non-deterministic delays. This might occur, for instance, if some TMs occupy a link for a longer or shorter time than the expected TM transmission time due to errors in the link such as dribble bits. In our experiments the TMs do not carry scheduling data [2] and thus are only 14 bytes long, which fits within the 46 bytes of data padding of an Ethernet frame of minimum size (72 bytes). Since we are using 100 Mbps Ethernet, we must therefore set τ to a value greater than $(72 \cdot 8 + 96)/100 = 6.72$ μs. We have chosen a value of 100 μs. We have also chosen to perform 1000 test runs, where each test run injects all possible ways of losing TMs in both slaves with $k = 4$. This yields $(\sum_{e=0}^{4-1}\binom{4}{e})^2 = 225$ fault injections per test run according to equation (3), giving us 225000 sample points. This already provided us with important insights, as described in the next paragraph and Section V, and we had no need for testing different parameter sets with our current

TABLE II. MEASURED EC OFFSET RESULTS.

	# samples	mean (μs)	std. dev. (μs)	max (μs)
Virtual Switch	225000	1.94	0.84	47.62
COTS switch	225000	0.69	1.36	91.37

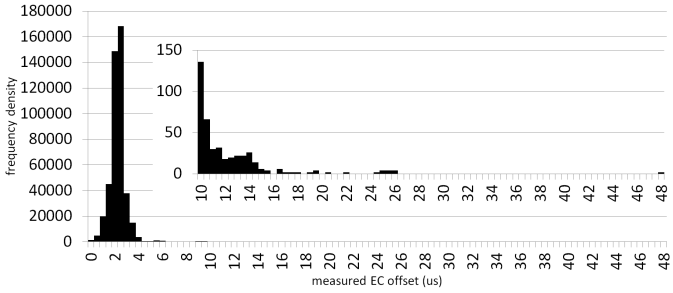

Fig. 5. Histogram of measured EC offset for shared machine with virtual switch. Bin size is 0.5 μs. The superimposed figure is a close-up of the right tail of the histogram.

implementation and experimental setups.

The results are shown in Table II. The mean and standard deviation of the measured EC offset indicate that with both experimental setups the implementation achieves a good EC synchronization, on the order of 0.1–0.2% of the EC length, in most cases. However, the main measure of interest is the maximum measured EC offset. This is so because FTTRS provides real-time communication, where the end of each EC constitutes a hard deadline for the round-based communication to take place. Unfortunately, with the current implementation, and with both experimental setups, we can get values that are significantly larger than the mean: on the order of 5–10% of the EC length. This is also confirmed by the histograms of figures 5 and 6. They both reveal that the distribution of the EC offset has a long tail in the current implementation.

V. CONCLUSIONS AND FUTURE WORK

This paper constitutes a first step towards building a prototype of FTTRS. Specifically, it presents a first implementation of an EC synchronization mechanism for the slaves, which is a key part of FTTRS. The implementation was tested with a single master and two slaves, all executing as processes on the same machine under the same GNU/Linux OS instance. This significantly simplified a first experimental verification of the implementation. The experiments consisted in having the master process transmit its TMs to the slave processes through both a virtual switch and a physical COTS switch. The results of the tests are promising: they show that the implementation achieves a good EC synchronization most of the time with the experimental setups we used. Nevertheless, they also highlighted that occasionally a large EC offset between the slaves can be observed. This, as was to be expected, is due to the slaves being executed as user processes on top of a non-real-time OS. We inherited this from the FTT-SE codebase. If large EC lengths are used, then a maximum EC offset on the order of 50-100 μs, as we have measured, are acceptable. However, with FT4FTT we are also targeting control applications with high sampling rates, where the measured values can be problematic. In this paper we therefore confirmed that the FTT-SE codebase must be further adapted to our needs.

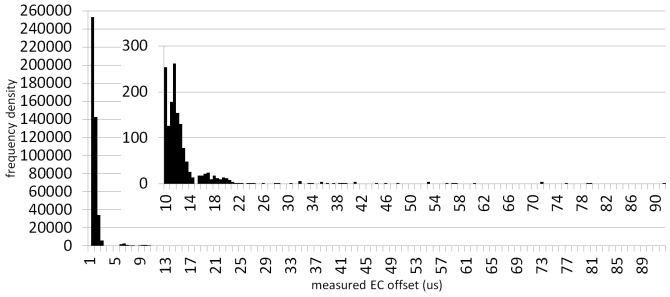

Fig. 6. Histogram of measured EC offset for shared machine with COTS switch. Bin size is 0.5 μs. The superimposed figure is a close-up of the right tail of the histogram.

The next step involves updating the implementation to take advantage of a Linux kernel that offers real-time features. We are considering Xenomai [9] for this. Moreover, we also plan to modify the experimental setup by moving the slave processes and the master process to different machines and measuring the EC offset in this new distributed setup. This will give us a better view of the precision with which ECs can be synchronized among slaves in an actual FTTRS implementation.

Further future work includes implementing the mechanisms to achieve replica determinism for two FTTRS masters and to then test the implementation of the EC synchronization mechanism with two such replica determinate masters.

ACKNOWLEDGEMENTS

This work was supported by project DPI2011-22992 and grant BES-2012-052040 (*Spanish Ministerio de economía y competividad*), and by FEDER funding.

REFERENCES

[1] D. Gessner, J. Proenza, M. Barranco, and L. Almeida, "Towards a flexible time-triggered replicated star for Ethernet," in *Proc. 18th IEEE Conf. on Emerging Technologies & Factory Automation (ETFA)*, Cagliari, Italy, Sep. 2013.

[2] P. Pedreiras and L. Almeida, "The Flexible Time-Triggered (FTT) paradigm: an approach to QoS management in distributed real-time systems," in *Proc. Int. Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2001.

[3] D. Gessner, J. Proenza, and M. Barranco, "A Proposal for Master Replica Control in the Flexible Time-Triggered Replicated Star for Ethernet," in *Proc. 10th IEEE Int. Workshop on Factory Communication Systems (WFCS)*, Toulouse, France, May 2014.

[4] ——, "A Proposal for Managing the Redundancy Provided by the Flexible Time-Triggered Replicated Star for Ethernet," in *Proc. 10th IEEE Int. Workshop on Factory Communication Systems (WFCS)*, Toulouse, France, May 2014.

[5] R. Marau, L. Almeida, and P. Pedreiras, "Enhancing real-time communication over COTS Ethernet switches," in *Proc. 6th IEEE Int. Workshop on Factory Communication Systems (WFCS)*. Torino, Italy: IEEE, 2006, pp. 295–302.

[6] (2014, May) FTT-SE v2.6.2 source code. [Online]. Available: http://paginas.fe.up.pt/~ftt/repository/ftt-se.2.6.2.tar.bz2

[7] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, 1997.

[8] R. Davoli, "Vde: Virtual distributed ethernet," in *Proc. 1st IEEE Int. Conf. on Testbeds and Research Infrastructures for the Development of Networks and Communities*. IEEE, 2005, pp. 213–220.

[9] P. Gerum, "Xenomai—Implementing a RTOS emulation framework on GNU/Linux," *White Paper, Xenomai*, 2004. [Online]. Available: http://www.xenomai.org/documentation/xenomai-head/pdf/xenomai.pdf