Achieving Elementary Cycle Synchronization between Masters in the Flexible Time-Triggered Replicated Star for Ethernet

Alberto Ballesteros[1], Julián Proenza[1], David Gessner[1], Guillermo Rodriguez-Navas[2], Thilo Sauter[3]
[1] DMI - Universitat de les Illes Balears, Spain
[2] Mälardalen University, Västerås, Sweden
[3] Center for Integrated Sensor Systems, Danube University Krems, Austria
{a.ballesteros, julian.proenza}@uib.es, davidges@gmail.com

***Abstract*—For a distributed embedded system (DES) to operate continuously in a dynamic environment, it must be flexible and highly reliable. This applies in particular to its communication subsystem. The Flexible Time-Triggered Replicated Star for Ethernet (FTTRS) aims at providing such a subsystem by means of a highly-reliable switched-Ethernet architecture based on the Flexible Time-Triggered paradigm (FTT), a master/slave communication paradigm where the master periodically polls the slaves using so-called trigger messages (TMs). In particular, FTTRS interconnects nodes by redundant communication paths provided by two switches, each embedding an FTT master that manages the communication. This allows FTTRS to tolerate the failure of one switch without interrupting the communication as long as the masters are replica determinate, i.e., provide identical service to the slaves. The master replica determinism entails the masters broadcasting their TMs in a lockstep fashion: when one master broadcasts a TM, the other should do the same quasi-simultaneously. In this paper we present a solution inspired by the Precision Time Protocol (PTP) for achieving this lockstep transmission and preliminary results showing the precision with which we can synchronize the masters on a software prototype.**

I. Introduction

There is a growing interest in operating distributed embedded systems (DES) in dynamic environments, usually for long periods of time and also with high reliability. This poses new challenges on the design of the communication subsystems for such DES, which have to be flexible but still dependable.

The *Hard Real-Time Ethernet Switching* (HaRTES) [1] has recently been suggested as a suitable infrastructure to support flexible and reliable communication for distributed embedded systems. HaRTES is an implementation of the Flexible Time-Triggered (FTT) paradigm [2] over Switched-Ethernet, and therefore handles real-time communication by means of a centralized master-multislave polling mechanism. Its main feature is that the FTT master is embedded within the switch itself, allowing enhanced error-detection and error-handling capabilities [1]. However, there is one limitation of the FTT paradigm that HaRTES inherits: the master constitutes a single point of failure because any failure of the master to deliver its service will cause communication to cease.

In the FT4FTT project (which stands for *Fault Tolerance for Flexible Time-Triggered Ethernet-based systems*) we propose an extension of HaRTES that eliminates this single point of failure by means of a replicated star [3] [4] [5]. The resulting communication infrastructure is called *Flexible Time-Triggered Replicated Star* (FTTRS) and is composed of two HaRTES-based switches interconnected via several redundant links known as *interlinks*. As depicted in Fig. 1, slaves connect to both switches using dedicated *slave links*.

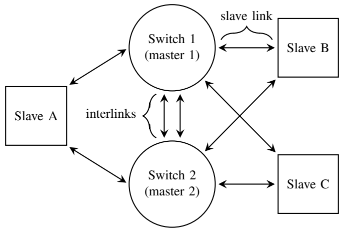

Fig. 1. FTTRS architecture.

The channel replication of FTTRS naturally provides tolerance to failures of the communication links. How to handle this redundancy has been first described in [3]. Later, some mechanisms for handling master replication, based on semi-active replication, were suggested in [4]. However, they were not totally defined and were not validated experimentally. This paper defines the full solution for semi-active master replication in FTTRS and shows some experimental results after a first software prototype implementation.

The problems to be solved by our master replication scheme can be described in terms of the master functionality. In any network following the FTT approach, time is divided into time slots of fixed duration called *Elementary Cycles* (ECs). Every EC is divided into three different windows: *Trigger Message*, *Synchronous* and *Asynchronous* window.

The *Trigger Message window* (TMW) is used by the master to construct and issue the *Trigger Message* (TM). This message notifies the slaves that a new EC has started and contains the list of messages that must be sent during the *Sync window*. The challenge is therefore to ensure that, upon master failure, the other master can take over immediately without disturbing the on-going communication. Since the main function of the master is to send the TM, this requires that both masters are *replica determinate* with respect to the TM. That is, they must be able to provide the same TM at approximately the same instant. In this way, master replacement can be made totally transparent for the slaves. This paper describes the two mechanisms that have been implemented in order to fulfill this property: (a) a novel fault-tolerant protocol for EC initialization, and (b) a mechanism for EC synchronization between the master replicas, which is based on a subset of the IEEE 1588 standard for clock synchronization.

II. Related work

Replica determinism in the time domain for the masters was already discussed in [4], where the authors identified the temporal problems to be addressed, and sketched a first solution. They suggested a semi-active master replication scheme, in which one master, called the *leader*, supplies the timing of the whole system. This master notifies the beginning of each EC by broadcasting its TMs to the slaves, through the slave links, and to the other master, through the interlinks. The second master, called the *follower*, autonomously issues its TMs to all the other nodes as well. However, the follower uses the leader's TMs to resynchronize and, thus, keep in pace with the leader. That is, it uses the arrival times of the leader TMs to hasten or delay the transmission of the next TMs.

Using the TM for synchronization purposes has advantages both for reliability and performance. On the one hand, as described in [5], during the TMW each master transmits k replicas of the TM to all the slaves and the other master, where k is set considering the bit error rate of the channel. If k is large enough, at least one TM replica will reach each of the recipients with a sufficiently high probability, even in presence of transient errors. Thus, the follower will be able to resynchronize in every EC. On the other hand, using the TMs makes the transmission of additional synchronization messages unnecessary, so the communication overhead is actually null.

Regarding the reliability of this solution, the follower remains synchronized unless the leader crashes or all the interlinks suffer a permanent fault. The first situation is transparently tolerated because the follower would continue its operation as the new leader, without resynchronization. In contrast, if the interlinks become disabled, the network is partitioned in two and the masters would not be able to stay synchronized. We are currently investigating how to enhance the slaves to be able to detect and handle this situation.

The preliminary design in [4] did not include any mechanism for time synchronization between masters. However, there have been previous efforts in addressing equivalent issues, like in [6], where the authors proposed an algorithm to synchronize multiple masters in an FTT multi-switch architecture. In that proposal, masters exchange clock synchronization messages during the *sync guard* window, i.e., a new window inserted at the beginning of the EC. These messages are the same as defined in the IEEE 1588 standard described next.

The IEEE 1588 standard, known as the Precision Time Protocol (PTP) [7], is a protocol defined for measurement and control systems that allows synchronizing distributed clocks with a precision down to the microsecond range. This protocol is based on a master-slave scheme, i.e., one or more PTP slaves adjust their clocks with respect to a reference clock provided by a PTP master. As can be seen in Fig. 2, first of all, the PTP master transmits to a given PTP slave the so-called *Sync* message. When doing so, the PTP master timestamps the transmission time t_m; whereas the PTP slave timestamps the reception time t_s. After that, the PTP master transmits the *Follow_Up* message, which contains the value of t_m.

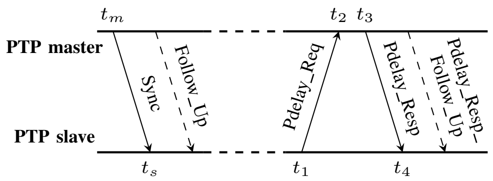

Fig. 2. Exchange of messages in PTP.

Additionally, PTP makes it possible to determine the propagation delay, i.e., the time needed by the *Sync* message to travel from the PTP master to the PTP slave. Assuming a symmetric delay, the *peer delay* mechanism can be used for this purpose. This mechanism initiates with the PTP slave sending a propagation delay request or *Pdelay_Req*. Similarly as with the *Sync* message, the PTP slave timestamps the transmission time t_1, whereas the PTP master timestamps the reception time t_2. Afterwards, the PTP master responds by sending a propagation delay response or *Pdelay_Resp*. The transmission and reception times, i.e., t_3 and t_4, are also registered. Finally, the PTP master transmits t_2 and t_3 by means of the *Pdelay_Resp_Follow_Up* message. The PTP slave determines the offset of its clock with respect to the PTP master using (1), where the fraction is the propagation delay.

$$offset = t_s - t_m - \frac{(t_4 - t_1) - (t_3 - t_2)}{2} \tag{1}$$

III. Detailed design

This section describes the ongoing research carried out to complete the preliminary design of the EC synchronization mechanism for the masters presented in [4] using some of the ideas described in [6].

In our synchronization mechanism we distinguish between two levels of synchronization that must be achieved. (a) The first level provides an initialization for the two masters to start their execution in a synchronous manner from the first EC onwards. This ensures that an FTTRS network tolerates master failures from the very first EC. Moreover, this level also calibrates the parameters required by the second level. (b) The second level is the synchronization of the ECs that allows both masters to transmit their TMs in lockstep. For this, the follower uses the arrival time of a leader's TM replicas to infer when the leader will start the next EC. Using this inferred time instant, the follower adjusts its own timing to transmit its own TM replicas simultaneously with the leader.

A. Level 2: Periodic lockstep resynchronization

As indicated above, the second level of synchronization allows the follower to know when it should start transmitting the TM replicas for the next EC, so as to do it as simultaneously as possible with the transmission of the TM replicas by the leader. In order to do this, the follower takes into account the arrival times of the leader's TM replicas for the current EC.

Fig. 3 identifies the relevant instants of the TM transmission. When the leader considers that EC number n has started, denoted $t_{\text{begin}}^{\text{lead}}(n)$, it constructs the corresponding TM taking into account the scheduling calculated in the previous EC. The time required for this is denoted δ_1. Then, at $t_{\text{TM}}^{\text{lead}}(n)$, it instructs the transmission of the first TM replica, denoted

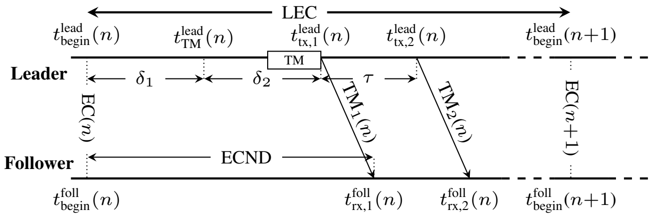
LEC
$t^{\text{lead}}_{\text{begin}}(n)$
$t^{\text{lead}}_{\text{TM}}(n)$
$t^{\text{lead}}_{\text{tx,1}}(n)$
$t^{\text{lead}}_{\text{tx,2}}(n)$
$t^{\text{lead}}_{\text{begin}}(n+1)$
TM
Leader
δ_1
δ_2
τ
EC(n)
$\text{TM}_1(n)$
$\text{TM}_2(n)$
EC(n+1)
Follower
ECND
$t^{\text{foll}}_{\text{begin}}(n)$
$t^{\text{foll}}_{\text{rx,1}}(n)$
$t^{\text{foll}}_{\text{rx,2}}(n)$
$t^{\text{foll}}_{\text{begin}}(n+1)$

Fig. 3. EC synchronization instants.

$\text{TM}_1(n)$. The time needed for this is denoted δ_2 and may significantly vary depending on the technology used, e.g., in a software implementation the operation of the OS has to be considered. At $t^{\text{lead}}_{\text{tx,1}}(n)$ the network interface issues the last bit of $\text{TM}_1(n)$ to the channel and, after the propagation delay, it reaches the follower, who then registers the arrival of $\text{TM}_1(n)$ at $t^{\text{foll}}_{\text{rx,1}}(n)$. As explained in [5], this transmission is carried out k times, with a constant separation of τ. Consequently, each time a TM replica is received by the follower, it registers $t^{\text{foll}}_{\text{rx},i}(n)$, where $i \in \{1, 2, 3, \dots k\}$.

Referring to Fig. 3, the beginning of the follower's next EC can then be calculated as

$$t^{\text{foll}}_{\text{begin}}(n+1) \approx t^{\text{foll}}_{\text{rx},i}(n) - (i-1)\tau - \text{ECND} + \text{LEC}, \tag{2}$$

where $t^{\text{foll}}_{\text{rx},i}(n)$ is the reception instant of any of the TM replicas; ECND is the *EC notification delay*, i.e., the time elapsed since the leader considers a given EC to have started until the follower receives the TM for said EC; and LEC is the fixed length of the ECs. Note that ECND is constant and is measured during the level 1 synchronization, as described in the next section. It is important to highlight why we consider that it is constant. First, the time δ_1 needed to construct the TM can be considered constant. This is because inserting the scheduling into the TM takes a negligible amount of time. Second, the time δ_2 needed to process the TM's transmission is constant. This is due to the fact that during the TMW the channel is idle and, thus, the internal transmission queue of the leader is empty. Consequently, there are no additional delays provoked by the transmission of other messages. Third, the propagation delay is constant. The reason for this is because the topology of the interlinks does not change. Finally, the time needed to process the TM's reception is constant. This is because, like with the transmission, the reception queue of the follower is empty and, thus, no message reception can provoke additional delays.

Following the PTP approach, we consider the leader as PTP master and the follower as PTP slave. In this sense, the TM is our *Sync* message, i.e., a message periodically transmitted by the leader to resynchronize the follower. However, since we do not need t_m, the *Follow_Up message* it not used. Apart from the clock state synchronization mechanism using the TM, we also perform a rate correction, i.e., we decrease the deviation suffered between two consecutive EC adjustments by considering the relative speed between leader and follower clocks. To do so, the follower measures the time between m resynchronizations and compares it with the expected value, i.e., m times LEC. The result is the relation between the two clocks, which is taken into account by the follower when computing the time for the next EC.

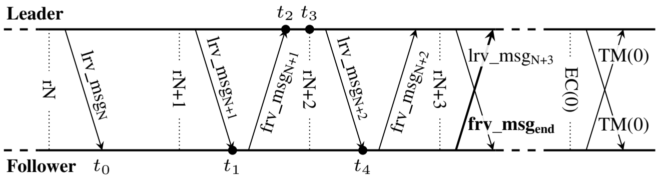

Fig. 4. Rendez-vous handshake.

B. Level 1: Achieving simultaneous start

In order to have master redundancy from the very beginning, it is necessary to force both masters to start their execution at the same time. For this, we introduce an initialization phase called *rendez-vous phase*, in which both masters exchange some messages to agree on when the first EC has to start. Additionally, the follower takes advantage of this exchange of messages to determine the ECND using a modification of the PTP peer-delay mechanism.

As depicted in Fig. 4, during the initialization the leader starts by periodically sending the *leader rendez-vous message* (lrv_msg). Similar to the TMs, this message divides the communication time into rounds which now are called *rendez-vous rounds*. These rounds have the same length as the ECs, i.e., LEC, which ensures that the conditions during the ECND measurement are identical to the ones during the normal operation of the masters. Note from the figure that the first lrv_msg seen by the follower is the one in a round N (lrv_msg$_{\text{N}}$). After that, the follower starts transmitting its own rendez-vous messages (frv_msgs). The goal is for both masters to transmit their rendez-vous messages in lockstep by the end of the initialization. However, at the beginning of the initialization the messages of the masters are out-of-sync since the follower does not know when round N began according to the leader. To achieve the synchronization, the follower begins by using instead the only temporal reference it has from the leader, namely the arrival time of lrv_msg$_{\text{N}}$ at t_0. The follower considers the next round to start at $t_0 + \text{LEC}$ and, thus, its rendez-vous message transmissions are initially delayed with respect to the leader's by the amount of time it takes a leader's rendez-vous message to reach the follower since the start of round N. To correct this phase error, the follower must therefore infer this time, which is equivalent to the constant ECND used in (2). This can be achieved by a mechanism similar to the one used in PTP to infer the propagation time.

Each new round leader and follower issue their rendez-vous messages. This exchange of messages allows the follower to determine the ECND, similarly as done for the propagation delay in PTP. Specifically, each frv_msg and lrv_msg is like a Pdelay_Req and Pdelay_Resp in PTP, respectively. The only difference to PTP is that t_1 and t_3 are registered at the instant in which follower and leader, respectively, consider the current round to have started. With this modification we take into account δ_1 and δ_2, which are relevant in this measurement. Additionally, with every lrv_msg the leader conveys $t_3 - t_2$. Thus, additional messages are not needed, as in PTP with the Pdelay_Resp_Follow_Up message. Finally, assuming that $\delta_1 + \delta_2 + \mathit{prop\_delay}$ is symmetric, ECND can be calculated using the same formula for the propagation delay in the peer-delay mechanism (see Eq. 1).

As can be seen in round N+3, once the ECND has been measured, the follower is synchronized with the leader and, thus, both issue their rendez-vous messages simultaneously. At this point the follower instructs the leader to finish the rendez-vous phase by transmitting a predefined rendez-vous message (frv_msg$_{\text{end}}$). This forces both masters to wait for the next rendez-vous round and then start the normal operation and, thus, the first EC, at the same time.

As concerns the reliability of this mechanism, note that channel errors can seriously affect the synchronization. For instance, the omission of one rendez-vous message would lead to a severe error in the calculation of the ECND. Thus, to tolerate transient errors, ECND is measured multiple times until its mean value does not significantly vary. Regarding the omission of frv_msg$_{\text{end}}$, we use the same approach as in the TM temporal redundancy mechanism, i.e., this message is transmitted k times.

IV. Experimental assessment

We carried out an experiment to both assess the correctness of the design and take preliminary measures of the precision that can be achieved in a software implementation. The testbed is composed of three stations: two interconnected master stations, leader (LS) and follower (FS), and one monitoring station (MS) connected to both master stations. Each master station is implemented in a regular multi-core PC and executes the FTT code for the the master, i.e., it transmits TMs and synchronizes with the other master as described. The MS is built using hardware for embedded devices so that the results provided are close to the ones in a real implementation. The data it provides is the TM offset, i.e., the absolute value of the time elapsed between the reception of the TMs issued for the same EC by LS and FS. Thus, a TM offset of zero would mean a perfect synchronization precision. The arrival times of the TMs are timestamped in the MS using *tcpdump*, an OS tool that provides one microsecond resolution. Finally, each station runs a regular GNU/Linux OS.

In our first experiment we registered the reception of one million TMs, using an EC length of 1 ms. This EC length makes desynchronization between ECs negligible, thus in this experiment we do not apply the rate synchronization mechanism. In Fig. 5 we show the relevant statistical results, a histogram of the TM offsets with a bin size of 1 μs, and a bar diagram showing the value of each sample. The first thing to highlight is the good results achieved in terms of the mean and standard deviation values, which means an average deviation of 0.6-0.83 % of the EC length. Second, regarding the histogram, note that it does not show all the values. This is because bin 31 μs and onwards have a frequency of 6 samples or less and, thus, it is not worth to represent them since they would be unnoticeable. The most relevant values appear in bins 6 and 7 μs, which embrace more than half of the samples. Finally, in the bar diagram, we can observe some peaks that reach up to 362 μs, i.e., the maximum. This value is important as it determines the level of synchronization that can be achieved. In our case, the follower can be hastened or delayed with respect to the leader more than the third of the EC, which is not acceptable. However, since these deviations are not very common (7 samples over 1 million exceed 100 μs) means that they are likely provoked by the indeterminism of the OS and, thus, that they can be reduced if we devote further efforts in tuning the software components of the system.

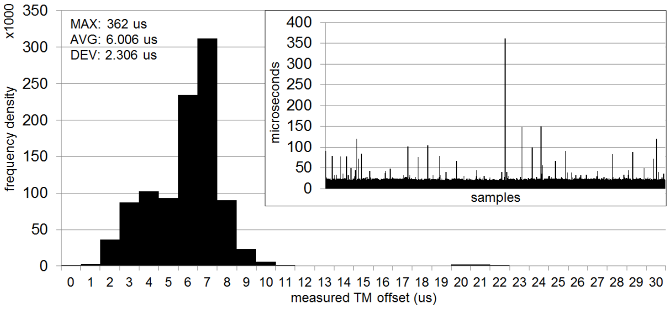

Fig. 5. Experiment results. The main figure is the histogram of the TM offsets with a bin size of 1 μs. The inset shows the value for each sample.

V. Conclusions and future work

In this paper we complete the design of the EC synchronization mechanism between the two master replicas in FTTRS, which was presented in [4]. For this, we adapt some of the ideas used in [6]. However, since we rely on the TM to perform all the synchronization tasks, there is no need for additional messages. This solution is verified by means of an experiment, in which we asses its correctness and take preliminary measures of the precision that can be achieved in a real software prototype. These experiments showed good results in terms of mean value and standard deviation. However, significant time deviations have been found. These deviations are provoked by the software components of the prototype, which are important sources of indeterminism. The next step involves tuning the software components to achieve a more deterministic behaviour of the system. We also address for future work a rendez-vous mechanism capable of seamlessly reintegrating a desynchronized master without having to stop the normal operation of the system.

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] R. Santos, "Enhanced Ethernet Switching Technology for Adaptive Hard Real-Time Applications," Ph.D. dissertation, Universidade Aveiro, 2011.

[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 Comput. Soc, 2001.

[3] D. Gessner, J. Proenza, M. Barranco, and L. Almeida, "Towards a Flexible Time-Triggered Replicated Star for Ethernet," in *18th IEEE Conf. on Emerging Technologies & Factory Automation (ETFA)*, 2013.

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

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

[6] M. Ashjaei, M. Behnam, G. Rodriguez-Navas, and T. Nolte, "Implementing a clock synchronization protocol on a multi-master Switched Ethernet network," *18th IEEE Conf. on Emerging Technologies & Factory Automation (ETFA)*, 2013.

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