Towards a node active replication schema for highly reliable distributed control systems based on TSN

Joan Evangelisti, Manuel Barranco, Julián Proenza, Alberto Ballesteros, Mateu Jover
Departament de Ciències Matemàtiques i Informàtica, Universitat de les Illes Balears, Spain
joanevan24@gmail.com, manuel.barranco@uib.es, julian.proenza@uib.es, a.ballesteros@uib.es, mateu.jover@uib.es

***Abstract*—Given their nature, many control applications that arise from the integration of Operation Technologies (OT) and Information Technologies (IT) are built on top of highly reliable real-time (RT) Distributed Control Systems (DCSs). Since a DCS is made up of several computing nodes that exchange information through a communication subsystem, to achieve high reliability it is necessary that both this subsystem and the service from the nodes are very reliable. To provide RT highly reliable communications while benefiting from Ethernet's advantages, Industry and Academia are pushing the Time-Sensitive Networking Ethernet standards (TSN). On the other hand, one of the most used strategies to ensure a highly reliable service from the nodes is to use fault tolerance in the form of active replication. Our general goal is to develop a complete fault-tolerant architecture (addressing faults both in the communication subsystem and in the nodes) for highly reliable real-time DCSs based on TSN. In this paper we show our ongoing work towards an active replication schema for the nodes of this architecture.**

***Index Terms*—reliability, fault tolerance, TSN, distributed control system, node replication**

I. Introduction

Nowadays new industrial control and automation applications are emerging from the integration of Operation Technologies (OT) and Information Technologies (IT). Given their nature, many OT-IT applications rely on real-time (RT) highly reliable *Distributed Control Systems* (DCSs). Examples of these DCSs can be found in contexts such as smart energy production and distribution networks (Smart grids), smart cities, or smart autonomous transportation.

An RT DCS must provide its services, e.g., execute a control algorithm to calculate actuation values, within deadlines. On the other hand, a highly reliable DCS must continuously provide a correct service with a high probability during a given time interval, e.g., a DCS that automates certain functions of a commercial aircraft must operate correctly throughout the entire flight.

A DCS is made up of several computing nodes that coordinate with each other by exchanging information through a communication subsystem. Therefore, to achieve very high reliability it is necessary that both the communication subsystem and the nodes provide their service in a very reliable manner.

When it comes to communications, the de facto link layer protocol for IT systems is Ethernet, given its high bandwidth, low cost, and its native integration with TCP/IP. So, over the past decades, a myriad of Industrial Ethernet protocols have been proposed to benefit from these advantages while enforcing high reliability and real-time guarantees for OT and OT-IT applications. However, these protocols provide partial solutions and/or present integration limitations between them. In order to overcome these limitations, both Industry and Academia are intensively promoting what are known as Time-Sensitive Networking Ethernet standards (TSN). In particular, TSN specify mechanisms to provide high reliability and real-time guarantees.

Regarding the nodes, one of the most used strategies to achieve high reliability is to use fault tolerance (FT) in the form of active replication [1]. This consists in providing the system with several independent replicas of each critical node. All replicas of a given critical node operate in parallel to produce each relevant result, e.g., the value of a sensor, an actuation value etc.; exchange it over the communication subsystem; and locally vote on it to obtain a consensus result to continue operating to produce the next result.

In order to reach a very high reliability it is fundamental to design fault-tolerant architectures which are *complete*, i.e., that include mechanisms for tolerating faults both in the nodes and in the communication subsystem. Currently, our research focuses on proposing such a complete fault-tolerant architecture for highly reliable RT DCSs based on TSN. In this sense, our main objective is twofold: (1) to propose a **node active replication** schema specially tailored to **TSN**; (2) to investigate how to use and configure different TSN, as well as developing TSN-compatible FT mechanisms, to provide a highly reliable TSN network. This paper describes our **ongoing work** towards the **1st** one of these **objectives**. To our best knowledge this is the first work addressing node replication over TSN.

First, we briefly explain what are the main aspects to consider for providing an adequate node active replication schema for DCSs in general, as well as what decisions we took so far for providing that schema on top of TSN. Then we sketch the node replication schema we propose, and show some preliminary tests on an ongoing prototype. Finally we conclude the paper and point out future work.

II. Design rationale

As just explained, a node active replication schema consists in having several independent node replicas (*replicas* for short) that execute the same functionality (application) in parallel. The program that implements this functionality can be exactly the same on each replica; but it can also be different on each one, to provide *program design diversity* and thus higher *fault independence* between replicas. The program can be divided into *phases* which are executed in a cyclic manner. At the end

of some phases, each replica exchanges with the others the relevant values (results) it obtained in that phase. Then, each replica locally votes on the exchanged values to agree with the others on a set of *consensus* values it uses to compute the next phase [6].

The way in which replicas coordinate to execute each phase in a synchronized manner depends on whether the application follows a *time-triggered* or an *event-triggered* approach. Since events may not happen simultaneously in each replica, and the execution time of any given phase can be different on each one, an event-triggered application makes it more complex to coordinate replicas so that they carry out phases in parallel. Thus we chose to follow a time-triggered application approach like in [3], in which all replicas start the execution of each phase at approximately the same time instant. Therefore, it is necessary to provide replicas with some kind of synchronization mechanism so that they execute the phases following the same time basis. Moreover, for the sake of robustness, that mechanisms should not depend on an external connection, e.g., GPS connection. Instead, the preferred solution is to rely on a mechanism provided by the communication subsystem itself. Fortunately TSN provides a *clock synchronization* mechanism called gPTP, standardized as 802.1AS, which allows nodes (and bridges) to synchronize with a so-called *global clock* (which is local to the network) with very high precision. This clock synchronization makes it possible to trigger the execution of any phase in all replicas quasi-simultaneously.

Clock synchronization also allows both that replicas transmit their values at the right times, and that *traffic shapers* (usually located at bridges) adequately forward those transmissions. In this sense note that TSN shapers represent an interesting advantage, as in principle any kind of TSN shaper can be used to ensure that the frames that carry the values proposed by the replicas are timely exchanged for voting in the corresponding phase.

The number of replicas, how they exchange their values, and how they vote basically depends on the *fault model* (types of faults), the *failure model* (how failures of components manifest) and the *connectivity* (which replicas can communicate with each other) [2]. Our fault model considers temporary and permanent unintentional faults (both internal like short-circuits and external like EMIs) affecting the hardware of nodes, bridges and links. For the failure model we assume that nodes and bridges can show *byzantine* failures, i.e. they can fail in any manner, and that links exhibit *omission* failures (frames corrupted at links are dropped by nodes and bridges).

As concerns the connectivity, to cover a broad range of control applications we consider that, in absence of faults, all replicas can communicate with each other. In this sense we define a *non-faulty replica* as a node replica that not only operates correctly, but that in addition can communicate with all other non-faulty replicas.

In order to both keep a cost-reasonable amount of node replicas and to reduce the complexity of the procedures they use to exchange and vote on messages, it is necessary to provide the system with mechanisms that *restrict* the failure model of nodes and bridges, i.e. mechanisms that force them, when faulty, to exhibit failures in modes that are more benign than byzantine ones.

For the nodes we propose to force them, when faulty, to exhibit *incorrect computation* failures, so that from the point of view of the application executing at each non-faulty replica, a faulty replica manifests in the worst case by either proposing incorrect application-level values for voting or proposing them at the wrong time. This would simplify the way in which faulty replicas are tolerated, i.e. non-faulty ones would just need to carry out a *majority voting* on its own values and the values received from the other replicas, as in [3].

Restricting node failures in such a way, when using a non-standardized communication subsystem, can be done by means of interconnection adhoc devices. For instance, the bridges of [3] include at its incoming ports adhoc (non-standard) error-containment mechanisms to drop any frame that is incorrect at the data link layer; so that an incorrect frame received at a replica can only be incorrect because either it includes erroneous application data or arrives at a wrong instant. Moreover, in those systems the adhoc interconnection devices can be designed to be *fail-silent*, as proposed for bridges in [3]. Relying on those bridges would be ideal, as their failure cannot compromise the majority voting (as long as a majority of non-faulty replicas can communicate), since a fail-silent bridge can only lead replicas observe that that bridge omits messages.

However, in the present work we impose the requirement of using off-the-shelf TSN devices for nodes and bridges, and keep full compatibility with TSN standards. This requirement makes more complex to restrict the failures of node replicas, and hardly possible to provide fail-silent bridges.

Thus, to restrict the failure modes of nodes and bridges as much as possible, while keeping full compatibility with TSN standards and off-the-shelf nodes and bridges, we are conducting a parallel research work as part of our 2nd general objective outlined in Section I. In that parallel work we aim to realistically characterize the way in which off-the-shelf TSN nodes and bridges fail, and then, combine and configure several TSN mechanisms to maximize error containment. This will minimize the severity with which failures manifest from the point of view of non-faulty replicas, and thus will allow simplifying the node replication schema. The preliminary results of that work show that although TSN mechanisms can be combined to treat most severe failures, we would need to include further mechanisms at a higher-level of the architecture (to avoid compromising TSN compatibility), e.g., provide nodes with a library between the application and data link layer that includes mechanisms to authenticate transmitting nodes, so as to prevent impersonations.

Once these additional mechanisms are added to the architecture, faults in nodes will be perceived by the rest of the nodes as incorrect computation failures. Thus, in the present work we can propose a cost-reasonable node active replication schema based on majority voting that just needs to include $N = 2k+1$ replicas to tolerate the failure of k of them.

At this point note that to tolerate faults it is mandatory

that non-faulty replicas are *replica deterministic* [4], i.e. all non-faulty replicas must have the same internal state (S_A) and, when they receive the same inputs, they must produce the same internal state change ($S_A \rightarrow S_B$) and the same outputs. Replica determinism can be enforced by means of *internal* and *external replica control techniques* [4]. We will ensure that replicas are internally deterministic by not using non-deterministic program constructs in their software.

For replicas to be externally deterministic, we need to ensure that they consistently and timely vote on the same inputs. Thus, first the TSN communication subsystem must satisfy the property of *reliable broadcast* [5]; which in our case means that whenever a non-faulty replica sends a frame, all non-faulty replicas will eventually receive it. Second, the TSN communication subsystem must satisfy the real-time (RT) requirements of these frames, so that all non-faulty replicas receive them timely for voting.

Enforcing reliable broadcast, while fulfilling the RT requirements, in TSN is also part of our 2nd general objective outlined in Section I and, thus, we are addressing it in a parallel work. In that work, to tolerate permanent faults in the communications we propose to connect each replica to two bridges of a redundant network topology (TSN allows using any topology). Specifically, we will use the IEEE 802.1CB *Frame Replication and Elimination* (FRER) TSN standard to provide redundant physical paths (*space redundancy*) between each pair of replicas over that topology. To tolerate transient faults in the communications we plan to combine FRER with a *time redundancy* strategy in which frames are pro-actively retransmitted. For enforcing the RT requirements we will use traffic shapers.

Finally, it is important to provide nodes with mechanisms that prevent unnecessary *redundancy attrition* (loss of node replicas) caused by faults that are not actually permanent. One of such mechanisms is a *Forward Error Recovery* mechanism intrinsic to the majority voting, i.e. a replica that produces erroneous values due to a transient fault can overwrite those values with the consensus values it obtains after voting and, then, continue operating correctly. However, more complex non-permanent faults require more advanced recovery mechanisms, like the ones proposed in [3]; thus we will research how to make the most of TSN to propose such kind of mechanisms.

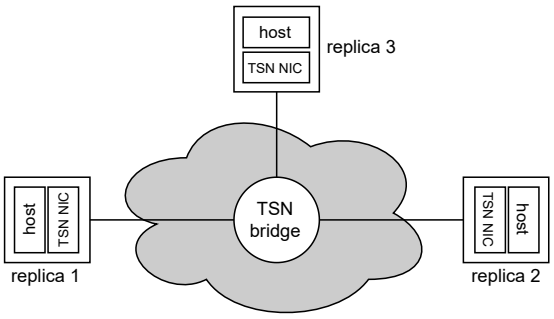

Fig. 1. Prototype's simplified topology

III. Ongoing prototype and preliminary tests

We are currently implementing a first prototype of our node active replication schema for TSN-based DCSs. Since our schema is independent from the topology, this prototype relies on the simple TSN network depicted in Figure 1. It is composed of 1 TSN bridge and 3 node replicas. Each replica is equipped with an *Intel Core i7-4770* 3.4GHz CPU (8 cores), running *Ubuntu 20.04*; 8 GB of RAM; and a *I350 Gigabit TSN Ethernet Network Interface Card* (NIC). The TSN bridge (switch) is the *SoC-e MTSN kit*. This bridge and the TSN NICs include several TSN standards, e.g., 802.1AS (gPTP), 802.1CB (FRER) and 802.1Qbv (TAS). Additionally the replicas are connected to a PC via a regular Ethernet switch not depicted in the figure. This allows sending test commands and retrieve debug information to/from replicas via SSH.

Figure 2 sketches the internal structure of a replica. The TSN NIC provides services for transmitting/receiving, as well as for gPTP clock synchronization. All replicas' NIC and the bridge should have the same view of this clock, so that they are tightly *synchronized at the communication level*.

The Data Link layer of the replica basically includes the *TSN lib*, which is a library we implemented to access the different transmission/reception services of the TSN NIC. This library basically provides a primitive to request for the transmission (*tx*) of Eth TSN frames, as well as an interrupt handler and a reading primitive for receiving such frames.

At the Application layer we basically implemented a library called *Clock lib* and two threads, namely *Control* and *Rx demux*. Clock lib basically includes a primitive called *wait(t)*, which allows the Control thread to actively wait until the gPTP clock reaches a specific instant of time t, as explained next.

The Control thread basically executes cyclically the 3 actions - *sense* (S), *calculate actuation values* (C), and *actuate* (A) - of a typical control application. The execution of this control cycle is divided into N phases (Ph_i). For replicas to execute these phases in a synchronized manner they must be *synchronized at the application level*, i.e. they must agree on what phase to execute and must trigger its execution quasi-simultaneously. The Control thread of each replica is configured with a set of *trigger instants*, which are common to all replicas, and which it uses as the parameter to call the *wait(t)* primitive of Clock lib. The first instant is t_{start}, which is the time at which all replicas must synchronize before executing the control cycle for the very first time. Each one of the other trigger instants (t_i) corresponds to the time at which replicas must trigger a given phase (Ph_i). After executing a control cycle, the Control thread of each replica updates the t_i(s) for the next cycle.

Note that some phases are used to carry out the above basic control actions, i.e. S, C or A. However, some phases are devoted for replicas to exchange the values they produce, whereas other phases are used to vote on those values. To exchange a set of values, the corresponding phase of Control thread calls the *tx* primitive of TSN lib. Thanks to this primitive, the phase can order the NIC to send the necessary

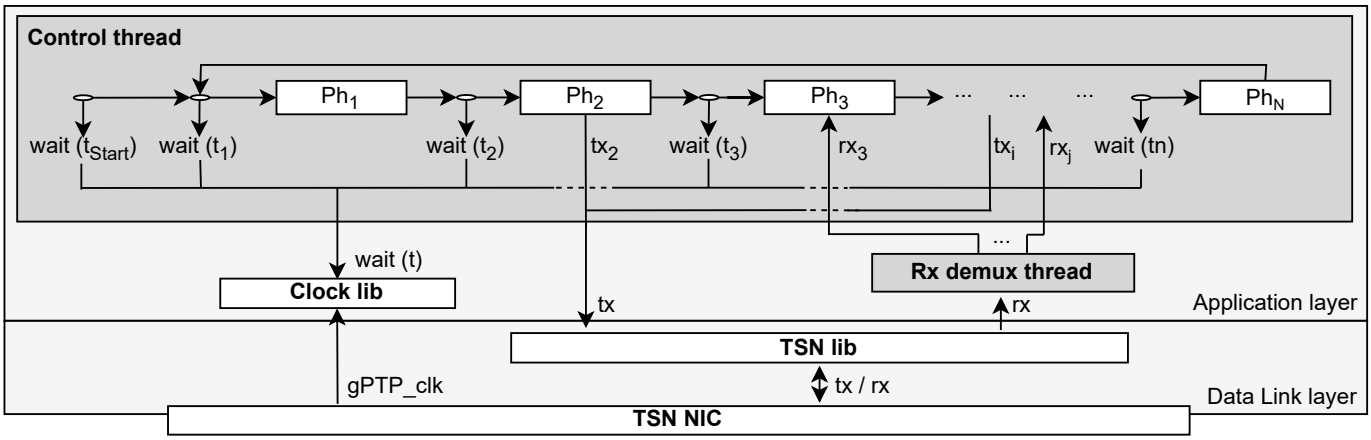

Fig. 2. Replica prototype internal schema

frames to the other replicas. The payload of these frames must include both the values to be transmitted and the meaning of those values. When a destination replica receives one of these frames, its NIC generates a reception interrupt that is processed by its TSN lib to activate the Rx demux thread. This thread uses TSN lib to read the frame payload; analyzes the meaning of the payload values; and stores them in a buffer dedicated to that meaning, e.g. a buffer dedicated to storing the value of the sensors. Eventually, the phase of Control thread responsible for voting using those values will extract them from that buffer. In this sense we say that Rx demux demultiplexes the receptions towards the corresponding phases of Control thread.

To check the feasibility of our node active replication schema, the current prototype includes a preliminary version of the phases. The control cycle is composed of 7 phases, each of which takes no longer than 5 ms to complete. This results in a cycle length of < 35 ms, which is an adequate value for many tight control applications. There are phases that produce data, phases that transmit frames, and phases that read them. On the one hand, we checked that the Control thread of each one of the 3 replicas adequately trigger each phase in a tightly synchronized manner. We observed not only that the gPTP clocks of the replicas are synchronized with the gPTP master clock of the TSN bridge with a precision on the order of hundreds of nanoseconds (which complies with the 802.1AS specification); but also that the Control threads trigger each phase quasi-simultaneously with a precision of ≤ 5us, which represents a close synchronization given the duration of each phase. On the other hand, we corroborated that the Control threads successfully transmit the intended frames, and that the Rx demux threads correctly demultiplex the reception of those frames.

IV. Conclusions and future work

Many emerging industrial applications are based on real-time (RT) highly reliable *Distributed Control Systems* (DCSs). These systems can achieve high reliability by relying on a fault-tolerant (FT) *complete architecture*, which includes FT mechanisms for both the nodes and the communication subsystem. Since TSN is expected to be the de facto communication technology of Ethernet-based industrial systems, we aim to propose a complete FT architecture for highly reliable RT DCSs based on TSN.

This paper describes our ongoing work on the FT mechanisms for the nodes of such architecture. Specifically we propose a node active replication schema taking into account the main particularities of TSN, and show the first feasibility results of an ongoing prototype. Although active node replication has been classically used for node FT, to our best knowledge this is the first work that proposes such kind of replication on TSN.

We plan to further extend the node FT mechanisms proposed here (e.g. to mitigate node redundancy attrition) and, then, integrate into a complete FT architecture both our node replication schema, and a fault-tolerant TSN communication subsystem we are developing in a parallel research work. This integration into a complete FT architecture will throw light on how to adequately combine, configure (and extend) TSN mechanisms so as to build complete highly reliable real-time TSN-based DCSs.

Acknowledgment

Project PID2021-124348OB-I00 funded by MICIU/AEI/10.13039/501100011033 and by FEDER, UE. Mateu Jover was supported by the Consejería de Educación y Universidades del Gobierno de las Illes Balears under the contract FPU2023-007-C.

References

[1] D. Powell (Ed), "Delta-4: A Generic Architecture for Dependable Distributed Computing", ESPRIT Research Reports. Springer Verlag, Wien, New York, 1991.

[2] L. Lamport et al., "The byzantine generals problem", ACM Transactions on Programming Languages and Systems, 4(3):382–401, July 1982.

[3] M. Barranco et al., "An Architecture for Highly Reliable Fault-Tolerant Adaptive Distributed Embedded Systems", Computer, 53(3):38-46, March 2020.

[4] S. Poledna, "Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism", Kluwer Academic Publishers, MA, USA, 1996.

[5] Ö. Babaoglu et al., "Reliable broadcast and communication models: Tradeoffs and lower bounds", Distributed Computing,(2):177–189, 1988.

[6] A. Avizienis, "The N-Version approach to fault-tolerant software", IEEE Transactions on Software Engineering, 11(12):1491–1501, 1985.