Experimental Evaluation of Network Component Crashes and Trigger Message Omissions in the Flexible Time-Triggered Replicated Star for Ethernet

David Gessner, Alberto Ballesteros, Andreu Adrover, and Julián Proenza
DMI, Universitat de les Illes Balears, Spain
{david.gessner, a.ballesteros, julian.proenza}@uib.es

***Abstract*—A distributed embedded system (DES) is made up of a set of computing nodes interconnected by a network. If we want the DES to continue to operate even if a subset of its network elements fail, the network must be fault-tolerant. In particular, this requires that the architecture of the network provides redundant paths between nodes and that any elements critical for the operation of the network are replicated. In the context of DES that must not only be highly reliable, but also provide sufficient flexibility to adapt to unpredictable requirement changes, the Flexible Time-Triggered Replicated Star for Ethernet (FTTRS) has been proposed. One of the core features of FTTRS is precisely its fault-tolerant network architecture. In this paper we present a proof-of-concept prototype of FTTRS and a series of fault-injection experiments. These experiments show that FTTRS can tolerate the crash of any single network element, as well as the crash of various combinations of multiple network elements. A variety of omission failures affecting the most critical FTTRS message (called the trigger message) are also tolerated.**

I. Introduction

A distributed embedded system (DES) must be both highly reliable and flexible if we want it to operate continuously in an environment that imposes unpredictable requirement changes. For this, it requires an underlying network that is reliable and flexible as well. The goal of the *Flexible Time-Triggered Replicated Star for Ethernet* (FTTRS) [1] is to provide such a network for a project called *Fault Tolerance for Flexible Time-Triggered Ethernet-based systems* (FT4FTT), which brings 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]. This paradigm follows the master/multi-slave model 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 an arbitrary number of FTT slaves connected to two interconnected custom Ethernet switches, each of which embeds an FTT master. Links connecting slaves to switches are called *slave links* and links interconnecting switches are called *interlinks*.

As in all FTT implementations, in FTTRS the communication time is divided into rounds called *elementary cycles* (EC) and in the beginning of each such EC a schedule is transmitted that tells the slaves what synchronous (periodic) messages should be transmitted. This schedule is calculated online based on the current real-time requirements, e.g., deadlines

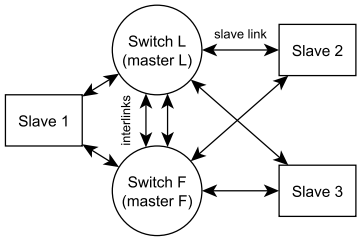

Fig. 1. FTTRS architecture.

and message transmission periods. These requirements may change during the operation of the system, which is what gives FTTRS, and in fact all FTT implementations, its flexibility. In FTTRS specifically, the schedule is calculated by each master (one in each switch) in a mutually consistent manner [3] and then broadcast by both switches simultaneously to the slaves in so-called *trigger messages* (TMs). To properly coordinate the simultaneous transmission of the TMs, a semi-active replication scheme is used: one of the switches, called the *leader*, is the reference that dictates the transmission instants, and the other, called the *follower*, synchronizes with the former [4]. For the correct real-time behavior of the system, it is critical that the schedule — and thus the TM — of each EC reaches every slave node and that it does so in a timely manner, even in the presence of transient faults in the communication channel. For this reason, each TM is proactively retransmitted by each switch multiple times at the beginning of the corresponding EC. The number of proactive retransmissions is termed the *TM redundancy level* and is given by a positive integer parameter that we denote by the letter k.

Regarding the use of the spatially redundant network by the slaves, each slave transmits each message through both its slave links simultaneously. Moreover, each switch forwards all slave messages to the other switch. As a result, in the absence of faults, each slave receives the same messages through each of its slave links.

This work-in-progress paper introduces a proof-of-concept prototype of FTTRS to show the viability of the FTTRS design. The prototype builds on previous ones that focused on assessing EC synchronization and that did not comprise the full FTTRS architecture: they either only had a single switch [5], or no slaves [4]. We then describe a series of fault-injection experiments that test that FTTRS can tolerate crash and omission failures of any single network component (i.e.,

 doi: 10.1109/WFCS.2015.7160583

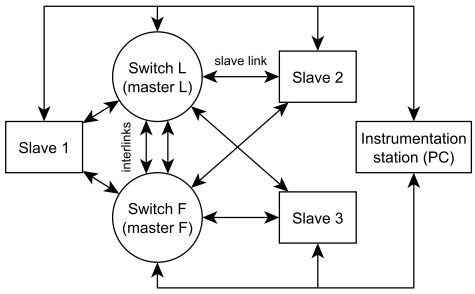

Fig. 2. Prototype testbed.

link or switch), as well as the crash or omission failure of various combinations of multiple network components. The crash of slaves or incorrect computations by them are not tested as they are out of the scope of FTTRS and are dealt with at other levels of the complete FT4FTT design [6]. More malicious failures are also not tested because the mechanisms to deal with them have not yet been implemented. However, such mechanisms do exist in the FTTRS design: in the design slave failure semantics are restricted to incorrect computations by means of port guardians [7] and the switches, with their internal masters, are forced to exhibit crash failure semantics by means of internal duplication and comparison [1].

The paper proceeds as follows. Section II describes the FTTRS prototype, together with some fault injection and logging facilities we added to properly conduct the experiments. Section III introduces the experimental evaluation: it describes the application running on the slaves during the experiments and continues with a description of these experiments. Finally, Section IV concludes the paper and points to future work.

II. The FTTRS prototype

Figure 2 shows the testbed setup. It consists of an FTTRS prototype with two switches and three slaves, extended by a series of Ethernet links connecting each of the switches and slaves to an instrumentation station.

The switches, slaves, and the instrumentation station are all implemented in commercial-of-the-shelf (COTS) personal computers (PCs). Each switch is implemented in a regular multi-core PC with a couple of Intel I350-T4 quad-port Ethernet server adapters. In contrast, each slave is built using hardware for embedded devices. Specifically, we acquired various Jetway JBC373F38-525-B barebones, each with an Intel Atom processor and four standard Ethernet network interface cards (NIC). This configuration provides sufficient ports to be interconnected as shown in Figure 2.

Figure 3 shows a block diagram of the internals of an FTTRS switch. The components that implement the FTTRS switch are the five Ethernet ports at the bottom of the figure and the *FTTRS switch module*, shown at the top left corner, which includes the *FTTRS master* and the *FTTRS switching module*. The ports are part of the NIC of the PC implementing the corresponding switch and the interface to these ports (i.e., the arrows shown immediately above them) is provided by the Linux kernel running on that PC. The FTTRS switch module is fully implemented in software and runs in the user space of the operating system (OS). Its code is based on a software

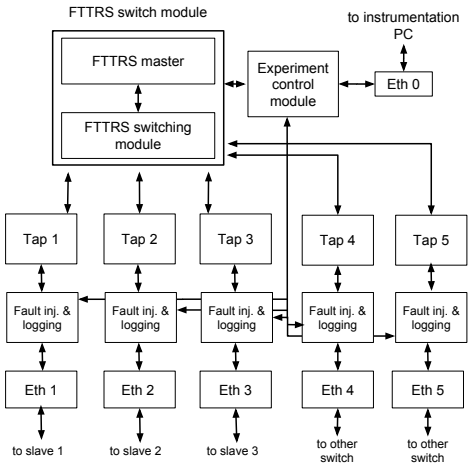

Fig. 3. Internals of an FTTRS prototype switch.

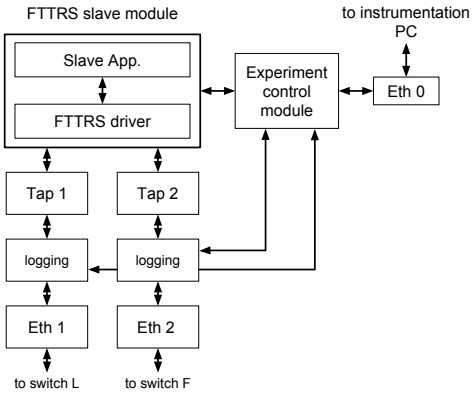

Fig. 4. Internals of an FTTRS prototype slave.

implementation of the HaRTES switch [8] [9]. The remaining components are not strictly part of FTTRS, but are extensions whose purpose is to facilitate the execution of the experiments. These components are a fault injection and logging module that we implemented in software for each of the links; several OS-provided virtual Ethernet interfaces (labeled *Tap* in the figure) that serve as an interface between the FTTRS switch module and each fault injection and logging module; and an experiment control module that allows to control the fault injection modules and the FTTRS switch module, as well as to retrieve any generated logs.

Figure 4 shows a block diagram of the internal structure of a prototype slave. Just like a switch, a slave has FTTRS specific components and additional components that make it easier to execute the experiments. The FTTRS specific components are the following. First, a software-implemented *FTTRS slave module*, which includes the application to be executed by the slave as well as an FTTRS driver that provides to the application the primitives necessary for transmission and reception of frames through the redundant FTTRS network. Second, two Ethernet ports, one for each switch. Regarding components that facilitate the execution of experiments, we again have an experiment control module connected to the instrumentation PC that allows to control the FTTRS slave module, as well as two logging modules that we implemented

and placed between the physical Ethernet ports and the FTTRS slave module with the help of two OS-provided virtual Ethernet *Tap* interfaces. Fault injection is not performed at the slave end, but only at the switch end. This does not restrict the set of faults that we can inject according to our fault model: for switch failures the switch end is the right place to inject the fault, and for link failures and message omissions either end of a link is adequate to inject faults.

III. Experimental evaluation of the prototype

This section first describes the application that we executed on our three prototype slaves and then the experiments we performed to verify that the FTTRS prototype correctly handles crash and omission failures occurring in switches and links. The logging information gathered during each experiment was stored and is available at [10].

A. The slave application

The application running on the slave nodes is a simplified replicated control application. All slaves perform the same task synchronously and then agree upon the result of the control. The operation of each slave can be divided into three phases. In the first phase each slave generates a value. This value might have come from a sensor, but to keep things simple, we opted for just using a simple counter. In the second phase this value is exchanged among the three nodes. Finally, the nodes execute a voting algorithm to reach a consensus on the value. The voting is a simple majority vote. Each phase is executed in a different EC, which means that the exchange of the counter values is scheduled by the masters every three ECs.

Since in this paper our focus is on the network, we inject faults into network components only (switches and links) and test that, despite of this, each slave is able to provide the value of its counter to the other two slaves. If so, we consider the fault to have been correctly tolerated.

B. Testing the tolerance to switch crash failures

The first two experiments consisted in provoking the crash of one of the switches. In the first experiment we made the leader crash and in the second the follower. In both cases the crash was only provoked after the initialization phase [4] that synchronized the TM transmissions of the two switches.

By inspecting the logs captured in the slaves we observed that, despite the crash of either switch, all slaves agreed upon the number of elapsed ECs and each of them was always able to share its counter value with the others. We therefore conclude that the crash of either switch was correctly tolerated.

C. Testing the tolerance to permanent omissions in links

The goal of this fault injection campaign was to verify that the FTTRS network architecture tolerates faults in slave links and interlinks that manifest as the permanent omission of any messages whose transmission is requested through them, i.e., that it tolerates the crash of links. This was achieved by having the fault injectors in the switches logically disconnect the different links in all the possible combinations that can be tolerated by the FTTRS network architecture — with the exception of a crash of all interlinks, which can be tolerated by FTTRS, but not by the current preliminary prototype. For practical reasons these injected faults were actually not permanent, but instead shortened to at least 10 ECs. This allowed us to logically reconnect a link and try another combination of link failures without having to reinitialize all slaves and switches. Nevertheless, having injected faults that last longer than three ECs can be considered equivalent to a permanent fault. This is so because a fault of more than three ECs already prevents the successful message exchange between slaves.

TABLE I. Permanent message omissions in links (one link at a time).

Target links	Affected ECs
l_{1L}	70–89
l_{1F}	90–109
l_{2L}	110–129
l_{2F}	130–149
l_{3L}	150–169
l_{3F}	170–189
l_{LF_1}	190–209
l_{LF_2}	210–229

In our prototype with two switches and three slaves, there are two links — one for each switch — per slave, giving a total of six slave links. In addition, there are two interlinks interconnecting the switches. We therefore have a total of 8 links and therefore 8 ways that a single link can suffer a permanent fault. We tried all 8 combinations of disconnecting one link at a time and disconnected each one of them for a duration of at least 10 ECs, as shown in Table I. In the tables l_{ij_n} indicates the link number n between slave or switch i and switch j — if there is only one link between i and j, then n is omitted. All 8 fault scenarios were tolerated.

After this experiment we proceeded with another one where we disconnected two links at a time. There are a total of $\binom{8}{2} = 28$ combinations for doing this. Of these 28 combinations, 24 can be tolerated such that communication is still possible among all three slaves. (The four combinations that do not allow all three slaves to communicate is when both links of the first slave fail, both links of the second slave fail, both links of the third slave fail, and both interlinks fail.) We tried these 24 combinations, as shown in Table II, and verified that the faults were in fact tolerated.

Finally, there are total of $\binom{8}{3} = 56$ combinations in which three slave links can crash at the same time. Of these, 24 still allow all slaves to communicate. We tried all these combinations, as shown in Table III, again verifying that they were all tolerated thanks to the redundant paths provided by the network topology of FTTRS.

D. Injecting transient faults into trigger messages

Apart from the links and the switches with their embedded masters, another critical element for the correct operation of any FTT-based communication infrastructure are the TMs. As we said in the introduction, in FTTRS each trigger message is proactively retransmitted k times by each master in each EC. Since TMs are also exchanged between the switches and then forwarded to the slaves, each switch transmits its own TM k times and the TM from the other switch another k times. This means that in each EC $2k$ TMs are transmitted through each

TABLE II. PERMANENT MESSAGE OMISSIONS IN LINKS (TWO LINKS AT A TIME).

Target links	Affected ECs	Target links	Affected ECs
l_{1L}, l_{2L}	70–89	l_{2L}, l_{3L}	310–329
l_{1L}, l_{2F}	90–109	l_{2L}, l_{3F}	330–349
l_{1L}, l_{3L}	110–129	l_{2L}, l_{LF_1}	350–369
l_{1L}, l_{3F}	130–149	l_{2L}, l_{LF_2}	370–389
l_{1L}, l_{LF_1}	150–169	l_{2F}, l_{3L}	390–409
l_{1L}, l_{LF_2}	170–189	l_{2F}, l_{3F}	410–429
l_{1F}, l_{2L}	190–209	l_{2F}, l_{LF_1}	430–449
l_{1F}, l_{2F}	210–229	l_{2F}, l_{LF_2}	450–469
l_{1F}, l_{3L}	230–249	l_{3L}, l_{LF_1}	470–489
l_{1F}, l_{3F}	250–269	l_{3L}, l_{LF_2}	490–509
l_{1F}, l_{LF_1}	270–289	l_{3F}, l_{LF_1}	510–529
l_{1F}, l_{LF_2}	290–309	l_{3F}, l_{LF_2}	530–549

TABLE III. PERMANENT MESSAGE OMISSIONS IN LINKS (THREE LINKS AT A TIME).

Target links	Affected ECs	Target links	Affected ECs
l_{1L}, l_{2L}, l_{3L}	70–89	l_{1F}, l_{2F}, l_{3L}	310–329
l_{1L}, l_{2L}, l_{3F}	90–109	l_{1F}, l_{2F}, l_{3F}	330–349
l_{1L}, l_{2L}, l_{LF_1}	110–129	l_{1F}, l_{2F}, l_{LF_1}	350–369
l_{1L}, l_{2L}, l_{LF_2}	130–149	l_{1F}, l_{2F}, l_{LF_2}	370–389
l_{1L}, l_{2F}, l_{3L}	150–169	l_{2L}, l_{3L}, l_{LF_1}	390–409
l_{1L}, l_{2F}, l_{3F}	170–189	l_{2L}, l_{3L}, l_{LF_2}	410–429
l_{1L}, l_{2F}, l_{LF_1}	190–209	l_{2F}, l_{3L}, l_{LF_1}	430–449
l_{1L}, l_{2F}, l_{LF_2}	210–229	l_{2F}, l_{3L}, l_{LF_2}	450–469
l_{1F}, l_{2L}, l_{3L}	230–249	l_{2L}, l_{3F}, l_{LF_1}	470–489
l_{1F}, l_{2L}, l_{3F}	250–269	l_{2L}, l_{3F}, l_{LF_2}	490–509
l_{1F}, l_{2L}, l_{LF_1}	270–289	l_{2F}, l_{3F}, l_{LF_1}	510–529
l_{1F}, l_{2L}, l_{LF_2}	290–309	l_{2F}, l_{3F}, l_{LF_2}	530–549

slave link. As long as at least one of the $2k$ TMs reaches each slave, all the slaves will agree on the number of elapsed ECs and which messages have been scheduled for each of them. As a result, the $2k$ TMs transmitted on a given link can be corrupted (leading to an omission) in $2^{2k}-1$ ways such that at least one TM gets through. If we have s slaves with two links each, then there are a total of $(2^{2k}-1)^{2s}$ possible scenarios of corrupting TMs of a given EC in all links such that still one TM gets through each slave link. If each slave has only one link remaining, then there are a total of $(2^{2k}-1)^{s}$ possible scenarios. All of them should be tolerated by FTTRS.

In order for our experimentation to be feasible and take a reasonable amount of time, we did two experiments. (1) One experiment with $k = 2$, two switches, and two slaves connected to both switches. In each EC we executed a different fault injection out of the $(2^4-1)^4 = 50625$ possible ones. (2) Another experiment with $k = 3$, three slaves, and two switches, but with all slaves only connected to one of the switches. In each EC of this second experiment we executed a different fault injection out of the $(2^6-1)^3 = 250047$ possible ones.

We verified that in both experiments all of the involved slaves agreed upon the number of elapsed ECs and that they correctly transmitted the messages scheduled for each EC. We conclude from this that all injected faults were tolerated by the FTTRS prototype.

IV. CONCLUSIONS AND FUTURE WORK

We presented a first proof-of-concept prototype of FTTRS to show the viability of its design. Using this prototype, we then performed a series of fault injection experiments to verify that the prototype correctly tolerates the crash and omission failures that according to the design it should tolerate. Specifically, that it tolerates the crash of either switch, the crash of any single link, the crash of two or three links that still result in a connected network topology, and the omission of multiple TMs per EC.

Future work includes implementing port guardians for the FTTRS switches and testing that they allow the prototype to prevent the propagation of errors caused by malicious behaviors of the nodes, such as impersonations and babbling idiots, that can impact the whole network; implementing tolerance to a crash of all interlinks and performing fault injection experiments to verify this tolerance; implementing and testing a prototype that integrates FTTRS with the other parts of the FT4FTT architecture, such as the node replication and voting mechanism; and perform further enhancements to the prototype to improve its timing behavior — which in particular includes reducing the jitter when frames are transmitted.

ACKNOWLEDGMENT

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 Tech. & 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. Ballesteros, J. Proenza, D. Gessner, G. Rodriguez-Navas, and T. Sauter, "Achieving Elementary Cycle Synchronization between Masters in the Flexible Time-Triggered Replicated Star for Ethernet," in *Proc. 19th IEEE Int. Conf. on Emerging Tech. and Factory Automation (ETFA)*, Barcelona, Spain, Sep. 2014.

[5] D. Gessner, I. Alvarez, A. Ballesteros, M. Barranco, and J. Proenza, "Towards an Experimental Assessment of the Slave Elementary Cycle Synchronization in the Flexible Time-Triggered Replicated Star for Ethernet," in *Proc. 19th IEEE Int. Conf. on Emerging Tech. and Factory Automation (ETFA)*, Barcelona, Spain, Sep. 2014.

[6] S. Derasevic, M. Barranco, and J. Proenza, "Appropriate consistent replicated voting for increased reliability in a node replication scheme over FTT," in *Proc. 19th IEEE Int. Conf. on Emerging Tech. and Factory Automation (ETFA)*, Barcelona, Spain, Sep. 2014.

[7] A. Ballesteros, D. Gessner, J. Proenza, M. Barranco, and P. Pedreiras, "Towards Preventing Error Propagation in a Real-Time Ethernet Switch," in *Proc. 18th IEEE Int. Conf. on Emerging Tech. and Factory Automation (ETFA)*, Cagliari, Italy, Sep. 2013.

[8] R. Santos, "Enhanced Ethernet switching technology for adaptive hard real-time applications," Ph.D. dissertation, Universidade de Aveiro, 2010.

[9] "Public repository of the flexible time-triggered communications." [Online]. Available: http://paginas.fe.up.pt/ ftt/sections/Repository

[10] "Log files for all the experiments." [Online]. Available: http://srv.uib.es/experimental-evaluation-of-fttrs/