©2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Improving Maintenance of FT4FTT: Extending it to Monitor and Log its Available Redundancy via Internet

Manuel Barranco, Adel Zendouh, Alberto Ballesteros, Julián Proenza
DMI, Universitat de les Illes Balears, Spain
Université Constantine 2 - Abdelhamid Mehri, Algeria
manuel.barranco@uib.es, zendouhadel@gmail.com, a.ballesteros@uib.es, julian.proenza@uib.es

***Abstract*—The FT4FTT project aims at proposing a complete Fault-Tolerant (FT) architecture for Real-Time (RT) critical adaptative Distributed Embedded Control Systems (DECSs) based on Ethernet. FT4FTT tolerates permanent faults in the channel and nodes by using a duplicated Flexible Time-Triggered (FTT) Switched Ethernet star and active replication of the nodes. It also includes mechanisms for node replicas to diagnose and reintegrate after temporary faults affecting the channel or their internal circuitry. However, FT4FTT has no mechanism to deal with channel and node redundancy attrition provoked by permanent faults. This paper presents our ongoing work to extend FT4FTT to both monitor/log its available redundancy, and to remotely access this information via Internet. This will allow to carry out proper maintenance actions, for instance, to timely restore the adequate redundancy level, forecast repairs, and assess the flexibility of the FT mechanisms of adaptative systems.**

I. Introduction

Providing high reliability for Real-Time (RT) Adaptative Distributed Embedded Control Systems (DECSs) is becoming a fundamental issue, as the criticality and the variability of the operational conditions of the systems under control are increasing. The FT4FTT project is devoted to proposing a complete Fault-Tolerant (FT) architecture to provide high reliability for RT critical adaptative DECSs.

FT4FTT is based on the Flexible Time-Triggered switched Ethernet (FTT-Ethernet) protocol [1]. FTT-Ethernet is a master/multislave approach that provides flexible RT communication, but that has no Fault-Tolerance (FT) mechanism. Thus, an FT4FTT control system includes mechanisms to tolerate faults affecting the links, the switch, the master and the slaves (nodes). As it will be explained later, F4FTT provides fault tolerance basically by means of hardware and time redundancy, e.g. by actively replicating the switches, the links, the critical slaves and the transmission of critical messages.

Moreover, FT4FTT provides each slave replica with mechanisms [2] to both diagnose temporary faults affecting its communication and/or operation, and reintegrate to prevent these faults from leading the replica to be perceived as permanently faulty. This is essential to take advantage of the redundancy, since it prevents temporary faults from unnecessarily causing *redundancy attrition*.

Nevertheless, FT4FTT still includes no mechanism to monitor/log the available redundancy of the system and to adequately maintain it. This an important limitation, since the channel/node redundancy attrition caused by permanently-faulty components that are not repaired necessarily reduces the system reliability. In other words, when FT is based on the use of redundancy, a decrease in the available redundancy compromises the system reliability.

Likewise, timely monitoring the available redundancy can be fundamental not only to carry out repairs that preserve an adequate level of reliability; but also to anticipate adequate actions to deal with a system failure, when the redundancy is about to exhaust and cannot be restored. Furthermore, if appropriate data about the available redundancy is logged, then these data can be analyzed a posteriori to enhance or optimize the system itself, e.g. to assess the quality of the different components, or to forecast potential threats and anticipate repair actions.

Monitoring/logging becomes specially important for adaptative systems, where the variability of the environment can exceed the predicted unreliability the FT mechanisms are designed for, e.g. if the maximum expected bit-error rate is exceeded then the planned number of message retransmissions may not suffice. Monitoring can help in timely detecting limit situations before the system fails; whereas logging can be used to evaluate the FT mechanisms and then improve their capacity for adapting to different situations.

Finally, we are interested in accessing, remotely through Internet, the data about the available redundancy of an FT4FTT control system. This will allow to monitor the redundancy by means of remote devices, placed almost anywhere, that can implement different strategies, e.g. alarms, to reduce the time to repair. Also, it will enable retrieving the logged data online to store and process it by adequate machines, e.g. with high storage and computational capacity, external to FT4FTT.

This paper presents our on-going work towards extending FT4FTT with mechanisms and devices to remotely monitor and log its available redundancy; mechanisms and devices that, then, can be used to attain the above-mentioned advantages.

The paper outlines the basics of FT4FTT. Then, it summarizes the design decisions about what is the data that characterizes the available redundancy, as well as how we have extended the FT4FTT architecture and its mechanisms to monitor/log these data and make them remotely accessible. Afterwards, it outlines the current prototype and a set of experiments to test it. Finally, it highlights the main conclusions and future work.

II. Basics of an FT4FTT control system

As it is depicted in the bottom left corner of Fig. 1, in FT4FTT each slave (node) communicates through two full-

978-1-5090-1314-2/16/$31.00 ©2016 IEEE

duplex HaRTES FTT-Ethernet switches that are internally duplicated and compared [3]. Since each HaRTES switch embeds an FTT Master, we will use the term *switch* and *master* interchangeably. Also note that in FT4FTT each switch includes a set of *Slave Error Counters* (SECs) per slave, to log the errors sent by each slave, e.g. untimely frames, and then disconnect the switch's port of any faulty slave [4].

Communication takes place in rounds, called *Elementary Cycles* (ECs) [1], masters cooperatively control. Specifically, both masters trigger each EC by quasi-simultaneously broadcasting the so called *Trigger Message* (TM), which indicates what messages must be transmitted, and by whom, in the EC.

Each critical slave is actively replicated [5]. Fig. 1 shows 3 slave replicas, although FT4FTT can include more than 1 replicated slave and, also, non-replicated ones. Each slave (each replica, Rep_i, in Fig. 1) transmits/receives to/from each switch, Switch_j, using a dedicated link, L_{ij}. The switches exchange their incoming traffic via multiple interlinks, I_is, and then forward the frames to the slaves. This provides redundant paths between each pair of slaves and, also, allows both masters to have a consistent view of the communications, which helps them to be replica determinate [6].

The replicas of a given slave perform exactly the same application, in parallel, and coordinate among them to take the right control decisions. The application is split into segments, which are mapped onto several ECs [2]. After each segment, the replicas exchange the results of the segment. Then, they vote locally and consistently to obtain a consensus segment result that each one of them uses to compute the next segment.

In each segment the masters evaluate each replica's ability to correctly transmit/receive the segment results. Based on this, the masters consistently increase/decrease a *Communication Error Counter* (CEC) for each replica. Also, after each segment, the masters provide each replica with information about this replica's ability and, then, each replica uses it to increase/decrease a local CEC (LCEC). Moreover, each replica manages a local *Discrepancy Error Counter* (DEC) to assess its ability to correctly vote [2]. The LCEC and DEC are used by the replica to diagnose itself as faulty. When so, the replica resumes its operation to tries to fix the fault and reintegrate with non-faulty replicas. Note that if a replica fails in diagnosing itself as faulty, the masters can force a watchdog timer located at the replica to resume it [2].

III. Design rationale

A. Characterization of the available redundancy

The FT4FTT's available redundancy is characterized by certain data concerning the status of the switches/masters, links, interlinks, and slaves.

As regards the switches/masters, we are interested in the *Master Operational Status* (MOS). Since each switch/master is internally duplicated and compared, a faulty switch/master simply does not communicate. Thus, the MOS is *up* or *down*.

What are the available paths depends on the available switches and, also, on the available links and interlinks. Thus, we characterize each link and interlink by the *Link Communication Status* (LCS) and the *Interlink Communication Status* (ICS) respectively. Since a faulty link/interlink can only crash or syntactically corrupt frames, which are then automatically discarded at the receiver, the LCS/ICS can only be *up* or *down*.

Concerning the slaves, first we monitor/log the *Slave Communication Status* (SCS) of each slave, i.e. whether each slave communicates or not (*up* or *down*) through any of its links, no matter which link. Second, to have a more accurate vision of the communication capabilities of each slave, we monitor/log the *Slave Error Counters* (SECs) each switch manages to diagnose what kind of fault each slave manifests (see Section II). We refer to the pair of SECs sets devoted to a given slave to as the *Slave Error Status* (SES) of that slave.

Moreover, when an slave is an slave replica, it is important to have extra information about the following aspects: (1) its ability to exchange the segment results for voting; (2) its ability to diagnose itself as suffering from faults that prevent it from exchanging these results; (3) and its ability to both vote correctly and diagnose itself as faulty when it cannot do so. For this purpose, we respectively monitor/log the following data about each replica (see Section II): (1) the CEC both switches consistently manage to assess the replica's capacity for exchanging the segment results, i.e. the *Replica Communication Error Status* (RCES); (2) the replica's local CEC (LCEC), i.e. the *Replica Local Communication Error Status* (RLCES); and (3) the replica's DEC, i.e. the *Replica Discrepancy Error Status* (RDES).

B. Monitoring and logging architecture

Fig. 1 sketches in grey-filled boxes the new elements of the architecture we designed to access and make remotely available the data that characterizes the available redundancy.

As explained later, the data about the available redundancy is encapsulated within frames that are periodically sent through both switches to the *Monitoring Server* (MSe). The *Message Gatherer* (MG) within the MSe de-encapsulates these data and transfers them to the *Event Process* (EP), which interprets them and populates the data base *Event DB* (EDB). Note that to precisely characterize the available redundancy, it must be monitored/logged with a time resolution of the order of few ECs, e.g. few milliseconds. Thus, to timely and compactly monitor/log this huge quantity of data, the EP only stores in the EDB *Redundancy Status Events* (RSEs), i.e. changes in the available redundancy. Finally, the MSe includes a set of *Redundancy Status Publishers* (RSPs). Each RSP provides a given type of application with remote access to the EDB. For instance, an RSP could be used to transfer/retrieve historic data about the available redundancy to/from an external storage.

Fig. 1 sketches an example in which an RSP is a *CoAP (Constrained Application Protocol) Server* that provides light M2M communication for accessing the EDB from remote devices. It shows a *Monitoring Station* (MSt), e.g. a workstation or a mobile device, that accesses the data about the available redundancy through the Web and then, displays or analyzes it. To pre-process the data in a specific manner and to minimize the communication load when accessing the MSe, the MSt interacts with an intermediary *Web Server* (WSe). Basically, within the WSe, there is a *HTTP Server* that executes server-side applications that interpret the HTTP requests sent by the

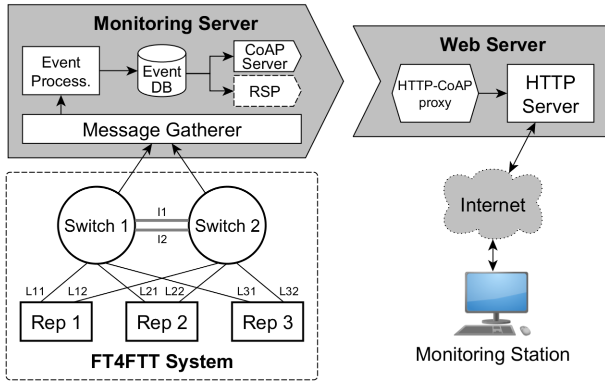

Fig. 1: System Architecture

MSt and, then, exchange the necessary messages with the *CoAP Server* in the MSe via a *HTTP-CoAP Proxy*.

C. FT4FTT new monitoring/logging mechanisms

To gather the data that characterizes the available redundancy and to send them to the MSe, we extended FT4FTT with a set of additional mechanisms.

The first one is the *Slave Communication Status Message* (SCSM) *procedure*, which is used for the masters to acquire the ICSs, LCSs, SCSs and RLCESs (see Section III-A). To store these data each master includes the next vectors: the *Interlink Communication Status Vector* (ICSV), the *Link Communication Status Vector* (LCSV), the *Slave Communication Status Vector* (SCSV), and the *Replica Local Communication Error Status Vector* (RLCESV). Each one of the elements of these vectors respectively stores the ICS of a given interlink, the LCS of a given link, the SCS of a given slave (or slave replica), and the RLCES of a given slave replica. Note that the LCSV of a master includes not only an element for each one the links connected to the master's switch, but also for each one of the links connected to the other master's switch.

The SCSM procedure is executed every EC. It consists in slaves (both slaves and slave replicas) and masters exchanging *Slave Communication Status Messages* (SCSMs) to populate the just mentioned vectors. In a first step, each slave transmits to both masters, at the beginning of the EC, an SCSM that piggybacks its identifier and, in the case of a slave replica, also its RLCES (the value of its LCEC). The SCSM is used as an *I am alive message*; so that when a master receives an SCSM through a given link from the corresponding slave, it sets to *up* both the element of the LCSV that represents the link, and the element of the SCSV that represents the slave.

In a second step, each master retransmits all the SCSMs to the other master through all interlinks. When each master receives an SCSM from the other one, it sets to *up*: (1) the element of its ICSV that registers that it is possible to receive through the corresponding interlink, (2) the element of its LCSV that represents the other switch's link where the SCSM was originally received, and (3) the element of its SCSV that represents the slave (or slave replica) that generated the SCSM. Note that by updating the element (3) the master obtains a correct view of the SCS, even if a fault prevented it from receiving the SCSM directly from the slave in the first step. Additionally, if the slave that sent the original SCSM ($SCSM_{or}$) in the first step is a replica, the master uses the RLCES piggybacked in the SCSM retransmitted by the other switch ($SCSM_{rt}$) to update the position of the RLCESV that corresponds to that replica. Specifically, if the master did not receive the $SCSM_{or}$ directly from the replica in the first step, the master simply updates the RLCESV position with the RLCES piggybacked in the $SCSM_{rt}$. Otherwise, the master checks if the RLCES of $SCSM_{or}$ coincides with the RLCES of $SCSM_{rt}$. If they do not match, the master writes a special value in the position of the RLCESV that corresponds to the slave replica, to indicate that the replica sent a different RLCES to each master.

The second added mechanism is the *Master Report Message* (MRM) *procedure*. It consists in each master sending to the MSe a MRM that is twofold. First, it serves as an *I am alive message* of each master, i.e. it allows the MSe to know each master's MOS. Second, each master uses it to report to the MSe the values of the mentioned vectors, as well as the value of the SES and RCES it dedicates to each slave and slave replica respectively.

The last mechanisms is the *Replica Report Message* (RRM) *procedure*. It consists in each replica's application sending to the MSe, once per EC through both switches, an RRM reporting the value of the replica's DEC, i.e. its RDES. Note that the RRM is devoted to reporting application-dependant status data (the DEC so far) and, thus, it is independent from the MRM, which conveys communication-status data.

IV. Prototype

Taking as a basis the FT4FTT prototype of [7], we are building up a prototype of the system depicted in Fig. 1 to assess the feasibility, correctness and performance of the monitoring/logging mechanisms and architecture herein proposed. On the one hand, we extended the functionalities of the switches and the replicas to implement a first version of the SCSM, MRM and RRM procedures. So far this version allows gathering the LCS of each link, as well as the SCS, RCES and RDES of each one of the 3 slave replicas.

On the other hand, to simplify the implementation of the monitoring/logging architecture, we embedded the *Web Server* (WSe) into the *Monitoring Server* (MSe). Both of them are thus placed in the same device, which executes Linux and which is connected to both switches. We implemented the MG and the PE of the MSe in C. The MG consists of two concurrent threads, each of which de-encapsulates the data of the frames received from one of the switches. The PE interprets these data to detect any change in the available redundancy, i.e. any RSE (see Section III-B); inserts the RSEs into a queue to accommodate potential bursts of RSEs; and then populates the *Event DB* (EDB). The EDB is a MySQL data base that contains 2 tables. The first one stores basic information about the FT4FTT elements, e.g. type (link, replica, etc.), elements' identifiers, MAC addresses, etc. The second one stores the status of all of these elements for each RSE occurrence, i.e. timestamp, LCSs, SCSs, RCESs and RDESs. We implemented both the CoAP server and the HTTP-CoAP proxy in Java using the Californium framework, whereas the WSe is an Apache.

Fig. 2 shows some information the *Monitoring Station* retrieves from the WSe. At the top there are the status (*up*

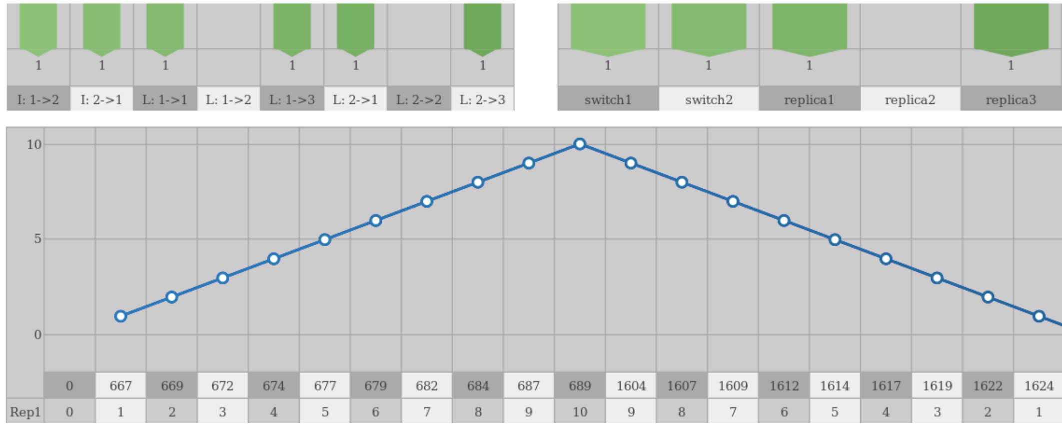

Fig. 2: Information shown in the Monitoring Station.

in green and *down* in grey) of the links (LCSs) and the slaves (SCSs). The bottom shows the RCES of one replica (one replica's CEC) in different ECs.

V. Testing the System

Next we summarize two of the fault injection tests we are carrying out to asses the proposed mechanisms and architecture. The first test checks the ability of the system to monitor and log the status of the slave replicas and links, i.e. the SCSs and LCSs. Specifically, we inspect the content of the EDB after disconnecting different link combinations. Each row of Table I show the status of the three replicas and their links, as stored in the EDB, after injecting some of these combinations. The column headers label the replicas and links as in Fig. 1. In row 1, when no link is disconnected, all the slaves and their links are *up*. In row 2 we disconnect link 1 from replicas R_1 and R_2, i.e. L_{11} and L_{21}. As shown, the system detects the new event and updates the EDB. Finally, in row 3 we simulate the failure of R_2 by disconnecting all its links. This row shows that the Monitoring Server detects not only the fault of the links, but also the inability of R_2 to communicate.

	R1	R2	R3	L11	L12	L21	L22	L31	L32
1	UP	UP	UP	UP	UP	UP	UP	UP	UP
2	UP	UP	UP	**DW**	UP	**DW**	UP	UP	UP
3	UP	**DW**	UP	**DW**	UP	**DW**	**DW**	UP	UP

TABLE I: Database for the SCSM test.

The second test checks the ability of the system to monitor and log the RCESs and RDESs, i.e. the value of the CECs and DECs. Specifically, we inject a fault to prevent one of the replicas from correctly receiving segment results during a limited number of consecutive segments. This leads the switches to increase the replica's CEC, since the replica cannot receive; and the replica to increase its DEC, as it has not enough segments results to consistently vote. Once we stop injecting the fault, both the CEC and DEC decrease, as the replica can correctly receive and vote from then on. The evolution of the CEC that results from this test is shown at the bottom of Fig. 2.

VI. Conclusions and Future Work

FT4FTT provides fault-tolerance mechanisms based on the use of hardware and time redundancy to attain high reliability in critical adaptative DECSs. However, it does not include any mechanism to monitor/log its available redundancy. This impedes to carry out adequate maintenance actions to prevent permanent faults from reducing the FT4FTT redundancy level and, thus, from compromising the system reliability throughout its operation. Herein we explain the design, a first prototype implementation, and some tests of a set of new mechanisms and devices we have added to FT4FTT to both monitor/log its available redundancy and, then, to remotely access this information via Internet. This will enable the addition of advanced maintenance mechanisms; for example, mechanisms that assess the response of adaptative systems that operate in highly-variable environments, in order to improve their capacity of adaptation, or to timely detect limit situations.

As future work we plan to finish the implementation of all the mechanisms and, then, to explore the possibility of integrating them into an Industrial Internet of Things (IIoT) framework, e.g. OPCA UA, DDS.

Acknowledgments

This work was supported by grants DPI2011-22992 and TEC2015-70313-R funded by the Spanish Ministerio de Economia y Competitividad (MINECO) and by the Fondo Europeo de Desarrollo Regional (FEDER).

References

[1] P. Pedreiras, L. Almeida, and P. Gai, "The FTT-ethernet protocol: Merging flexibility, timeliness and efficiency," in *24th Euromicro Conf. on Real-Time Systems*. IEEE Computer Society, 2002.

[2] S. Derasevic, M. Barranco, and J. Proenza, "Designing fault-diagnosis and reintegration to prevent node redundancy attrition in highly-reliable control systems based on FTT-Ethernet," in *World Conference on Factory Communication Systems (WFCS), 2016 IEEE.*

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

[4] A. Ballesteros, D. Gessner, J. Proenza, M. Barranco, and P. Pedreiras, "Towards Preventing Error Propagation in a Real-Time Ethernet Switch," in *Emerging Technology and Factory Automation (ETFA), 2013 IEEE.*

[5] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, "Understanding replication in databases and distributed systems," in *Distributed Computing Systems, 2000. Proc. 20th Int. Conf. on*. IEEE.

[6] S. Poledna, "The problem of replica determinism," in *Fault-Tolerant Real-Time Systems*. Springer, 1996, vol. 345.

[7] A. Ballesteros, S. Derasevic, D. Gessner, F. Francisca, I. lvarez, M. Barranco, and J. Proenza, "First Implementation and Test of a Node Replication Scheme on top of the FTTRS," in *Proc. 12th IEEE World Conf. on Factory Comm. Systems (WFCS)*, 2016.