A Proposal for Managing the Redundancy Provided by the Flexible Time-Triggered Replicated Star for Ethernet

David Gessner, Julián Proenza, Manuel Barranco
DMI, Universitat de les Illes Balears, Spain
davidges@gmail.com, {julian.proenza, manuel.barranco}@uib.es

Abstract

Distributed embedded systems that must operate continuously and satisfy unpredictable requirement changes must be reliable and flexible. Flexibility for changing real-time requirements may be provided by Hard Real-Time Ethernet Switching (HaRTES), an implementation of the master/multi-slave Flexible Time Triggered (FTT) communication paradigm. HaRTES relies on a single switch that embeds the FTT master and thus constitutes a single point of failure. Moreover, the trigger messages (TMs), which are periodic polling messages sent by the master to the slaves, are critical for the correct operation of any FTT implementation. Nevertheless, HaRTES has no mechanism to prevent their loss. To provide high reliability through fault tolerance, the Flexible Time-Triggered Replicated Star (FTTRS) duplicates the HaRTES switch and master, and modifies the masters to proactively retransmit the TMs. This spatial and temporal redundancy requires an appropriate management by the slaves. This paper discusses how this management can be performed.

1. Introduction

If a distributed embedded system (DES) must operate continuously while satisfying unpredictable requirement changes, then it must be both highly reliable and flexible. In particular, this requires that the communication channel of the DES satisfies those attributes.

The goal of the *Flexible Time-Triggered Replicated Star for Ethernet* (FTTRS) [1] is to provide such a channel for systems with changing real-time requirements. It is part of the *Fault Tolerance for Flexible Time-Triggered Ethernet-based systems* (FT4FTT) project, which aims at providing high reliability and flexibility for all crucial parts of a DES.

FTTRS is based on *Hard Real-Time Ethernet Switching* (HaRTES) [2], a switched Ethernet implementation of the Flexible Time Triggered (FTT) communication paradigm [3]. HaRTES implements a micro-segmented star topology with the HaRTES switch as a central element. The switch provides the most relevant functions of FTT by embedding the FTT master, which polls multiple slaves by means of a single periodic message called *trigger message* (TM). The master thereby divides the communication time into rounds of fixed duration, where each round is initiated by a new TM transmission. These rounds are called *elementary cycles* (ECs) and are divided into a *synchronous window* followed by an *asynchronous window*. In each EC, the corresponding TM conveys a *schedule* that tells the slaves which messages to transmit during the synchronous window. During the asynchronous window, a slave can transmit asynchronous messages autonomously.

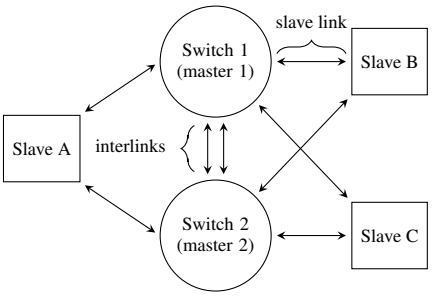

Figure 1. FTTRS architecture.

In HaRTES, the single switch with its embedded master is a single point of failure. Moreover, transient channel faults can easily prevent a slave from receiving a TM, even though these messages are critical for the correct operation of the FTT scheme implemented by HaRTES.

The single point of failure is eliminated through the FTTRS architecture [1] (see Figure 1). It has two HaRTES switches, each with its own master. The slaves are connected to both switches by means of *slave links* and there are redundant links between the switches called *interlinks*.

Regarding the TMs, we propose to always transmit each TM multiple times. This makes them significantly more robust to transient link faults, but without introducing non-deterministic delays in their delivery, as would be the case with on-demand retransmission schemes.

For FTTRS to function, the slaves must be prepared to manage the redundant channel provided by the duplicated switches, as well as the redundant TMs broadcast by each master. This paper discusses how the slaves perform this management.

Section 2 describes the fault model and a few assumptions. Section 3 proposes a mechanism for the slaves to manage the TM redundancy; whereas Section 4 discusses how slaves can manage the channel redundancy. Finally, Section 5 concludes the paper and points to future work.

 doi:10.1109/WFCS.2014.6837600

2. Fault model and assumptions

In FTTRS, the switches and the masters are fail-silent. This means that they have crash-failure semantics, i.e., they either provide a correct service or they remain silent permanently. This is enforced through internal duplication and comparison. Moreover, the masters are assumed to be replica determinate [4]. In FTTRS this means that they broadcast the same contents in their TMs and that they coordinate their TM broadcasts to occur simultaneously. The enforcement of this is currently under development.

Regarding the slaves, they have incorrect computation failure semantics in the value domain, i.e., they can transmit incorrect data in their messages, but they cannot transmit at arbitrary times. In particular, they are prevented from transmitting messages while the TMs are broadcast. This prevents the delivery of TMs being delayed by pending or in-progress slave transmissions, which is important for the TM redundancy management to work properly (see Section 3). The slave failure semantics are enforced using port guardians at the switches [5] and additional integrity checks to prevent two-faced behaviors (see Section 4.2).

We exclude from our fault model all software faults, i.e., the software is assumed to be fault free.

Finally, slave links and interlinks can suffer both permanent and transient faults. However, since all corrupted frames are dropped by the FTTRS switches and slaves (due to the frame check sequence in the Ethernet frames), the faults can only manifest as the loss of messages.

3. Managing the TM redundancy

As explained in the introduction, in FTTRS each TM is transmitted multiple times. Specifically, each EC starts with a *trigger message transmission window*, or *TM window* for short. During that window, each master broadcasts k replicas of the TM, where k is a function of the channel's bit error rate. Nothing else is transmitted on the slave links during the TM window. If k is well calculated, this ensures that each slave receives at least one TM replica from each master, even in the presence of transient link faults. With this, the delivery to the slaves of the data conveyed in the TMs is ensured: a slave can simply read the data from any of the TMs it receives in a given EC. However, TMs not only convey data, but are also used to synchronize the start of each EC among the slaves.

In HaRTES, the slaves use as the EC synchronization event the arrival time of the single TM of the corresponding EC. In FTTRS the synchronization mechanism is modified to work with the new TM transmission scheme.

Of the k TM replicas transmitted on each slave link, the corresponding slave might receive all k replicas or only a subset of them due to transient errors in the links. Regardless of which specific replicas each slave receives on each of its slave links, the time instants when the slaves consider each EC to start and end must align. This can be achieved by a simple mechanism if **(a)** k has a value that guarantees that each slave gets at least one TM replica if it has a link without permanent faults that is connected to a non-faulty master; **(b)** some additional information is conveyed in the TMs; **(c)** all TM replicas belonging to the same EC have the same size; **(d)** the TM broadcasts of the masters are coordinated as stated in the previous section, i.e., when one master broadcasts the ith TM replica, the other master does the same quasi-simultaneously, giving the appearance that the TM broadcasts occur in lockstep; **(e)** all TM replicas of the same EC are broadcast with the same constant time τ between the start of the broadcast of one replica and the next, where τ is greater than the sum of the TM transmission time and the Ethernet interframe gap (this is important to ensure that the TM transmissions are spaced uniformly in time even if the transmission time of some TM replicas is shortened due to transient errors); **(f)** the amount by which the clocks of non-faulty slaves can drift apart during one EC is negligible; and **(g)** the difference between propagation times on the slave links is negligible.

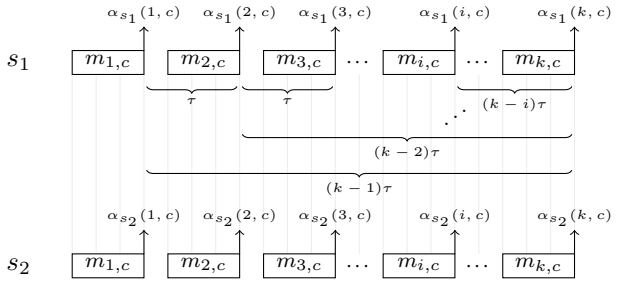

Figure 2. Alignment of TM arrival times.

As stated previously, satisfying (a) requires knowing the channel's bit error rate, or at least a pessimistic estimate. Regarding (b), the additional information is the value of k; a sequence number for each TM replica, which takes integer values from 1 to k; and the value of τ, which we call the *TM interarrival time*. The duration of an EC is also necessary, but this was already provided in the TMs used in HaRTES. Requirement (c) can be trivially achieved since the replicas contain the same data, except for the sequence number, whose code can be made to have the same length in all replicas. Requirements (d) and (e) are achieved through the replica enforcement mentioned in Section 2. Requirement (f) is realistic for typical EC durations (on the order of a few milliseconds). Finally, requirement (g) is realistic if the difference in the lengths of the slave links is not too excessive. If these requirements are satisfied, then the following EC synchronization mechanism can be used for the slaves.

Let S_c denote the set of slaves in a given FTTRS network that remain non-faulty at the end of EC c. Moreover, let $m_{i,c}$ denote the TM with sequence number i in EC c, where $i \in \mathbb{N}$ and $1 \leq i \leq k$. Also, let $M_{s,c}$ be the set of TMs that a slave $s \in S_c$ receives in EC c. Furthermore, let $\alpha_{s,l}(i,c)$ denote the time at which $m_{i,c} \in M_{s,c}$ is received by slave $s \in S_c$ on link $l \in \{l_1, l_2\}$, with $\{l_1, l_2\}$ denoting the set of slave links of s. Note that because of requirements (d), (e), and (g), if $m_{i,c}$ is received through

both l_1 and l_2, then $\alpha_{s,l_1}(i,c) \approx \alpha_{s,l_2}(i,c)$. In the following we therefore omit the link subscript, e.g., we simply write $\alpha_s(i,c)$. Figure 2 illustrates the TM arrival times for an EC c and two slaves $s_1, s_2 \in S_c$.

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

$$\alpha_s(k,c) = \alpha_s(i,c) + (k-i)\tau. \quad (1)$$

This is also illustrated in Figure 2 for slaves s_1 and s_2.

Note that (1) can only be calculated if i and c exist such that $m_{i,c} \in M_{s,c}$, i.e., if s received at least one TM replica in c. Since we are assuming requirement (a), this will be the case. Thus, (1) can be calculated for all $s \in S_c$ for each EC c. If in addition we have that

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

then we can synchronize the start of EC c among s_1 and s_2 using the TM arrival time $\alpha_{s_1}(k,c)$ as the synchronization event.

For (2) to be true it is enough for requirements (d), (e), and (g) to be satisfied. In that case, if there exists $m_{i,c} \in M_{s_1,c} \cap M_{s_2,c}$, then $\alpha_{s_1}(i,c) = \alpha_{s_2}(i,c)$, and (2) is obviously true. On the other hand, if there does not exist $m_{i,c} \in M_{s_1,c} \cap M_{s_2,c}$, there must still exist $m_{i,c} \in M_{s_1,c}$ and $m_{j,c} \in M_{s_2,c}$, where $i \neq j$, because of requirement (a). If $i < j$, then $\alpha_{s_2}(j,c) = \alpha_{s_1}(i,c) + (j-i)\tau$ and thus $\alpha_{s_2}(k,c) = \alpha_{s_1}(i,c) + (j-i)\tau + (k-j)\tau = \alpha_{s_1}(i,c) + (k-i)\tau = \alpha_{s_1}(k,c)$. Therefore (2) is again satisfied and the slaves can be synchronized. A similar argument can be made if $j < i$.

Because of requirement (f), the synchronization can be achieved in practice by having s_1 set a timer to expire after $(k-i)\tau$ units of time when it receives a TM $m_{i,c}$ at time $\alpha_{s_1}(i,c)$. When the timer expires, the slave would consider the EC c to start. Similarly, slave s_2 would set a timer to expire after $(k-j)\tau$ units of time when it receives a TM $m_{j,c}$ at time $\alpha_{s_2}(j,c)$, where j may have a different value than i. The timers of both slaves would expire at $\alpha_{s_1}(k,c) = \alpha_{s_2}(k,c)$, and they would consider c to start at the same time, thus having synchronized the start of c.

Regarding the synchronization of the end of an EC c among two slaves $s_1, s_2 \in S_c$, it can be achieved by having them each set a second timer to expire d units of time after the above described timer expires, where d is the duration of c, which is available to the slaves because of (b).

4. Managing the channel redundancy

The previous section described a mechanism that the slaves can use to manage the redundant TMs transmitted by the masters. This mechanism was designed to work even if some of the TMs are lost due to transient faults. This section focuses on how the slaves can manage the redundancy present in FTTRS to tolerate permanent faults, namely, the spatial redundancy of the channel.

4.1. Transmission and reception of messages

In FTTRS, if a slave needs to transmit a message, it does so through each of its two slave links. This simplifies transmissions substantially in presence of permanent faults: slaves use their links simultaneously all the time and therefore link or switch failures are masked instead of requiring an error treatment by the slaves. In other words, when permanent faults occur, slaves do not change the way in which they transmit messages.

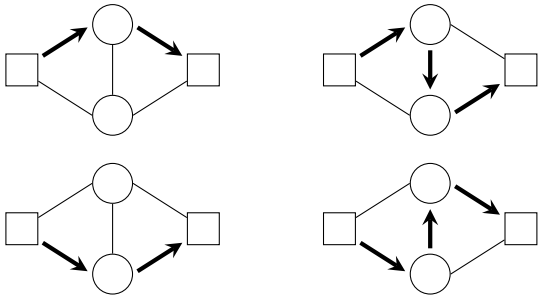

Figure 3. Replica radiation.

As to the reception of messages, in FTTRS all messages coming from a slave link and destined to other slaves are forwarded through the interlinks. This means that the two transmissions of a message by a transmitting slave, each occurring on a separate slave link of the transmitter, result in up to four messages being received by a recipient slave. This is illustrated in Figure 3. We call the phenomenon of two transmissions resulting in up to four receptions *replica radiation*.

Note that replica radiation makes sure that a pair of slaves communicate through the maximum number of possible paths between them. Since the switches have crash failure semantics and the slaves do not present two-faced behaviors (see Section 4.2), all copies of a message received by a slave due to replica radiation will be identical. Moreover, identical messages received in an EC must have originated from the same transmission, i.e., they are replicas of the same message, and therefore cannot be confused with messages from other transmissions. This is so because both synchronous and asynchronous messages are unique in each EC. Specifically, in FTTRS, as well as in other FTT implementations, no two identical synchronous messages are ever scheduled for the same EC. Asynchronous messages are also unique in each EC because in FTT they must have a minimum interarrival time that is an integer multiple of the EC length. To ensure the uniqueness of synchronous and asynchronous messages even if the slaves behave incorrectly, the port guardians at the FTTRS switches drop any message that violates the schedule or the minimum interarrival time of a message. Thus, a slave can be sure that if it receives the same message multiple times in an EC, then all copies correspond to the same transmission.

When it is time for the slave to deliver the message to the locally executing application, only a single copy of the received messages should be delivered. In this way, the channel redundancy is made transparent to the slave's application. Since all copies are identical, any one of them can be delivered by the slave. This maximizes the probability of a message transmitted by one slave being deliv-

ered by the intended recipients in the presence of permanent faults in links or interlinks.

With respect to the time of delivery, messages are not delivered as soon as the first copy is received. Instead, and according to a total order multicast service [6] that is also implemented by the FTTRS masters, and which is currently developed within the FT4FTT project under the name of ReTOPS, the message delivery should only be performed by a slave when instructed to do so by the masters. We refer the reader to [6] for the details on when deliveries should be performed according to this total order multicast service.

4.2. Preventing two-faced behaviors

As stated above, the management of the channel redundancy relies on the slaves not exhibiting two-faced behaviors. This calls for mechanisms to prevent this unwanted behavior.

In FTTRS, like in other FTT implementations, each of the messages that slaves exchange belongs to a *stream* [3]. Such a stream specifies certain real-time properties such as deadlines, periods, and minimum interarrival times, and other properties such as which slave is the transmitter for the messages of the stream. Streams also identify messages. This means that two messages transmitted in the same EC and identified as belonging to the same stream should be copies of each other and thus convey the same payload. Therefore, to prevent one slave from presenting a two-faced behavior to the others it is necessary to prevent the transmission of two messages identified as belonging to the same stream, but which carry different payloads. Note that we do not worry about a slave transmitting messages with invalid stream identifiers since such messages would be dropped by the port guardians at the switches.

According to our fault model we only consider two-faced behaviors caused by hardware faults. A hardware fault can cause a slave to exhibit a two-faced behavior by corrupting a payload before it is transmitted on one link, but after it was transmitted on the other link. Other causes of two-faced behavior are assumed to have a negligible probability. For example, it might also be possible for a two-faced behavior to occur due to the program counter of a slave's CPU being corrupted in such a way that the CPU jumps back to previously executed code after it has transmitted the original payload on only one slave link. If it jumps back to the segment of code that reads the payload from some sensor, the sensor reading might have changed, and a different payload might be obtained. Although we acknowledge that in theory such a scenario could occur, we consider it very unlikely. Moreover, software techniques similar to the ones used to prevent processes from concurrently accessing critical code regions (e.g., semaphores) could be used to further reduce the probability of such a scenario.

Our proposal to prevent two-faced behaviors consists in using additional integrity checks — that is, in addition to Ethernet's frame check sequence, which cannot detect payload corruptions that occur before the payload is handed over to one of the two Ethernet interfaces of the transmitting slave. Specifically, we propose to calculate a checksum for each payload to be transmitted. Importantly, the checksum must only be calculated once for each payload. When a slave transmits the payload, it appends the corresponding checksum. On the switch side of the links, the port guardians submit all incoming messages to an integrity check using the additional checksum. If a message does not pass the check, the corresponding port guardian drops it. Thus, frames that could lead to two-faced behaviors due to a corrupted payload do not propagate beyond the slave links of the transmitting slave.

5. Conclusions and future work

HaRTES has two key vulnerabilities. The first is the lack of robustness against permanent faults due to the HaRTES switch and master being a single point of failure. The second is the susceptibility to losing the master's periodic trigger messages due to transient faults, even though these messages are critical for the correct functioning of HaRTES. To eliminate these vulnerabilities FTTRS introduces both spatial and temporal redundancy. In this paper we proposed a series of mechanisms that allow the FTTRS slaves to manage this redundancy.

Future work includes the implementation of a software driver for the slaves to show the viability of the mechanisms introduced in this paper, and the design of mechanisms to maximize the number of slaves that can continue to operate even if all interlinks fail and therefore the masters may no longer be replica determinate.

Acknowledgements

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

References

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

[2] R. G. V. dos Santos, *Enhanced Ethernet switching technology for adaptive hard real-time applications*, PhD thesis, Universidade de Aveiro, 2010.

[3] 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*, 2001. IEEE Comput. Soc.

[4] S. Poledna, *Fault-tolerant real-time systems: the problem of replica determinism*, Kluwer Academic Publishers, 1996.

[5] A. Ballesteros, D. Gessner, J. Proenza, M. Barranco, and P. Pedreiras, "Towards preventing error propagation in a real-time Ethernet switch", in *18th IEEE Conf. on Emerging Technologies and Factory Automation*, 2013, Cagliari, Italy.

[6] G. Rodriguez-Navas and J. Proenza, "A proposal for flexible, real-time and consistent multicast in FTT/HaRTES Switched Ethernet", in *18th IEEE Conf. on Emerging Technologies & Factory Automation (ETFA)*, Sept. 2013.