Towards Preventing Error Propagation in a Real-Time Ethernet Switch

Alberto Ballesteros, David Gessner,
Manuel Barranco, Julián Proenza
DMI - Universitat de les Illes Balears, Spain
a.ballesteros@uib.es

Paulo Pedreiras
DETI/IT - Universidade de Aveiro, Portugal
pbrp@ua.pt

Abstract

Flexible Time-Triggered communication (FTT) allows a distributed embedded system (DES) to adapt to changing real-time requirements at runtime. This facilitates the continuous operation of the DES under dynamic environments that change over time. However, for continuous operation, high reliability in the nodes of the DES is also crucial. This can be achieved using node replication, as long as failure independence between replicas is ensured, which calls for preventing the propagation of errors. Our goal is to prevent the propagation of byzatine node behaviours and to ensure that local errors in the channel cannot disturb the global communication. For this, we construct the HaRTES/PG switch, a new switch based on the HaRTES implementation of FTT for Ethernet. This paper presents as a first step a study of the possible errors that may lead to Byzantine node behaviors and a global communication disturbance in HaRTES, as well as some ideas on how to prevent the propagation of these errors in HaRTES/PG.

1 Introduction

Traditionally, distributed embedded systems (DES) have been designed to operate in environments that do not change over time. This has led to static approaches that are inadequate for continuous and correct operation under dynamic environments. The alternative are flexible approaches. However, flexibility alone is not enough to guarantee continuous operation—reliability is also essential.

The goal of the project *Fault Tolerance for Flexible Time-Triggered communication* (FT4FTT) is to show that it is possible to build a highly reliable DES that can adapt its real-time operation upon changing requirements imposed by a dynamic environment. For this, the communication subsystem of FT4FTT, currently under development, uses the Flexible Time-Triggered communication paradigm (FTT) [6], which is a bandwidth efficient approach to achieve flexibility. Specifically, it replicates a switch based on the *Hard Real-Time Ethernet Switching* (HaRTES) architecture, which implements FTT for Ethernet [7]. However, for FT4FTT to achieve its goal, it must also provide high reliability for the nodes of the DES. In fact, the reliability of the nodes is particularly important since the nodes, compared with the communication subsystem, have the greater impact on the final reliability [1]. Thus, FT4FTT proposes to also use node replication. Node replication, must ensure failure independence between replicas, which is achieved by preventing *error propagation* [5] from one node to another, by which we mean preventing an error generated in a given node from creating other new errors in another node. Specifically, error propagation can be prevented by preparing non-faulty replicas to cope with the failure of another replica. The difficulty of this coping mechanism depends on how that failure can manifest, i.e., it depends on the *failure semantics* [3] of the faulty replica.

If the node replicas have non-Byzantine failure semantics, then replica failures can be handled much more easily. Enforcing such failure semantics can be achieved by local mechanisms at each node or by enhancing the communication subsystem appropriately [2]. The latter approach has the advantage that it can keep the internal design of the nodes simpler. Moreover, if switches are used such that all communication must pass through them, then their global view of the communication can be exploited. This is the case in the communication subsystem of FT4FTT.

Moreover, independently from node replication, to make FT4FTT highly reliable, it is also necessary to ensure that local errors occurring at any location in the communication channel cannot propagate in such a manner that they disturb the global communication.

Furthermore, since the switch acts as an intermediary for all communication, it is necessary to ensure that it does not fail by forwarding internal errors.

This paper presents a first step towards designing the FT4FTT fault-tolerance mechanisms. Specifically, we consider how to handle from one switch the errors generated in the nodes. For this, we design HaRTES/PG, a modified *HaRTES switch with Port Guardians* that (**G1**) ensures non-byzantine behaviours for the nodes and (**G2**) prevents the propagation of node errors that can disturb the global communication. Regarding internal switch errors, they will be dealt with in future work.

In the next section the hardware and software details of the HaRTES architecture are briefly reviewed. Section 3 presents the fault model of HaRTES. In sections 4 and 5 we study the value and timing errors that may manifest in the nodes and propose some solutions to prevent the propagation of those that are relevant for achieving the goals.

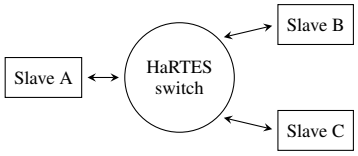

Figure 1. HaRTES architecture.

Finally, section 6 concludes the paper by summing up the contribution and pointing out future work.

2 An overview of HaRTES

HaRTES uses a microsegmented switched-Ethernet topology, in which embedded nodes, or *slaves*, are interconnected by a custom Ethernet switch called HaRTES switch (see Fig. 1). This switch incorporates a component called *master* that makes it possible for slaves to exchange real- and non-real-time traffic.

Specifically, the master organizes the communication in fixed duration slots called *Elementary Cycles* (ECs), each of which is divided into a synchronous window (for time-triggered traffic) and an asynchronous window (for event-triggered traffic). A given EC is triggered by the master through a Trigger Message (TM). This message synchronizes the slaves, conveys the EC-schedule for the time-triggered messages, and defines the window durations. Event-triggered traffic, in contrast, can be transmitted autonomously by a slave without being polled by a TM.

Real-time traffic is confined into virtual communication channels called *streams* following a publisher-subscriber messaging pattern where *publishers* attach to the streams as transmitters and *subscribers* as receivers. In HaRTES each stream has a set of real-time attributes that are enforced by the master and the switch itself. Example attributes are periods, minimum inter-arrival times, priorities, and deadlines. Note that streams are managed by the master, but all the changes are triggered by slaves through *update request messages*. In this sense, the concept of flexibility in FTT protocols refers to the ability of the slaves to ask for the modification of the stream attributes at runtime.

3 New HaRTES fault model

The HaRTES architecture (see Fig. 1) can be divided in terms of the location in which errors are generated. We have identified two types of error regions: *slave* (or external) regions and a *switch* (or internal) region. Each slave region comprises one slave and the Ethernet link that connects it to the switch. Within such a region errors may corrupt the frames, which, in turn may provoke unwanted behaviours in other network participants (the master or other slaves). As to the switch region, it encompasses all the internal components of the switch, which may fail by generating incorrect outputs. These failures may alter the correct behaviour of the switch and, thus, make it deliver an incorrect service to the slaves. As the reliability of the switch can be simply increased by using, e.g., a duplication with comparison approach, the containment of internal errors will be addressed in future work. Therefore, we will only focus on handling errors produced in the slave regions.

Frames sent by a slave may be corrupted arbitrarily, either by a slave malfunction or a fault in the channel. In this regard, the HaRTES switch includes mechanisms to ensure that synchronous messages are consistent with the schedule and that asynchronous messages respect their minimum inter-arrival time. However, other behaviours preventing a reliable communication, like the ones described in G1 and G2 (see Sec. 1), are not treated by the HaRTES switch.

To achieve goal G1, we identified node behaviours that are specifically byzantine, i.e., *two-faced* and *impersonation* behaviours. The former occurs when a slave sends different versions of the same message to different slaves; the latter occurs when the message is forged pretending to be from another network participant.

Regarding goal G2, we distinguish two types of behaviours that can disturb the communication. A *bandwidth theft* happens when a faulty slave sends traffic outside the space or time window reserved to it, in a way that it takes some bandwidth from another slave. This can cause subsequent messages to be delayed and the violation of deadlines. Moreover, if the behaviour is repeated many times, it can provoke the starvation of the other slaves, which is called a *babbling-idiot*. A *corrupt request* occurs when an update request contains errors, which can result in the incorrect deletion of a stream or the unfair modification of the resources assigned to it.

4 Analysis of errors

In this section we identify the scenarios that need to be dealt with to achieve goals G1 and G2. For this, we study how value and time deviations in a frame affect other network participants. For value errors we follow a layered approach in which we distinguish the *Ethernet layer*, the *FTT layer*, and the *Application layer*. For each layer we examine the protocol data unit (PDU) field where the data corruption appears (see Fig. 2). Afterwards we consider errors related to the time at which the PDU is received, i.e., whether it is delivered in advance or delayed with respect to the instant of time in which it is expected.

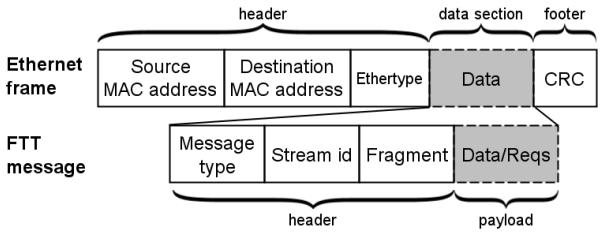

Figure 2. Diagram of PDUs.

Errors in Ethernet PDUs Ethernet frames [4] have a header, a data section, and a footer. The header contains the source and destination MAC addresses of the frame, and the ethertype. The latter identifies the type of upper-layer message encapsulated in the data section. Note that, since we currently only consider FTT messages to be exchanged over HaRTES/PG, the ethertype has for all Ethernet frames the same value, i.e., the one corresponding to FTT messages. The data section contains the payload of

the frame. Finally, the footer carries a Cyclic Redundancy Check (CRC) code for the header and the data sections.

- *Source MAC address*. This field is used by the master to identify the source of a message and, thus, any value error can have severe consequences, which depend on the final value and the moment in which the error occurs. The most important error to prevent occurs when the source address is altered in a way that it corresponds to another network participant, i.e., an impersonation.
- *Destination MAC address*. In HaRTES, messages are delivered to their destinations based on the information provided by the FTT layer, and not based on the destination MAC address. However, the value of this field is still used by receivers to discard those messages that are not addressed to them. Consequently, a corruption may cause an omission.
- *Ethertype*. An error in this field can lead a receiving node to reject the frame since it would have an un-recognized type. The error would thus become an omission.

Errors in FTT PDUs As can be seen in Fig. 2, a message transmitted by an FTT slave always has a header containing three fields: the type of the message, the stream through which it is sent, and the fragment number. This latter field is part of the fragmentation protocol of FTT, which makes it possible to transmit a big FTT message in various Ethernet frames. The rest of the message's content depends on its type. Data messages have a section containing the data that belongs to the application layer; whereas update request messages contain the group of slave requests.

- *In general*. These are errors that affect all message fields, i.e., they violate the format of an FTT message. These errors, in turn, may cause a failure in the receiving slave.
- *Message type*. An error in this field may manifest in such a way that the corrupted message type identifies the message as a master message, even if the message was transmitted by a slave. This results in an impersonation that can provoke unwanted resynchronizations or protocol information inconsistencies in the receiving slaves.
- *Stream id*. The value of this field is used by the switch to forward messages to their destinations. Thus, any corruption can provoke a message to be delivered to the wrong slave, resulting in a bandwidth theft. Moreover, in those cases in which the slave accepts the message, i.e., the destination MAC address corresponds with the slave's MAC address, an impersonation occurs.
- *Fragment*. This value is used by receivers as an index to the specific chunk of data being conveyed. Therefore, a corruption may cause the fragment to be misplaced or even omitted when the value is out of range. In both cases, this error provokes a value error in the payload, which can only be handled at the application layer, where the joined fragments become structured data.
- *Data*. Although the content of this field cannot be validated with the information provided by the FTT layer, its size can be incorrect according to the scheduling. Specially, if the message's payload exceeds the volume authorised by the master, a bandwidth theft may occur.
- *Group of requests*. In update request messages, any modification in the value of the requests is considered a corrupted request. Note that, as explained, these messages allow slaves to demand stream-management operations, which may wrongly modify or even prevent the communication of other slaves.

Errors in application PDUs The Application layer manages the data exchanged between applications. The structure of this data depends on the specific purpose of the application and, thus, this layer is out of the scope of this work. Note that, since all the timing issues are handled by the underlying time-triggered communication protocol, these errors only affect the value of the data.

Timing related errors These are errors leading to untimely receptions, i.e., errors hastening or delaying the reception of a frame:

- *Early reception*. A frame received in advance with respect to the time window in which it should be transmitted can cause a bandwidth theft error in the time domain. Moreover, a frame repeatedly delivered too early provokes a babbling idiot error, which can also provoke the omission of other messages.
- *Late reception*. A frame that is slightly delayed and, thus, is delivered outside the time window in which it should be transmitted, may result in a bandwidth theft error. Note that, in this case, the late reception of a real-time frame does not only occupy some reserved space of time, but also violates the real-time constraints defined for that frame. Additionally, a frame that is infinitely delayed results in its omission.

5 Preventing error propagation

In this section we propose some solutions to contain byzantine behaviours (goal G1) and behaviours avoiding a correct communication among slaves (goal G2). Note that, for the sake of brevity, we will focus on the errors related to these behaviours. However, the other ones, except for omissions and application-layer errors, can also be handled by means of the mechanisms described here. For this, we add a set of *port guardians* to the switch ports, which make it possible to detect and discard all those real-time frames transmitted from slaves that are incorrect, considering the error model. Finally, errors that cannot be detected are left for a future discussion, as they demand important restrictions in the protocol flexibility.

Regarding two-faced behaviours, note that the switch acts as an intermediary for all communication. Therefore, they can be eliminated by preventing their existence in the HaRTES/PG switch itself. As explained in Sec. 3, this can be achieved, for instance, by using an internal duplication and comparison mechanism within the switch.

Impersonations can be produced in different ways. Consequently, to completely avoid them, various different mechanisms have to be deployed. First, an error in the source address could be easily detected if we restrict the protocol by enforcing a static assignment of MAC addresses to the switch's ports. In this sense, guardians could discard frames whose source address does not correspond with the MAC address of the slave connected to the port. The second type of impersonations can be handled by guardians if they discard master-exclusive messages coming from the slaves. Finally, the correctness of the stream id in the received message must be assessed. For this, we propose to check its correspondence with the frame's MAC destination address. In this sense, note that, as described in Sec. 4, the destination MAC address identifies the slaves that should receive the message. Since the stream id and the destination MAC address contain redundant information we can implement a duplication with comparison scheme. More specifically, guardians can discard all those frames whose destination address does not correspond with the destination address assigned to the stream through which it is being sent. Note that this error-detection approach has a 100% coverage for one of the two values, since in HaRTES a stream is always bound to a unique multicast address.

Bandwidth theft behaviours can occur in different situations. First, wrong addressing in frames can be avoided using the stream id error-detection mechanism described in the previous paragraph. Second, messages containing more data than the one authorized by the master can be discarded by guardians as long as they gather from the master the expected size of each message. Finally, ordinary untimely messages can be discarded taking into account the synchronization information provided by the switch. However, babbling idiot behaviours must be specifically managed for each kind of message. Unexpected synchronous messages can be readily identified and removed by guardians thanks to the TM, which explicitly specifies the ones that shall be transmitted in each EC. Regarding asynchronous messages, their associated minimum inter-arrival time can be used by the guardians for this purpose.

Corrupted requests deserve a special discussion, as the switch does not have enough information to determine the correctness of a list of update requests. Therefore, these behaviours can only be prevented by avoiding their existence, which demands important restrictions in the protocol. More precisely, they involve a decrease in the flexibility of the protocol. Since flexibility is one of the main advantages of FTT, this decision has to be further studied and, thus, we leave this issue for a future work.

Finally, note that the solutions proposed are based on port guardians, which address the containment of transient errors. However, the operation of guardians can be generalized to cope with permanent failures. For this, we propose to deploy a set of error counters, which would allow to monitor the slaves' behaviour and, in presence of a permanently faulty node, the switch itself can disconnect the port and/or request external help to replace the node.

6 Conclusions and future work

In this paper we proposed some design ideas for HaRTES/PG, an enhanced HaRTES switch with advanced error-handling capabilities for replicated environments. The main goal of this switch is to prevent the propagation of byzantine failures and global communication disturbances. To characterize the sources of these behaviours we presented a systematic study in the value and time domains of the errors that can be generated by slaves, while we discussed their consequences on the system.

We proposed some solutions to contain these errors, considering the information the switch can provide. In this sense, detectable errors can be avoided at the entrance of the switch's ports by means of guardians. A special case of detectable errors are impersonations, which require additional restrictions in the FTT protocol to be fully detected. Non-detectable errors, like errors affecting update requests, are left for future work as they can only be prevented by means of protocol restrictions.

This ongoing research will assess the completeness of the results here presented. Moreover, the focus will move to the internal region, to also investigate how to handle internal switch errors. After that, the fault tolerance of replicated topologies will be considered. Finally, we plan to construct a prototype to prove the feasibility of the design.

Acknowledgements

This work was supported by project DPI2011-22992 and grant BES-2012-052040 (Spanish *Ministerio de economía y competividad*), by the Portuguese Government through FCT - Fundação para a Ciência e a Tecnologia in the scope of project Serv-CPS -PTDC/EEA-AUT/122362/2010 and by FEDER funding.

References

[1] M. Barranco, J. Proenza, and L. Almeida. Reliability improvement achievable in CAN-based systems by means of the ReCANcentrate replicated star topology. In *Factory Communication Systems (WFCS), 2010 8th IEEE Int. Workshop on*, pages 99–108. Ieee, May 2010.

[2] G. Bauer, H. Kopetz, and W. Steiner. The Central Guardian Approach to Enforce Fault Isolation in the Time-Triggered Architecture. *Proc. 6th Int. Symposium on Autonomous Decentralized Systems*, 2003.

[3] F. Cristian. Questions to ask when designing or attempting to understand a fault-tolerant distributed system. In *Proc. 3rd Brazilian Conference on Fault-Tolerant Computing*, 1989.

[4] IEEE. IEEE Std 802.3-2002 - Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications, 2002.

[5] J.-C. Laprie. *Dependability: Basic Concepts and Terminology*. Dependable computing and fault-tolerant systems. Springer-Verlag, 1992.

[6] P. Pedreiras and L. Almeida. The flexible time-triggered (FTT) paradigm: An approach to QoS management in distributed real-time systems. In *Proc. Int. Parallel and Distributed Processing Symposium*. IEEE Comput. Soc, 2001.

[7] R. Santos. *Enhanced Ethernet Switching Tecnology for Adaptibe Hard Real-Time Applications*. PhD thesis, Universidade de Aveiro, 2010.