 doi:10.1109/ETFA.2013.6648153

A proposal for Flexible, Real-Time and Consistent Multicast in FTT/HaRTES Switched Ethernet

Guillermo Rodriguez-Navas and Julián Proenza
Departament de Ciències Matemàtiques i Informàtica
Universitat de les Illes Balears, Spain
guillermo.rodriguez-navas@uib.es, julian.proenza@uib.es

Abstract

Hard Real-Time Ethernet Switching (HaRTES) is an implementation of the Flexible Time Triggered (FTT) communication paradigm over Switched Ethernet, which intends to provide hard real-time communication in a flexible manner. This paper presents a first proposal for enhancing HaRTES with a service of total order multicast for synchronous messages. This service uses the centralized online scheduling service of FTT in order to reduce complexity and bandwidth utilization.

1. Introduction

The distributed embedded systems community has apparently overcome the old controversy between event-triggered and time-triggered communication, and now seems more in favor of hybrid solutions, which try to get the best of both worlds. Among the techniques combining event-triggered and time-triggered mechanisms, the Flexible Time Triggered (FTT) paradigm stands out because of its thoughtful design and maturity [1, 9]. Initially developed for CAN, FTT has been adapted to other types of networks, including Ethernet and Switched Ethernet [7].

The properties of the FTT paradigm with respect to dependability have received less attention than the real-time properties. A few recent works have studied how to combine FTT with traditional fault tolerance mechanisms, in order to increase the reliability of the communication system [5, 2] but, in general, many aspects still have to be addressed. This paper tackles one of such aspects: the need of a communication service for guaranteeing *total order multicast/broadcast*, also known as *atomic broadcast* [3] for the synchronous messages of FTT. As reported in the literature, this type of service can provide significant benefits for dependability, typically reducing the complexity of the algorithms for managing membership, reaching consensus or, in general, handling node replication; see for instance [10].

This paper considers one specific implementation of FTT over Switched Ethernet: the Hard Real-Time Ethernet Switching (HaRTES) [4]; this implementation exhibits great potential for dependability and is already being used in the project FF4FTT as the basis for a novel replicated star topology for Switched Ethernet. The total order multicast protocol proposed in this paper defines a novel 4-phase commit protocol that relies significantly on the centralized online scheduling of FTT and its built-in fault tolerance mechanisms.

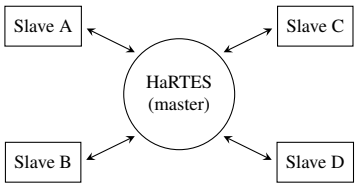

Figure 1. HaRTES architecture.

2. Background

As shown in Figure 1, HaRTES uses a simplex microsegmented star topology, with one central element, the HaRTES switch, to which each slave is connected through a full-duplex link. The main particularity of HaRTES is that this central element also contains the FTT master.

2.1 The FTT master functionality

In accordance to the FTT paradigm, the master manages all communication following a centralized master/multi-slave scheme, meaning that one single message serves for polling several slaves [1]. The master divides the communication time into rounds of constant duration called *Elementary Cycles* (ECs). The EC is partitioned into two windows: the *Syncronous window* (Sync), for transmission of synchronous messages, and the *Asyncronous window* (Async), for transmission of asynchronous messages. The master initiates each EC by broadcasting a special message called the *Trigger Message* (TM).

The TM also contains a field specifying which synchronous messages should be transmitted within the just initiated EC. This information, which we will call the *EC-schedule*, is periodically calculated by the master using the information of its *System Requirements Database*

(SRDB); a database that keeps information about all the streams of messages existing in the system.

Each stream of messages can be either synchronous or asynchronous and, analogously to CPU task scheduling, is characterized by its period, message length, deadline, offset, etc. For further flexibility, any slave can request modifications of its own data in the SRDB, i.e. modifications of the streams that this slave generates. In order to guarantee schedulability, the master executes an admission control test for any requested change and accepts the change only if the test turns out positive.

2.2 Publisher/subscriber in FTT

The last implementations of FTT over Switched Ethernet include a service for managing group communication following the well-known multicast publisher/subscriber scheme. Due to space limitations, the details of this service cannot be explained in this paper, but we summarize here the main properties it does provide.

Let N be the set of slaves of a HaRTES system, and let M be the set of message streams that these slaves exchange among them. The relationships between the slaves and the streams can be described with the help of two functions, *Publisher-of* and *Subscriber-of*, denoted as $P_i(m)$ and $S_i(m)$ respectively. These functions are defined as $P : N \times M \rightarrow \{0,1\}$, with $P_i(m) = 1$ if and only if slave i is the publisher of stream m; and $S : N \times M \rightarrow \{0,1\}$, with $S_i(m) = 1$ if and only if slave i is a subscriber of stream m.

The master keeps a data structure with the existing groups of publisher/subscribers, containing the identification of the respective streams and the associated physical addresses and ports. Specific requests issued by the slaves allow addition (binding) and removal of new slaves, either publishers or subscribers, to the groups. Note that the above definitions do not prevent, in principle, having more that one publisher per stream. But, in practice, the master limits this number to one. Thus, if $P_i(m) = 1$ then $P_j(m) = 0$ for any $j \neq i$.

In summary, every slave i knows $P_i(m)$ and $S_i(m)$ for each stream $m \in M$, whereas the master knows $P_i(m)$ and $S_i(m)$ for each slave $i \in N$ and stream $m \in M$. These properties constitute the basis of the failure detection mechanism described in Section 4.

3. Problem formulation

For the specification of our total order multicast/broadcast service, we use the notation discussed in [3]. It defines a primitive Broadcast(n) which indicates that a node has transmitted message n and a primitive Deliver(n) which indicates that a node has delivered message n for processing.

Using these primitives, total order broadcast is fulfilled if the following properties are satisfied [3]:

Validity. If a correct process broadcasts a message n, then it eventually delivers n.

Agreement. If a correct process delivers a message n, then all correct processes eventually deliver n.

Integrity. For any message n, every process delivers n at most once, and only if n was previously broadcast by the sender.

Total Order. If two correct processes p and q both deliver messages n and n', then p delivers n before n' if and only if q delivers n before n'.

3.1 Fault assumptions

Regarding channel faults, we only consider transient faults of the links/ports that may lead to omissions of messages. Note that due to the segmented nature of the star topology, these failures are typically inconsistent. Permanent faults of the links, caused for example by partition or stuck-at, can be handled with other mechanisms like spatial redundancy, and thus will not be considered.

The failure semantics of the slaves is not restricted, but we assume that the HaRTES switch provides suitable error containment functions that prevent error propagation. In particular, slaves that transmit synchronous messages out of their authorized windows are detected and isolated by the switch.

Our protocol assumes that the probability of failure of the HaRTES switch, including the embedded FTT master, is negligible. This reliability assumption is substantiated with internal duplication and comparison of the master, combined with replication of the switch. For instance as it is proposed for the HaRTES replicated star topology named FTTRS. An important feature of FTTRS upon which our protocol also relies, is the assumption that the TM is reliably and consistently transmitted in every EC. This is enforced through the use of both temporal and spatial redundancy: the TM is transmitted a number of times over each replicated star [6].

3.2 Consistency attributes

The described mechanisms of HaRTES, combined with appropriate fault tolerance mechanisms such as FTTRS, enforce a number of properties which allow us to reformulate the problem of total order broadcast in simpler terms. We call these properties *consistency attributes*.

More specifically, thanks to the reliable and consistent broadcast of the TM and the centralized publish/subscribe mechanism, the following attributes are fulfilled:

- All nodes start and finish the EC simultaneously.
- All slaves know which synchronous streams are scheduled in the current EC.
- Each slave knows whether it should receive a message to which it is subscribed, and can therefore detect an omission of said message.

- The master knows which slave is the publisher of each scheduled message. By monitoring the port associated to this slave, the master can detect whether the publisher actually sent the message or not.
- The master knows exactly which slaves are subscribers of a certain scheduled message. By forcing every subscriber to acknowledge the reception of said message, any inconsistent broadcast can be detected by the master.

FTT also exhibits a very interesting property: the EC-schedule implicitly provides an ordering of messages that is consistent throughout the network. If the slaves are forced to deliver the received messages in the same order in which they were scheduled, it can be safely said that: in an EC of FTT, *agreement* (delivering the same messages) implies *total order* (delivering them in the same order).

4. Protocol description

We propose using a novel protocol for achieving agreement and integrity, and suggest two techniques for achieving validity.

4.1 Achieving agreement and integrity

Both agreement and integrity can be achieved with a commit protocol executed in 4 phases. These phases are all executed within the transmission of two consecutive TMs, and for this reason we say that the protocol provides agreement and integrity on an EC-by-EC basis.

Figure 2 shows schematically how the different elements of the system participate in the four phases of the protocol, whereas Figure 3 shows how the different phases are mapped onto the EC structure. The latter also shows the location of two important instants that will be discussed later on: the *Accept point* and the *Delivery point*. The protocol works as follows.

In phase I, called the *Schedule phase*, the master (M) broadcasts the TM containing the EC-schedule of the just initiated EC. After reception of this message, all the slaves know which streams are active for that EC. The example of Figure 2 depicts a case in which there is only one stream scheduled, with A acting as the publisher and the other nodes being subscribers.

In phase II, called the *Broadcast phase*, every publisher of a scheduled message broadcasts its message during the Sync window. In this phase, the master is passive and the switch performs store and forward. Note that the transmission of any of the messages can be affected by channel faults, and be inconsistent. Due to this, the subscribers do not deliver the received messages yet, and wait for the decision taken in the last phase.

Phase III, which is called the *Acknowledge phase*, serves for notifying the master about the result of the broadcasts. Given a scheduled message m, each subscriber sends either a positive notification (ACK), if it received the message, or a negative notification (NAK) if it

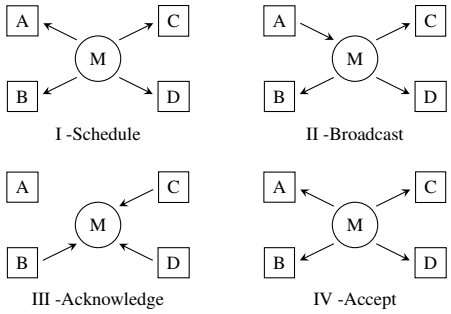

Figure 2. The 4 phases of our commit protocol, with their message flows.

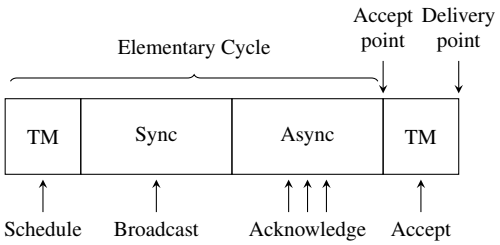

Figure 3. Distribution of the 4 phases over the EC structure.

did not receive the message. Although the protocol would work as well with the ACK messages only, we introduce the NAK messages for improving fault isolation, since reception of a NAK proves that the node is working properly and the inconsistency is more likely due to a transient channel error. The ACK and NAK messages are not transmitted immediately after reception of the messages; instead, they are transmitted within the Async window. In this phase, the master is still passive and the switch performs store and forward. The slaves that are not subscribers do not send any kind of notification to the master.

At the end of the Async window, the master has enough information to decide whether the broadcast of a message was consistent *among its subscribers* or not. This instant is called the Accept point and marks the start of the last phase, which is called the *Accept phase*. A message is accepted if and only if all of its subscribers have transmitted and ACK. Otherwise it is aborted.

At the Accept point, the master constructs a vector called *EC-Status vector* (EC-SV), which is very similar in format to the EC-schedule and indicates with one bit (a flag) which messages are accepted and can be delivered. For broadcasting this information in a reliable and timely manner, the master sends the EC-SV within the next TM. This means that every TM actually transports two different types of information: the EC-schedule of the new EC and the EC-SV of the previous (just finished) EC.

After reception of the EC-SV, the subscribers can deliver the messages for processing. This instant is known

as the Delivery point. Delaying the message delivery until the start of the next EC should not have a significant effect on the applications. In the FTT paradigm, the EC actually gives the minimum clock granularity, meaning that the ordering and delays of the messages within one EC is irrelevant for the application. This short delay can also be easily incorporated into the scheduling equations.

If we define the following primitives: *EC-Schedule(m)* indicating that message m was scheduled for the current EC; *EC-Broadcast(m)* indicating that message m was transmitted in the current EC by its publisher; and *EC-Deliver(m)* indicating that a subscriber delivered message m at the start of the next EC, then our protocol enforces the following properties for any multicast/broadcast:

EC-Agreement. If a correct process EC-delivers a message m, then all correct processes EC-deliver m.

EC-Integrity For any message m, every process EC-delivers m at most once, and only if m was previously EC-scheduled and EC-broadcast.

Total Order is implied by the Agreement property.

4.2 Achieving validity

Following are two different techniques that can be applied in order to achieve the remaining property: Validity. Our goal is to guarantee that every message transmitted by a non-faulty publisher is eventually delivered.

The first possibility provided by FTT is the automatic re-scheduling of erroneous transmissions by the master. This technique was discussed in [8] for FTT-CAN, and it can be adapted to any system in which the master is able to detect every omission, like ours. The main advantage of this approach is its simplicity, but it has the disadvantage of delaying the retransmission of a message at least one full EC: if an omission occurs in the i-th EC, the retransmission will be scheduled not sooner than the (i+2)-th EC. This happens because the Accept phase is located at the end of the EC and there is not time for recalculating the schedule before the transmission of the next TM.

Another possibility that reduces the delay of the retransmission is using a probabilistic approach for bandwidth reservation in the next EC. The underlying assumption is that the number of aborted messages in an EC will usually be low. Therefore, what the master can do is reserve some bandwidth in the next EC, just in case a retransmission is needed. The schedule of messages for this reserved bandwidth is directly the EC-SV, because it explicitly says which messages were aborted and need retransmission. In this manner, the master does not have to recalculate the EC-schedule. Whenever retransmissions are not needed, the unused bandwidth can be consistently reclaimed by the slaves as part of the Async window.

5. Conclusions and future work

This paper presented a proposal for enforcing total order multicast/broadcast of synchronous messages over an implementation of FTT Switched Ethernet called HaRTES. A significant contribution is the definition of a novel 4-phase commit protocol based on the online centralized scheduling of HaRTES and its centralized publish/subscribe service. The complexity and bandwidth requirements of the protocol remain low because it depends on having reliable and consistent transmission of the TM.

The next step of the work is to perform a complete evaluation of the protocol, including formal verification by means of model checking and reliability analysis. An interesting problem is studying the relationship between transient channel faults and false negatives, i.e. broadcasts that are consistent de facto but which the master considers inconsistent and aborts.

Acknowledgements

This work was supported by project DPI2011-22992, granted by the Spanish *Ministerio de Economía y Competitividad*, and by FEDER funding.

References

[1] L. Almeida, P. Pedreiras, and J. A. Fonseca. The FTT-CAN protocol: Why and how. *Industrial Electronics, IEEE Transactions on*, 49(6):1189–1201, 2002.

[2] M. Barranco, G. Rodriguez-Navas, D. Gessner, and J. Proenza. Towards the integration of flexible-time-triggered communication and replicated star topologies in CAN. In *IEEE 16th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–4. IEEE, 2011.

[3] X. Défago, A. Schiper, and P. Urbán. Total Order Broadcast and Multicast Algorithms: Taxonomy And Survey. *ACM Computing Surveys*, 36:2004, 2003.

[4] R. G. V. dos Santos. *Enhanced Ethernet Switching Technology for Adaptive Hard Real-Time Applications*. PhD thesis, Universidade de Aveiro, 2010.

[5] J. Ferreira, L. Almeida, J. A. Fonseca, P. Pedreiras, E. Martins, G. Rodriguez-Navas, J. Rigo, and J. Proenza. Combining operational flexibility and dependability in FTT-CAN. *Industrial Informatics, IEEE Transactions on*, 2(2):95–102, 2006.

[6] D. Gessner, J. Proenza, M. Barranco, and L. Almeida. Towards a Flexible Time-Triggered Replicated Star for Ethernet. In *IEEE 18th Int. Conf. on Emerging Technologies & Factory Automation*. IEEE, Sept. 2013.

[7] R. Marau, L. Almeida, and P. Pedreiras. Enhancing real-time communication over COTS ethernet switches. In *6th IEEE Int. Workshop on Factory Communication Systems (WFCS'06)*, June 2006.

[8] L. Marques, V. Vasconcelos, P. Pedreiras, and L. Almeida. Tolerating transient communication faults with online traffic scheduling. In *Industrial Technology (ICIT), 2012 IEEE International Conference on*, pages 396–402. IEEE, 2012.

[9] P. Pedreiras, P. Gai, L. Almeida, and G. Buttazzo. FTT-Ethernet: A Flexible Real-Time Communication Protocol That Supports Dynamic QoS Management on Ethernet-Based Systems. *Industrial Informatics, IEEE Transactions on*, 1(3):162–172, Aug. 2005.

[10] S. Poledna. *Fault-tolerant real-time systems: the problem of replica determinism*. Springer, 1995.