Schedulability Analysis of AR-TP, a Ravenscar Compliant Communication Protocol for High-Integrity Distributed Systems [*]

Santiago Urueña[1], Juan Zamorano[1], Daniel Berjón[2], José A. Pulido[2], and Juan A. de la Puente[2]

[1]Technical University of Madrid
Dept. of Comp. Architecture and Technology
E28660 Boadilla del Monte, Spain
{suruena,jzamora}@datsi.fi.upm.es

[2]Technical University of Madrid
Dept. of Telematic Systems Engineering
E28040 Madrid, Spain
{berjon,pulido,jpuente}@dit.upm.es

Abstract

A new token-passing algorithm called AR-TP for avoiding the non-determinism of some networking technologies is presented. This protocol allows the schedulability analysis of the network, enabling the use of standard Ethernet hardware for Hard Real-Time behavior while adding congestion management. It is specially designed for High-Integrity Distributed Hard Real-Time Systems, being fully compliant with the Ravenscar Profile.

1 Introduction

A Distributed Hard Real-Time System must not respond later than expected to its inputs. Therefore the timing behavior of all of its actions shall be bounded, including the access to the network. Fieldbuses like CAN Bus [14] were the communication networks traditionally used in automation systems because they fulfilled the requirements of distributed hard real-time systems, including predictability (deterministic behavior), low cost and reliability [19]. But nowadays these networks cannot provide the required bandwidth for the more demanding distributed real-time systems [17]. For example, CAN Bus cannot improve its bandwidth because it is limited by the propagation time. Therefore developers are looking for cost-effective alternatives, including network technologies that were not designed for systems with real-time requirements.

Two widely-used Local Area Network (LAN) technologies that do not have a deterministic behavior are IEEE 802.3 [9] (commonly known as *Ethernet*), and the IEEE 802.11 family of standards [10] (also known as *Wi-Fi*). These technologies are by far the most used communication networks in office environments, and through the years they have increased their bandwidth and decreased their cost, becoming *de facto* networking standards. This ensures that they will continue to be maintained and improved in the future. These advantages over other communication networks make both technologies appealing options for the development of distributed systems. Industrial versions of Ethernet and Wi-Fi have been developed to support more aggressive conditions, like those faced in automation and other industrial environments, hardening network interfaces and wires. But their non-deterministic arbitration mechanism, designed for a shared physical medium, prevents their direct use as a communication network with real-time constraints. A deterministic network protocol must be used to enable a hard real-time communication.

This paper describes the *Arbitrated Real-Time Protocol* (AR-TP), a communication protocol designed for High-Integrity Distributed Hard Real-Time Systems that employs token passing for avoiding collisions. The protocol was based on the *Real-Time Ethernet Protocol* (RT-EP) [12], a research protocol also based on transmission control techniques, but with some extensions (increased performance, congestion management, and better fault-tolerance mechanisms). AR-TP has been designed for safety-critical embedded hard-real time systems, especially in the aerospace and transportation domains. These high-integrity systems exhibit a set of strict requirements not found in other application fields, heavily derived from the entailed certification process.

The certification of an application to a high level of criticality can be hard to held (usually impossible) if no constraints are imposed to the development process. The Ada Ravenscar Profile [3] is a subset of the Ada programming

[*]Work supported by MEC, project TRECOM (TIC2002-04123), and the EC 6FP, project ASSERT (IST 4033).

language [18] developed with the certification process in mind. It restricts the Ada tasking features in such a way that defines a computational model that allows current response time analysis techniques [1, 15]. A proof of the correct temporal behavior must be provided for the certification process of a hard real-time system. Some examples of the restrictions are the use of a static number of threads and locks, that no more than one thread can be waiting at any condition variable, executing under fixed preemptive priority scheduling (FPS) [2] with the immediate ceiling priority protocol [7]. As a side effect, the obtained reduced complexity Ravenscar run-time systems have also high performance and small footprint, being thus well suited for hard real-time systems with tight deadlines or embedded systems with low hardware resources.

Although the Ravenscar profile does not place any restriction on the sequential part of the programming language, usually some restrictions are adopted in order to enable static analysis and other validation techniques that are also needed for the certification process. All in all, the programming language subset derived from the Ravenscar restrictions is enough for the development of embedded systems. However, although most communication protocols could be coded using the Ravenscar Profile, the timing analysis can be difficult to held or nearly impossible. AR-TP has been specifically designed to be easily implemented under those restrictions, as well as predictable enough for modeling its temporal behavior. The Ravenscar profile has been standardized by ISO as part of the Ada 2005 language revision, and together with deterministic protocols like AR-TP paves the way for certifying High-Integrity Distributed Hard Real-Time systems.

This paper is organized as follows. Section 2 describes the arbitration mechanism of Ethernet and Wi-Fi, explaining the sources of non-determinism and outlining the different types of methods developed to overcome this limitation. Section 3 gives a detailed description of the protocol, including its precise temporal behavior and an analysis technique of the schedulability of the network. Section 4 briefly illustrates the experiences with the implemented prototype, enumerating the future research that will be made about the Arbitrated Real-Time Protocol. Finally, section 5 presents the conclusions of this work.

2 Real-time over non-deterministic networks

The arbitration mechanism of Ethernet (CSMA/CD) monitors the physical medium until no signal is detected, sending the message when the Interframe Gap (IFG) time has expired (96-bit times, i.e. 9.6 μs at 10 Mbit/s). If a collision is detected during the transmission of the frame, the node stops transmitting the message, sends a jam signal to ensure that other stations detect the collision too, and waits for a random time before trying to send the frame again.

The basic arbitration mechanism of Wi-Fi (CSMA/CA) senses the physical medium when the node wants to transmit. If it is idle for a DIFS interval (Distributed coordination function Inter Frame Space) it broadcast the message immediately. But if the medium is busy when the node begins to sense the channel or while waiting for the DIFS interval, the transmission is deferred, selecting a random back off timer that is only decremented when the medium is idle.

This random wait in both arbitration mechanisms is the main cause of the non-predictability of Ethernet and Wi-Fi. Researchers have been looking for a solution to make Ethernet real-time capable for years, methods that sometimes could also be used with other network technologies such as Wi-Fi. Several approaches have been proposed [13]:

- **Modification of the Media Access Control**: Some researchers ([5, 16]) have proposed to modify the Media Access Control (MAC) layer of Ethernet to make the arbitration algorithm deterministic, not conforming with the IEEE 802.3 back off algorithm. In some cases this can be achieved with a special device driver, or reprogramming the firmware of the network interfaces of some manufacturers.

- **Switched Ethernet**: To avoid the performance penalty caused by collisions, the full-duplex Ethernet architecture was introduced. When two or more stations are connected through a switch collisions are completely avoided because there are two distinct communication channels for the transmission and reception of frames, thus eliminating the source of non predictability (although they introduce a processing delay that must be taking into account when analyzing the timing behavior of the system, and could introduce a small amount of extra traffic caused by the management protocols among switches).

- **Traffic shaping**: This type of proposals tries to avoid collisions controlling the amount of traffic being sent into the network [11]. When the volume of traffic is low the probability of a collision decreases (however, it is not zero), thus limiting the number of possible retransmissions. Although this approach works for soft real-time it cannot be used within a hard real-time system because it is a probabilistic method.

- **Addition of transmission control**: As well as transmission control can make reliable an unreliable transmission medium, a software layer can make Ethernet deterministic. If at every instant only one node has the right to transmit the occurrence of collisions are completely avoided, and therefore the behavior of the network is predictable.

The biggest problem with the solutions that require modifying the Media Access Control layer is that they do not follow the standard, not working with off-the-self network devices, and therefore the main advantage of Ethernet is lost. Traffic shaping in half-duplex Ethernet is a valid approach for soft real-time, but not for systems with hard real-time constraints because it is not fully deterministic.

Although full-duplex Ethernet avoids collisions, a switched Ethernet network is not directly hard real-time capable because messages can be lost if the buffers of a switch become full. However, this is a promising approach when combined with other methods like traffic shaping (transmission control for a fully switched Ethernet is overkill because there are no collisions), bringing new possibilities like the categorization of messages [8].

Finally, the use of transmission control techniques to avoid non-determinism is an effective method that can be implemented by a portable software layer above full-conforming standard hardware, and without modifying the device drivers. This method can work with any network topology, including Wi-Fi, half-duplex Ethernet, full-duplex Ethernet, or a heterogeneous network made of any combination of hubs, switches, bridges, repeaters or other network devices. However, some of them do not support the use of switches or bridges for full performance.

In summary, there are two trends for hard real-time systems: either the use of transmission control techniques, or full-duplex Ethernet combined with other methods. (It is worth noting that starting with 10 Gbps Ethernet the half-duplex mode has been dropped from the standard, so more studies about real-time for switched Ethernet will come.) The first approach will be used when it is required a shared physical medium like half-duplex Ethernet (for those applications that cannot tolerate the processing delays introduced by switches), and, of course, wireless LANs.

3 Description of the Protocol

3.1 Overview

The Arbitrated Real-Time Protocol (AR-TP) is a research real-time communication protocol for local area networks with a non-deterministic shared media like half-duplex Ethernet, Wi-Fi, or other wireless networks. It is based on RT-EP (Real-Time Ethernet Protocol), a research multipoint local area network protocol designed to achieve full predictability over half-duplex Ethernet thanks to the addition of a transmission control layer for avoiding collisions. In RT-EP a station is not allowed to put any frame on the network until it receives a special frame, called *token*, which gives it the right to transmit. Stations are organized into a logical ring where, as the network is a shared medium, every station receives all the frames.

In RT-EP the token is first circulated through all stations to write onto this special frame the priority of their messages. When the token arrives to the last node, the right to transmit is finally granted to the station with the highest priority message. When the data message has been transferred, the token is circulated again to start another negotiation cycle. The protocol is fully distributed, and is prepared to detect and solve some types of faults, like the loss of a token or a failing station. Under RT-EP the maximum blocking time for a message can be computed because the random wait introduced by the Media Access Control of Ethernet is completely eliminated. Moreover, as fixed priorities are assigned to messages, there are mature schedulability techniques for analyzing the timing behavior of the network, thus being adequate for distributed systems with hard-real time requirements. Other advantages of the protocol are its fault-tolerance mechanisms (e.g. there is no single point of failure) and especially that it works with unmodified Ethernet hardware.

However, the performance of RT-EP is lower than other protocols because a high number of arbitration packages are needed for every transferred message. This means that the network bandwidth is decreased, and that the nodes must circulate the token even when there are no messages to transmit (a delay is introduced before sending every token in order to reduce the CPU overhead). Other issues of RT-EP are that it is Ethernet specific, and that reduces the Maximum Transfer Unit (MTU) of the network due to the addition of a new header to data messages.

AR-TP was developed in order to eliminate some drawbacks of RT-EP —specially the performance and CPU overhead— while maintaining its advantages —predictability and fault-tolerance— as well as adding other features derived from the requirements. The main differences of AR-TP with respect to RT-EP are:

- **multiple messages per cycle**: more than one message could be sent in the transmission phase.
- **congestion management**: a framework for handling network overloads is added.
- **better fault-tolerance**: addition of a mechanism for message acknowledgment.

The bandwidth is increased in AR-TP because up to *n* messages could be sent in each transmission phase, as well as other improvements like decreasing the number of packets needed in the arbitration. Therefore the ratio between info data and control data is higher, the overhead of the protocol is lower and thus the throughput is increased. Moreover, less control messages are used in AR-TP than in RT-EP. However, the higher the cycle time, the worse the maximum blocking time —a vital concern of hard-real time systems—, so the optimum value of *n* is constrained by the characteristics of the system. It depends on the number of stations, the maximum size of data messages, and the worst admissible blocking time.

3.2 Detailed Description

Given a distributed system composed by M stations interconnected by a communications network, where each station can be a producer or/and a consumer, i.e. sends data messages or receives data messages (or both). All the producer nodes of the network are organized into a logical ring where each station has been assigned with a unique network identifier (ID). Each node must be informed about the configuration of the ring to know the ID of its successor and predecessor.

In AR-TP the access to the (shared) medium is controlled by the use of a token. The possession of this special message gives the station the right to transmit. It also contains information about the number of messages waiting to be transmitted, and the ID of the stations with the highest priority messages. After the initialization of the system, when the logical ring has been established, the first communication cycle starts when the node with the lowest ID creates a new token. This token is initialized by setting to zero the number of enqueued frames (messages waiting to be sent) and the sequence number. Each communication cycle has two phases: the *arbitration phase* and the *transmission phase*.

At the arbitration phase the token circulates through all the stations to determine the nodes with the n highest priority messages. While holding the arbitration token, each station checks the n priority slots of the token, where empty slots have priority 0. If one or more slots have recorded a lesser priority than any of the messages of the current station, it overwrites those slots with the priority of its messages and its ID. But if the station has no messages or the preceding nodes have filled all the slots with higher priorities, it does not modify these fields of the token. If the number of enqueued messages of this station has changed with respect to the previous cycle, it modifies the token field with the global number of enqueued messages, and then transmits the token to its successor.

When the arbitration token arrives to the last node, it will have recorded the ID of the stations with the highest priority messages. Then this last station, just after determining whether any of its messages has higher priority than the other nodes, modifies the message type to the "transmission token" value, starting the transmission phase. At this phase the token is circulated among the "winning" stations to allow them to send their messages. The token is updated with the new global number of enqueued messages. When all the messages have been sent, the transmission phase ends, starting a new communication cycle.

The last node that has transmitted a data message puts the new token in the next arbitration phase. This new token is circulated immediately unless no frames were sent at the transmission phase of the previous cycle. In that case, if the priority of zero messages were recorded in the token at the end of the arbitration phase, a delay W is introduced before starting the next cycle to reduce the transmission of control packets when there are no messages to send. It should be noted that the new initiator of the arbitration phase is probably different in every cycle.

Due to the fault-tolerance mechanisms, the behavior of the protocol requires some time-outs to detect the token loss, as well as a failing station. Refer to the paper about RT-EP [12] for more information.

3.3 Temporal Behavior

As stated above, temporal predictability is of paramount importance when certifying a hard real-time system. The *response time* of the protocol is defined as the time since between the message is passed to the AR-TP layer at the sender node, until its arrival at the destination node. All operations are bounded, and schedulability analysis can be done. The *cycle time* of AR-TP can be modeled as shown below. The k-th cycle time $Cycle_k$, of a system with M producer nodes can be obtained from the equation

$$Cycle_k = \overbrace{\left(t_{delay} + t_{token}\right) \cdot M}^{arbitration\ phase} + \begin{cases} W & n_k = 0 \\ \overbrace{\sum_{i=1}^{n_k} \left(t_{delay} + Msg_{i,k}\right)}^{transmission\ phase} & n_k > 0 \end{cases}$$

where t_{token} is the token transmission time, t_{delay} is the delay between messages, n_k is the number of messages sent in the k-th arbitration phase, $Msg_{i,k}$ is the transmission time of the i-th message of the cycle k, and W is the wait time.

The minimum delay between messages (t_{delay}) is imposed by the hardware, e.g. in Ethernet it is equal to the IFG (0.96 μs at 100 Mbit/s). This delay could also be augmented by software for other purposes, e.g. to leave enough execution time to process the token in some slow nodes (e.g. microcontrollers). Note that in RT-EP this delay between messages is used to reduce the CPU overhead of the stations (mainly when there are no messages to send), whereas in AR-TP the wait time W is used for this purpose, and this wait is only introduced after an arbitration phase with no messages. Of course $W \geq t_{delay}$.

Under AR-TP, the response time of any message is bounded. The best response time of the protocol is when the last node of the arbitration phase wants to send a minimum-size message with a higher priority than any of the other stations just before writing the final token, i.e. the minimum-size message is produced just before the start of the transmission phase, being the first to be sent, i.e.

$$R_{best} = t_{token} + t_{delay} + Msg_{min},$$

where Msg_{min}, is the transmission time of a minimum size message. However, the best case is not used to analyze the schedulability of a hard real-time system, but the worst case. In the worst case, the response time of a message μ_i is

$$R_i = Q_i + Tr,$$

where Q_i is the *queuing delay* —the longest time that a message can be enqueued before being sent— and Tr is the duration of the maximum transmission phase, i.e.

$$Tr = \left(t_{delay} + Msg_{max}\right) \cdot n.$$

The duration of the arbitration phase does not depend on the value of n, and is defined as

$$Ar = \left(t_{delay} + t_{token}\right) \cdot M,$$

i.e. $M-1$ arbitration tokens plus the transmission token sent by the last node announcing the highest priority messages.

The value of the queuing delay Q_i is composed of three delays: the *blocking time*, the *interference* and the *arbitration phase*. The response time in the worst case has to take into account the blocking time caused by the non-preemptability of the network, potentially causing a priority inversion. The maximum blocking time B of the protocol is given by

$$B = Ar + \max(Tr, W),$$

that occurs when the initial node produces a message just after the arbitration token has been transmitted, having to wait the whole arbitration phase as well as the longest transmission phase.

The interference of a message is defined as the number of cycles it must wait until being sent because there are enqueued messages of higher priority. Therefore,

$$Q_i = B + \left\lfloor \frac{\sum_{j \in hep(i)} \left\lceil \frac{Q_i}{T_j} \right\rceil}{n} \right\rfloor (Ar + Tr) + Ar$$

where $hep(i)$ is the set of messages with higher or equal priority than μ_i, and T_j is the minimum interarrival time (the period) of message μ_j. The interference depends on Q_i itself, thus a recurrence relation can be formed to obtain the final value of the queuing delay [20].

This blocking time depends heavily on the number of messages sent by cycle (n), the delay time (t_{delay}) and the size of the maximum message (Msg_{max}). The system designer can adjust any of these parameters to obtain an adequate blocking time, while maximizing the throughput of the network and minimizing the CPU overhead. The value of n can be as high as the allowed by the maximum admissible blocking time. If the application needs a very tight response time, n can be reduced to 1, giving the same blocking time as RT-EP (in fact the response time is slightly better because there are less arbitration frames in AR-TP than in RT-EP).

The priority of a message is proportional to the temporal constraints of its data, i.e. the closer the relative deadline the higher its assigned priority. This fixed priority assigned to a message is obtained by using the deduced response time analysis technique. However, although the current implementation uses Fixed Priority Scheduling, the message priorities do not need to be fixed, and they could be scheduled using dynamic techniques like Earliest Deadline First, for example. New techniques are being developed to analyze the schedulability of the network under other scheduling methods.

3.4 Congestion Management

The system is said to be overloaded when it has to process more jobs than the constrained by its resources can cope with. Overloads can happen by an unforeseen event not taken into account at development time, or during operation due to an implementation bug. Therefore, although temporal analysis is conducted during system design to obtain a correct behavior, dependable systems must be prepared to handle overloads at runtime. The protocol has been designed to cope with network overloads (congestions), i.e. when there are too many messages to be sent by the nodes without violating their time constraints. Usually, under the nominal mode of operation, the priority of a message is proportional to the temporal constraints of its data, as stated above. However, when the system is overloaded not all messages will be sent before the violation of their deadlines, therefore the priority of the message should be proportional to the criticality of the data, and not the urgency.

To achieve this behavior, in AR-TP each message has associated two priorities, one used when the system is in the nominal operating mode, and the other when the network is under an overload. A network congestion is detected when the number of enqueued messages at the end of the previous cycle exceeds a determined threshold O_1. Each station checks this value reading the last token sent in the previous cycle. In the next arbitration phase the overload priority will be considered instead of the regular one. The network congestion is considered to be finished when the number of enqueued messages decreases to a given constant O_2, where $O_1 \geq O_2$. Two thresholds are used for the mode change instead of one to avoid overload mode bouncing. The value of both constants depends on the network bandwidth, the number of nodes of the ring, as well as the maximum size of messages, being thus system specific.

The whole algorithm has been designed to have a constant computational complexity, i.e. $O(1)$ in the big O notation: in the average case, only a constant number of frames has to be examined before updating the arbitration token (the upper bound is the maximum number of messages allowed in a transmission phase), even when a mode change have just occurred. To this end, the current implementation uses two output queues in every node —one ordered by the regular priority whereas the other is ordered by the congestion priority— and thus there is no need after a mode change to reorder the messages.

During a congestion it is highly probable that some messages violate their time constraints while being enqueued, so, to avoid sending obsolete data, each message has also assigned an absolute deadline that must be checked before recording its priority in an arbitration token. Of course, the sender task of that message is informed when this happens. However, in the worst case, all the messages of a node can have violated their deadlines before being sent. Therefore, the entire queue would be examined dropping all messages while searching for the highest priorities, having thus a linear complexity. A background task is introduced for dropping all enqueued messages that have violated their deadlines, trying to avoid this situation.

4 Implementation and Future work

A preliminary version of the AR-TP protocol has been developed with the GNAT/ORK [6] cross-compilation system for PC-compatible computers. The target computers are Advantech PCM-3350 single board computers, which are PC/104 compatible and include an AMD Geode processor at 300 MHz as well as an Intel i82559 LANCE (Local Area Network Controller for Ethernet). The i82559 can operate the network at 10 or 100 Mbit/s and it was designed for the PCI bus. It must be noted that the stations are interconnected by a 100 Mbit/s Ethernet hub and not by a switch, which would introduce a noticeable transmission delay as well as extra traffic.

The intended middleware layers for high integrity distributed applications are shown in figure 1. The whole system —the low-level driver and the AR-TP protocol layer— has been implemented from scratch following the Ravenscar profile, which is supported by the Open Ravenscar Kernel. It is planned to add new low-level drivers for other network technologies, e.g. wireless local area networks. The Ravenscar profile provided a foundation adequate indeed for the development of the user application, the communication protocol, and the low-level Ethernet driver. It retains nearly every element from the Ada language needed for low-level development, such as representation clauses, which greatly improve code readability. The guide of usage of the Ravenscar profile [4] has been a great reference while implementing the protocol. The computational model of the Ravenscar profile has allowed a precise response time analysis and keeps the run-time kernel small and efficient.

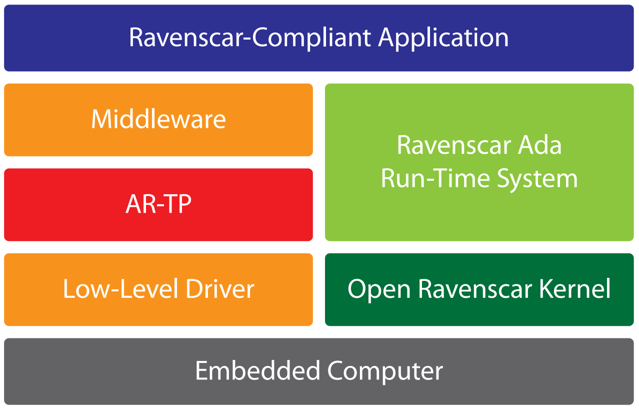

Figure 1. General architecture

The first impressions are that AR-TP greatly improves the performance of RT-EP and the behavior of the system under transient and permanent overloads. Future work is directed towards the integration of the protocol with higher layers of the communication stack, and the modification of a middleware to fully exploit every AR-TP feature (e.g. dual priorities, message deadlines). At that point, performance measures will be taken to compare the behavior of AR-TP with respect to RT-EP and maybe other predictable protocols. It is also planned to develop a tool for analyzing the network schedulability of a given hard real-time distributed system, and able to give a feasible priority assignment to each message. Finally, more studies about the dependability of AR-TP should be done, modeling also the temporal behavior of the protocol when recovering from a token loss or a failing station.

5 Conclusions

As the complexity of distributed real-time systems grows, their hardware resources must be increased. The networks traditionally used in this type of systems are becoming not capable of transmitting the required amount of information, therefore faster network technologies are being used, and even those without a real-time behavior like Ethernet or Wi-Fi. This paper has presented AR-TP, a protocol that employs transmission control techniques to avoid the non-determinism of Ethernet or Wi-Fi, making them usable for hard real-time systems.

This token-passing protocol can be implemented by a portable software layer above full-conforming standard hardware, working with any network technology and topology, including Wi-Fi, half-duplex Ethernet, full-duplex

Ethernet, or a heterogeneous network made of any combination of hubs, switches, bridges, repeaters or other network devices. It offers advanced network scheduling policies (including congestion management), static temporal analysis techniques, and its distributed architecture guarantees a fault-tolerant protocol well suited for wireless networks.

The protocol has been fully implemented in the Ada programming language, taking advantage of the Ravenscar Profile. Ada is well suited for the development of hard-real time applications and high-integrity systems, specially the Ada 2005 revision that includes several facilities not found in other programming languages. Future work is directed towards the integration of the protocol with communication middlewares, as well as developing new temporal analysis techniques. Also, it is planned to explore the behavior of AR-TP in Wireless Local Area Networks.

References

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5), September 1993.

[2] A. Burns. Preemptive priority based scheduling: An appropriate engineering approach. In S. Son, editor, *Advances in Real-Time Systems*. Prentice-Hall, 1994.

[3] A. Burns, B. Dobbing, and G. Romanski. The Ravenscar tasking profile for high integrity real-time programs. In L. Asplund, editor, *Reliable Software Technologies — Ada-Europe'98*, number 1411 in LNCS, pages 263–275. Springer-Verlag, 1998.

[4] A. Burns, B. Dobbing, and T. Vardanega. Guide for the use of the Ada Ravenscar profile in high integrity systems. *Ada Letters*, XXIV(2):1–74, 2004.

[5] R. Court. Real-time ethernet. *Computer Communications*, 15:198–201, April 1992.

[6] J. de la Puente, J. Ruiz, and J. Zamorano. An open Ravenscar real-time kernel for GNAT. In H. B. Keller and E. Plöedereder, editors, *Reliable Software Technologies — Ada-Europe 2000*, number 1845 in LNCS, pages 5–15. Springer-Verlag, 2000.

[7] J. Goodenough and L. Sha. The priority ceiling protocol: a method for minimizing the blocking of high priority Ada tasks. In *Second International Workshop on Real-Time Ada Issues*. ACM SIGAda, 1988. Ada Letters, 8(7).

[8] The Institute of Electrical and Electronics Engineers, New York, USA. *IEEE Std. 802.1D-2004*, June 2002.

[9] The Institute of Electrical and Electronics Engineers, New York, USA. *IEEE Std. 802.3-2002*, March 2002.

[10] The Institute of Electrical and Electronics Engineers, New York, USA. *IEEE Std. 802.11-2003*, June 2003.

[11] S.-K. Kweon, K. G. Shin, and G. Workman. Achieving real-time communication over ethernet with adaptive traffic smoothing. In *6th IEEE Real-Time Technology and Applications Symposium (RTAS 2000)*, Washington, USA, June 2000.

[12] J. M. Martínez and M. González Harbour. RT-EP: A fixed-priority real time communication protocol over standard ethernet. In T. Vardanega and A. Wellings, editors, *Reliable Software Technologies - Ada-Europe 2005*, volume 3555 of *LNCS*. Springer-Verlag, June 2005.

[13] P. Pedreiras, L. Almeida, and P. Gai. The FTT-Ethernet protocol: Mergin flexibility, timeliness and efficiency. In *14th Euromicro Conference on Real-Time Systems*, pages 1–10. IEEE Computer Society Press, 2002.

[14] R. Bosch Gmbh, Germany. *CAN Specification-Version 2.0 Part A*, 1991.

[15] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28:101–155, November 2004.

[16] Y. Shimokawa and Y. Shiobara. Real-time ethernet for industrial applications. In *IECON'85*, pages 829–834, 1985.

[17] Y. Song. Time constrained communication over switched ethernet. In *4th IFAC International Conference on Fieldbus Systems and their Applications*, Nancy, France, November 2001.

[18] S. T. Taft, R. A. Duff, R. L. Brukardt, and E. Plöedereder, editors. *Consolidated Ada Reference Manual. Language and Standard Libraries. International Standard ANSI/ISO/IEC-8652:1995(E) with Technical Corrigendum 1*, volume 2219 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[19] J. Thomesse. Fieldbuses and interoperability. *Control Engineering Practice*, 7(1):81–94, January 1999.

[20] K. Tindell, A. Burns, and A. Wellings. Calculating controller area network (can) message response times. *Control Engineering Practice*, 3(8):1163–1169, 1995.