Deterministic Integration of Hard and Soft Real-Time Communication over Shared-Ethernet

Paul Regnier [1] **George Lima** [1]

{**pregnier, gmlima**}**@ufba.br**

[1] Distributed Systems Laboratory (LaSiD) - Computer Science Department (DCC)
Pos-Graduation Program on Mechatronics - Federal University of Bahia
Campus de Ondina, 40170-110, Salvador-BA, Brazil

Abstract. *This paper presents a protocol proposal that makes Ethernet suitable for supporting modern real-time systems. Applications that could benefit from the proposed protocol are mainly those composed of heterogeneous distributed components, which are specified in terms of both hard and soft timing constraints. Indeed, using the proposed protocol, hard and soft real-time tasks can efficiently and predictably share an Ethernet bus. The bus utilization can be optimized by an appropriate allocation of the available bandwidth into hard and soft communication. Moreover, the protocol, compatible with standard Ethernet hardware, can implement a decentralized medium access control, increasing communication flexibility and reliability.*

1. Introduction

Research on real-time systems have been facing several challenges in the last few decades. From the application domain perspective, such systems, once seen as composed of simple, strict and short hard periodic tasks, typically used for control loops, nowadays include applications such as telecommunication, industry, military, aerospace, assets monitoring, automobile etc. While parts of these systems may contain the usual hard periodic tasks, requiring known bounded response time and controlled jitter, other parts may be made of much more complex soft (or non-hard) tasks to deal with sporadic or aperiodic events [Liu 2000], leading to a considerable task diversity. Further, from the point of view of their structure, real-time systems have been moving from centralized to distributed. This requires a suitable communication network capable of dealing with the new demands task diversity requires. Indeed, network protocols must deal with different traffic patterns and must provide not only controlled jitter and bounded message transmission time, as required by the usual hard tasks, but also high throughput, as demanded by soft and other non-periodic tasks. As a result, industrial communication networks, known by being reliable and predictable, may not suffice anymore due to their usual low bandwidth.

Ethernet has been considered a promising alternative for designing distributed real-time systems. It provides high bandwidth, it is cheap, efficient and widely used. However, one cannot consider its direct use in the real-time systems domain. This is due to its *probabilistic* bus arbitration scheme, used by the Carrier Sense Multiple Access with Collision Detection protocol (CSMA/CD) [IEEE 2001, Dolejs et al. 2004]. Several solutions to this lack of determinism, either based on hardware or software modification, have been proposed along the last three decades. A brief description on related work is given in section 2, although interested readers can refer to [Hanssen and Jansen 2003,

Decotignie 2005] for details. To the best of our knowledge, none of those former approaches succeed to offer hard real-time guarantees, high throughput and fault tolerance mechanism using a decentralized medium access control without hardware modification.

This paper describes a broadcast-oriented software-based shared-Ethernet protocol built on top of the Ethernet layer. The protocol, presented in section 3, provides deterministic communication guarantees without jeopardizing flexibility neither increasing hardware costs. Also, it preserves the Ethernet broadcast communication capacity, facility required by many real-time systems. The protocol is reliable, predictable, efficient, optimizes the bus utilization, and is capable of dealing with different communication patterns (eg periodic and aperiodic). By providing an innovative dynamic reservation mechanism, the protocol also offers an adaptable bandwidth allocation to hard real-time traffic.

2. Software-based Ethernet Approaches to Real-Time Ethernet

Perhaps the simplest deterministic software-based solution is to divide time into a sequence of slots, each of which is assigned to a specific system node. Nodes can transmit their message only in their respective time slot. This scheme, known as Time Division Multiple Access (TDMA) [Kurose and Ross 2005], gives a great deal of communication predictability. This characteristic suits well industrial applications, composed mainly of hard periodic tasks. However, TDMA lacks flexibility since even if a station has nothing to transmit, its time slot will be wasted.

Token-based protocols are usually more flexible than TDMA. They use a token to give permission to a node to transmit messages. The nodes are organized in a logical ring and the token rotates on the ring. Only the token holder node has the right to transmit on the bus. The token can be explicit or implicit. Explicit token uses a token message to allow nodes to transmit. Whenever a node terminates a transmission, it sends the token to the next node according to the schedule. RETHER [Venkatrami and Chiueh 1994] and RTnet [Hanssen et al. 2005] are examples of explicit-token Ethernet-based protocols. Although these protocols deal with different traffic patterns, they introduce extra overhead in case of token loss. Implicit-token protocols based on Timed Packet Release mechanism (TPR) offer a better alternative [Pritty et al. 1995, Carreiro et al. 2003, ControlNet 1997]. In brief, a token is implicitly passed on to the next node independently of whether or not the previous node on the ring transmits its message. The absence of message means that a node is giving up its right to use the bus. Thus, the bus can be utilized more efficiently.

Other protocols, like FTT [Pedreiras et al. 2002], use a master-slave model to control bus access. Here a special node plays the role of an arbiter (the master), polling other nodes (the slaves). Slaves are allowed to transmit their messages only if they get permission to do so. This model can handle hard and soft traffics but exhibits single point of failure, which can be dealt with by master replication. In any case, there is an asymmetric load distribution on the net, which may imply problems of scalability and performance.

Another way to avoid collisions is switch-based solutions [Kopetz et al. 2005]. However, the use of switches introduces routing, forwarding and buffering delays. Also, broadcast communication implementation is not as easy as in shared-Ethernet and require special care to manage traffic congestion in the switch buffers [Wang et al. 2002, Lo Bello and Mirabella 2001].

In this paper we mix two approaches together, TDMA and implicit token, to get

the best out of each one. TDMA is used to create two windows to predictably deal with hard and soft messages, respectively. Inspired by the Virtual Token Protocol (VTPE) [Carreiro et al. 2003], the bus access in each window is controlled by organizing the nodes in two logical rings, one for each window, and by using implicit token so that bus utilization can be optimized. The proposed protocol, called *DoRiS*, which stands for *Double Ring Service for Real-Time Ethernet*, is flexible and efficient.

3. DoRiS

DoRiS is a collision-free protocol built on top of Ethernet hardware. Located between the IP and the physical layers of the Internet stack, the protocol works as a logical layer, extending the CSMA/CD MAC layer. *DoRiS* was designed to support hybrid systems where industrials sensors, actuators and controllers share the communication network with other soft applications. It is important to notice that the processing speed and the communication characteristics of these two types of application may differ to a great extent. Indeed, most industrial appliances available on the shelf have low processing capacity compared to the Ethernet bandwidth. For example, Carreiro et al. [2003] consider typical 8051 microcontrollers that spend up to $111\,\mu s$ to process an incoming 64B message. As the transmission time on a 100Mbps Ethernet link of such message is $5.12\,\mu s$, such an appliance allows only about 4.6% of bus utilization for hard real-time communication. On the other hand, multimedia application requires higher transmission rates and faster processing nodes. Considering these observations, *DoRiS* provides both temporal guarantees for hard real-time tasks and high bandwidth access to soft and best effort ones.

After the computation model, the *DoRiS* scheme will be presented in details in the next sections. We will focus the protocol description on the functions that guarantee the shared medium access without collision. As they are based on the Ethernet standard, the functions to receive messages will not be described. Important mechanisms such as node synchronization, group membership and fault tolerance will be adressed in future work.

3.1. Computation Model

We consider a set of nodes connected through a shared-Ethernet bus. Each node is uniquely identified in the system and may run hard real-time or soft tasks. For the sake of notation, we refer to the former simply as *tasks* while to the latter as *processes*. Let N_H (N_S) be the numbers of hard tasks (soft processes). We define the two following ordered sets $R^H = \{T_1, T_2, \ldots, T_{N_H}\}$ and $R^S = \{P_1, P_2, \ldots, P_{N_S}\}$. These sets are the two logical rings of *DoRiS*, where a single token rotates.

Received messages are processed by the tasks within a maximum processing time, denoted π. This time is usually associated to message processing by industrial appliances (micro-controllers, sensors etc). We assume that these appliances, called hereafter *slow nodes*, need temporal guarantees and that their messages are short, usually periodic, and have hard real-time constraints. Such messages, called *hard messages*, are assumed to have a constant length of 64B and are transmitted through the network within a maximum transmission time $\delta \ll \pi$. The processing time for message transmission is assumed to be included in the task computation time and so is ignored during the protocol description.

Our protocol uses the publish-subscribe communication model according to which whenever a task or process has a message to send, it transmits such a message using the

Ethernet broadcast standard address (48 address bits set to 1) [Dolejs et al. 2004]. When various applications are located at the same node, they uniquely identify their messages using the type field of the Ethernet frame. Each task/process receives and processes every message. Based on the source address, they then decide whether they are interested in the message. In practical applications, tasks may not have to process all messages or may drop unused ones. However, for the sake of model simplification, we assume here that all hard real-time messages are fully processed by the tasks.

As mentioned before, if only slow nodes are present in the *DoRiS* segment, the bus may be under-utilized. However, if there are fast nodes connected to the bus, *DoRiS* allows them to use this spare bandwidth. Assuming that nodes are equipped with receiving buffers, receiving and processing hard messages are independent actions that can happen simultaneously. This also implies that more than one message can be sent in a row. Therefore, the maximum bandwidth rate available for hard real-time traffic is $B_m = \delta/\pi$. It is important to notice that B_m indicates the bound above which buffer overflows occur since the slowest node will receive more hard messages than it can process.

3.2. The Protocol Structure

The communication on the *DoRiS* bus is structured as a succession of time intervals, called *DoRiS segment* and denoted DS_k for $k \in \mathbb{N}^*$. Each DS_k is itself divided into two windows, $\mathcal{W}_k^H$ and $\mathcal{W}_k^S$, respectively associated to hard and soft real-time traffics. Tasks send messages in $\mathcal{W}^H$ and processes use $\mathcal{W}^S$ to transmit theirs. The size of $\mathcal{W}^H$ and $\mathcal{W}^S$ are denoted Δ^H and Δ^S and the *DoRiS* segment size is defined by $\Delta = \Delta^H + \Delta^S$.

We define a *DoRiS rotating sequence* RS_k associated to DS_k as the ordered list of N_H consecutive *DoRiS* segment that begins by DS_k, ie $RS_k = (DS_k, \ldots, DS_{k+N_H-1})$ as illustrated in Figure 1 focusing on DS_i.

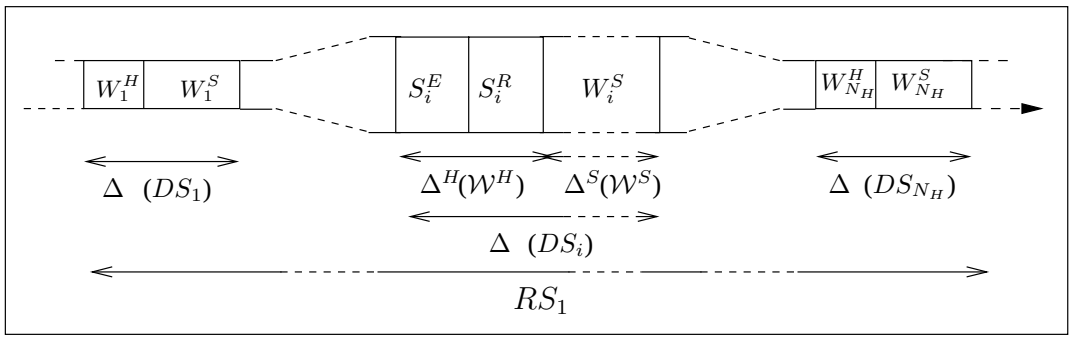

Figure 1. The $\mathbf{N_H}$ *DoRiS segments* of $\mathbf{RS_1}$ with a focus on $\mathbf{DS_i}$

We assume that tasks and processes are equipped with local timers, respectively denoted t_{T_i} and t_{P_i}, instances of the node timer. For simplicity, if T_i (or P_i) and T_j (or P_j) are two tasks (processes) located at the same node, their local timers, instances of the node timer, are equal. To divide the bandwidth into *DoRiS* segments, these timers assume values from 0 to Δ and are reset periodically at the end of each DS_k. Each task T_i and process P_i maintains also local counters, respectively denoted K_{T_i} ($1 \leqslant K_{T_i} \leqslant N_H$) and K_{P_i} ($1 \leqslant K_{P_i} \leqslant N_S$). Local timers and counters are used to organize the token circulation between R^H and R^S as will be seen in the next two sections.

To allow for some flexibility and scheduling policy, the hard real-time window $\mathcal{W}_k^H$ is further divided into two slots, the elementary ($\mathcal{S}_k^E$) and the reservation ($\mathcal{S}_k^R$) slots, as shown in Figure 1 as for DS_i. Messages sent in these two slots are hard messages, respectively called elementary and reservation messages. $\mathcal{S}^R$ is used to offer a reservation mechanism, which will be explained in section 3.3.2.

3.3. The Hard Real-Time Ring

3.3.1. The Elementary Slot

Tasks are logically organized as a ring where a unique implicit token rotates, granting medium access authorization according to temporal and logical rules.

Consider a slot $\mathcal{S}_k^E$. We define the mapping Id from $\mathbb{N}^*$ to $\{1, \ldots, N_H\}$ as $Id(k) = (k-1)\, mod(N_H) + 1$. Let $i = Id(k)$, T_i is said to hold the token in $\mathcal{S}_k^E$ if its counter K_{T_i} equals i and if its timer is equal to zero (ie when $\mathcal{S}_k^E$ just begins). For instance, considering $N_H = 4$, T_3 holds the token during $\mathcal{S}_3^E$, $\mathcal{S}_7^E$, $\mathcal{S}_{11}^E$ and so on. The following predicate formalizes these conditions:

$$ElemHolder(i) \triangleq \quad (t_{T_i} = 0) \quad \wedge \quad (K_{T_i} = i)$$

Whenever $ElemHolder(i)$ becomes true, T_i broadcasts an elementary message by executing the $sendElemMsg(i)$ operation, which consists of transmitting its local counter K_{T_i}, its reservation list (see next section), and its optional data if there are any. Sending an elementary message in each DS_k makes it easier to implement failure detectors and gives communication regularity, characteristics required by hard real-time tasks. Clearly, the logical condition $t_{T_i} = 0$ does not consider processing delays and should be expressed by means of time interval for implementation.

The protocol ensures the following invariant. In each $\mathcal{S}_k^E$, for all T_i, the counter K_{T_i} equals $Id(k)$. This is achieved by means of (periodically) incrementing the local counters by 1 at the end of each DS_k.

Observe that, as each DS_k contains one elementary message, tasks can synchronize themselves by using the End-Of-Frame interrupt of the broadcast message. Also, if an elementary message suffers an omission fault, the progression of the local counters will not be blocked, as they rely on local timers. As a consequence, the maximum drift of local timers has to be bounded according to the synchronization protocol and the number of consecutive omissions assumed to be tolerated. We are currently dealing with these aspects of the protocol and will not adress them in this paper.

3.3.2. The Reservation Slot

The reservation scheme is used to implement some *classes* of message by means of the reservation slots, which make some extra bandwidth available for the tasks. To implement this scheme each task T_i sends its reservation list upon the execution of $sendElemMsg(i)$ (cf. previous section). This list is a possibly empty subset of the ordered set $\{i+1, \ldots, N_H, 1, \ldots, i-1\}$ that indicates the reservation slots in RS_k where T_i will transmit extra data. For example, with $N_H = 4$, reservation list $\{3, 1\}$ sent by T_2 in $\mathcal{S}_6^E$ means that T_2 requests $\mathcal{S}_7^R$ and $\mathcal{S}_9^R$ in RS_6. If T_i sends an empty list, no reservation is requested.

When a task T_i receives a reservation list sent in DS_k, it updates a boolean vector of size N_H, denoted Γ_i. If the entry $\Gamma_i[j = Id(k')]$ is true, $\mathcal{S}^R_{k'}$ is reserved. Otherwise, $\mathcal{S}^R_{k'}$ is free. The value true for $\Gamma_i[j]$ is only valid for all *DoRiS* segments in RS_k. This implies that $\Gamma_i[j]$ is reset after the end of DS_{k+N_H-1}. It is important to note that *at least one reservation slot per RS_k ($k = 1, 2, \ldots$) is guaranteed per task*. Indeed, T_i ($i = Id(k)$) is the first task which can request the reservation of $\mathcal{S}^R_{k+N_H-1}$.

For a task to transmit its message in a reservation slot, it has to be sure that the reservation procedure was successful and such a reservation slot is not reserved by any other task. For example, omission faults of elementary messages could let Γ_i be inconsistent. The protocol deals with these requirements in an efficient way by establishing a reservation condition. Indeed, T_i may only request a reservation in $\mathcal{S}^E_k$ if its *state is consistent*, which is formalized by the following predicate for $k > N_H$:

$$Consistency(i,k) \triangleq T_i \textit{ received all elementary messages sent in } RS_{k-N_H}$$

For $k \leqslant N_H$, T_i is consistent if it received all elementary messages since DS_1.

Let $\Delta^E = \delta + IPG$ be the size of $\mathcal{S}^E$, where IPG is the Inter Packet Gap [IEEE 2001]. A task T_i holds the token in $\mathcal{S}^R_k$ if it has a reservation for $\mathcal{S}^R_k$ and its timer t_{T_i} equals Δ^E. Formally,

$$ReservHolder(i,k) \triangleq (t_{T_i} = \Delta^E) \wedge (\Gamma_i[K_{T_i} = Id(k)] = true)$$

Two interesting properties must be highlighted. First, the token rotation time is constant and equals $N_H\Delta$, which gives communication regularity for the tasks. Second, there are three implicit message classes: one that uses elementary slots only; the other is regarding the guaranteed reservation slot; and the third that makes use of the free (but not guaranteed) reservation slots. These message classes makes the protocol more flexible when compared to the TDMA approach.

3.4. The Soft Ring R^S

As in the hard ring, processes use their timers to control the soft ring. Elementary messages, as they are periodic and of constant size, are used as a temporal reference by all processes. In brief, the EOF of every elementary message is used as a *pulse* which define the beginning instant of the next DS_k. Like the VTPE protocol [Carreiro et al. 2003], this is actually a decentralized version of the Timed Packet Release (TPR) mechanism [Pritty et al. 1995] since no node/process plays a central role for sending the pulse.

Unlike the hard ring, processes do not have to always send messages. We introduce d_r, a temporal parameter of the protocol. The token rotates based on the carrier sense mechanism provided by Ethernet, using the following TPR mechanism. From the beginning of each $\mathcal{W}^S$, for each period d_r that it senses the medium idle, a process increments its local counter K_{P_i}. Otherwise, some other process is transmitting and the medium is busy. In this case, each process waits for the EOF interrupt associated to the current transmission and then increments its counter. The parameter d_r is chosen to be $1\,\mu s$ (considering a 100Mbps bus). This ensures unambiguous detection of the start of a packet transmitted by a previous node [Pritty et al. 1995]. Hence, a process P_i is allowed to transmit when the following spredicate holds:

$$SoftHolder(i) \triangleq (\Delta^H \leqslant t_{P_i} < \Delta^H + \Delta^S) \wedge \text{ Medium is idle } \wedge (K_{P_i} = i)$$

While the token rotates, the remaining time in $\mathcal{W}^S$ decreases. Thus, a process P_i could receive the token without enough time to transmit its message in the current $\mathcal{W}^S$. As this could happen an arbitrary number of times, a process might indefinitely be unable to transmit its message. To solve this problem, we define a "STOP" message as a special soft message (64B) that stops the token time passing in the current $\mathcal{W}^S$. Hence, a STOP message is used to distinguish a process that has nothing to send from a process that has not enough time to transmit. If P_i sends a STOP message, it will be the first to hold the token in the next $\mathcal{W}^S$. As $\mathcal{W}^S$ ends when the remaining time is equal to δ, a process that receives the token always has enough time in the current $\mathcal{W}^S$ to send a STOP message.

Considering that a process P_i may suffer a crash and then recovers, its local counter K_{P_i} may be out of date when P_i recovers. There are two cases that have to be dealt with. First, when an empty $\mathcal{W}^S$ happen, the process with the smaller identifier is assumed to hold the token first in the next $\mathcal{W}^S$. This is done by resetting K_{T_i} to 1 at all processes. Second, if P_i observes a soft message transmission, it identifies the message sender from the source address and type field of the message. Using the DS_k beginning instant and the message Start-Of-Frame instant, P_i localizes the token. As can be noted, the protocol is safe in both cases.

In the worst case, if only one process wants to transmit, it can suffer the maximum delay of $N_H\, d$. If all processes wants to transmit, the last to receive the token may wait the worst token rotation time of $(N_H - 1)\Delta^S$.

As for the bandwidth allocation, it is important to choose adequate values for the window sizes. As two hard messages may be sent in each DP, each slow node needs 2π time units to process them. Thus, as there are buffers, $\Delta^S \geqslant 2\pi - \Delta^H$ must hold. The bandwidth allocation for hard messages is maximized if one chooses $\Delta^S = 2\pi - \Delta^H$. This gives $B_h = 2\delta/\Delta$ of the bandwidth for the hard ring. As $\Delta = \Delta^H + \Delta^S$, $B_h = \delta/\pi$. Assuming $\pi = 111\mu s$, as in [Carreiro et al. 2003], $B_h = 4.6\%$. The remaining bandwidth (95,4%) is available for the soft ring.

4. Conclusion

This paper has described *DoRiS*, a protocol based on two logical rings that allows the co-existence of both hard and soft real-time traffics on the same Ethernet bus. Both types of traffic are isolated from each other in the time domain and the bus utilization can be optimized. By using the TDMA approach to implement the hard ring, *DoRiS* achieves predictability for hard real-time applications. The implicit token approach was used to control the soft ring. This configuration can allow for a better bandwidth allocation. Using typical parameters for slow nodes, it has been shown that *DoRiS* can allocate up to 95% of the bandwidth for soft traffic.

The described protocol is currently being formally specified and then will be implemented, possibly using some open source real-time operating system. Further protocol functionalities must be considered in future work. For example, appropriate priority policies can be used to take advantage of the reservation mechanism. Also, mechanisms such as membership control and dynamic reconfiguration can increase significantly the reliability and flexibility of *DoRiS*. They will be present in future steps of our research.

Acknowledgment. We would like to thank Flávio Morais de Assis Silva, Francisco Borges Carreiro and the anonymous referees for their valuable advice and com-

ments. We also acknowledge the Brazilian funding agencies CNPq (grant 304084/2003-4) and FAPESB (grant 3381/2005) for their financial support.

References

Carreiro, F. B., Fonseca, J. A., and Pedreiras, P. (2003). Virtual Token-Passing Ethernet - VTPE. In *Proc. FeT2003 5th IFAC Int. Conf. on Fieldbus Systems and their Applications*, Aveiro, Portugal.

ControlNet (1997). Controlnet Specifications - edition 1.03. ControlNet International.

Decotignie, J.-D. (2005). Ethernet-based real-time and industrial communications. *Proc. IEEE (Special issue on industrial communication systems)*, 93(6):1102–1117.

Dolejs, O., Smolik, P., and Hanzalek, Z. (2004). On the Ethernet use for real-time publish-subscribe based applications. In *Proc. IEEE Int. Workshop Factory Communication Systems*, pages 39–44.

Hanssen, F. T. Y. and Jansen, P. G. (2003). Real-time communication protocols: an overview. Technical Report TR-CTIT-03-49, University of Twente.

Hanssen, F. T. Y., Jansen, P. G., Scholten, J., and Mullender, S. J. (2005). RTnet: a distributed real-time protocol for broadcast-capable networks. In pp electronic edition, editor, *IEEE Joint Int. Conf. on Autonomic and Autonomous Systems and Int. Conf. on Networking and Services (ICAS/ICNS)*, pages 168–177.

IEEE (2001). Information Technology - Telecommunications and Information exchange between systems - Local and Metropolitan Area Networks specific requirements - part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) access and method Physical Layer Specifications. ISO/IEC 8802-3.

Kopetz, H., Ademaj, A., Grillinger, P., and Steinhammer, K. (2005). The Time-Triggered Ethernet (TTE) design. In *Eighth IEEE Int. Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*.

Kurose, J. F. and Ross, K. W. (2005). *Computer networking: a top-down approach featuring the Internet*. Addison Wesley, 3rd edition.

Liu, J. W. S. (2000). *Real-Time Systems*. Prentice-Hall.

Lo Bello, L. and Mirabella, O. (2001). Design issues for ethernet in automation. In *Proc. 8th IEEE Int. Conference on Emerging Technologies and Factory Automation*, pages 213–221, Antibes Juan-les-pins, France. EFTA 2001.

Pedreiras, P., Almeida, L., and Gai, P. (2002). The FTT-Ethernet protocol: Merging flexibility, timeliness and efficiency. In *EUROMICRO Conf. Real-Time Systems*, volume 2, pages 1631–1637.

Pritty, D., ans J. Smeed, J. M., Banerjee, D., and Lawrie, N. (1995). A realtime upgrade for Ethernet based factory networking. In *Int. Conf. Industrial Electronics (IECON)*.

Venkatrami, C. and Chiueh, T. (1994). Supporting real-time traffic on Ethernet. In *15th IEEE International Real-Time Systems Symposium (RTSS'94)*, pages 282–286.

Wang, Z., Song, Y., Chen, J., and Sun, Y. (2002). Real-time characteristics of Ethernet and its improvement. In *Proceeding of the 4th World Congress on Intelligent Control and Automation*, Shanghai (P.R. China). IEEE Computer Society Press.