ELSEVIER

Computer Communications 21 (1998) 1446–1451

computer communications

Analysis of channel utilization for controller area networks

José Luis Sevillano*, Arturo Pascual, Gabriel Jiménez, Antón Civit-Balcells

Facultad de Informática, Universidad de Sevilla, Avda. Reina Mercedes, s/n. 41012, Seville, Spain

Received 21 May 1997; received in revised form 30 April 1998; accepted 1 May 1998

Abstract

In this paper, a model for the analysis of the CAN protocol is presented, extending the analysis already developed for fixed priority multiprocessors. Two cases are considered, which correspond to the main commercial devices. The model allows us to obtain some useful results for the design of CAN-based systems. The main conclusions are those regarding bus utilization, both total and per priority level. The results are validated using simulation. Finally, the real-time behavior is also briefly discussed. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Controller area network; Performance analysis; Markov chain models; Real-time

1. Introduction

One of the most important elements in a distributed control system is the communication channel. Shared broadcast buses are usually preferred to point-to-point links, because the latter are more complex when the number of devices is high. Besides, they require a comparatively high delay to establish a connection [10]. One of the communication protocols for broadcast buses is CAN (Controller Area Network [7]), which although designed for the automotive industry, has been successfully used in many other control applications. The CAN protocol meets all the requirements of real-time communication systems: message delays are bounded [11], it includes a complete message-based priority policy, cost per node is low and several off-the-shelf devices include built-in interfaces for CAN: microcontrollers Philips 82C592 (8051 compatible), Siemens SABC167CR, Motorola 68HC08/5, Intel 80196,...; controllers Philips 82C200, Intel 82527, etc. Furthermore, the CAN architecture only defines the lowest two levels of the OSI model. The application levels are based on specific protocols that depend on the application, for instance, DEVICEnetTM for communicating PLCs and intelligent sensors, or M3S in rehabilitation engineering [4]. This reduced number of levels leads to simpler systems and lower delays, and therefore supposes a significant improvement when compared to systems like MAP.

When we are interested in the real-time behavior of a communication protocol, the main aspect is the study of 'schedulability' of the system, that is, the guarantee that messages will meet their 'deadlines' or 'crisis times'. Santos et al. [8] obtain the conditions to guarantee the schedulability of several priority disciplines, including fixed priority discipline. Tindell et al. [12] present a specific study for the CAN protocol, where they obtain the worst-case bound of message delays using an analysis developed for fixed-priority pre-emptive scheduling, which include details such as error management and 'Remote Transmission Request' messages (RTR). The importance of details such as the number of transmission message buffers and the effect of using a particular CAN controller has also been pointed out [14]. From the point of view of the protocol general behavior, fixed priority systems are well studied in the literature [16], although CAN introduces some characteristics that prevent us from using these studies in a direct way [9]. The main difference is that CAN uses message-based priorities. In order to make this point clearer, we need to briefly describe the protocol.

CAN is a multi-master (or *stations*) broadcast bus operating at a maximum data rate of 1 Mbps. Bus length is limited in order to allow synchronization, although the actual length depends not only on the propagation delay on the cable, but also on delay times of CAN controllers and transceivers, and even on oscillator tolerance [1]. Typically, at 1 Mbps the bus must be no longer than 50 m.

Every message is assigned a unique identifier (11 bits), which not only allows receivers to filter messages, but it also assigns a priority to the message. Although there are several frame types, for our study we will only consider *data frames*, which are composed of 47 overhead

* Corresponding author.

PII S0140-3664(98)00166-2

bits (Identifier, CRC, ACK, etc.) and a data field of up to 8 bytes.

When a station has a message for transmission, it waits until the channel is idle and then transmits (just like a 1-persistent CSMA) beginning with the identifier and monitoring the bus during transmission. If two or more stations try to transmit simultaneously, the wired-AND behavior of the bus makes the lower priority station lose arbitration, as it sends a recessive bit ('1') and sees a dominant bit ('0') transmitted by the higher priority station. In that case, it stops transmitting and waits for the bus to become idle again. Identifiers are required to be unique, so that the winning message is not interrupted after a collision. Therefore the channel is always busy with useful transmissions. Of course, the overhead bits limit the maximum throughput to 58% even in the best case (data field of 8 bytes). In fact, this value is usually lower because 'bit stuffing' may insert up to 19 bits. Version 2.0 of CAN protocol introduces a longer identifier field (29 bits) in 'extended format' messages in order to increase the priority levels, although in many applications 11 bits are more than enough.

Two varieties of CAN controllers exist: BasicCAN (e.g. Philips 82C200), which typically uses a single transmission buffer and the CPU manages the queuing of messages; and FullCAN (e.g. Intel 82527), which implements a more complex buffering system, based on shared memory locations, and an 'acceptance filter' so that only the useful messages are presented to the CPU. This different buffering mechanism has a strong effect on performance [14].

The rest of the paper is organized as follows. First of all, we present a simple model for CAN extending the analysis already developed for fixed priority multiprocessors [2]. We include the effect of the buffer size for transmission. Then we validate the analysis through simulation, and show some interesting results. Finally, we discuss some aspects regarding the schedulability of CAN-based systems.

2. Analysis of the CAN protocol

We consider a system composed of N stations and W identifiers per station, that is, any station generates messages with up to W different identifier values. We first assume that stations have a buffer that can store at most one message at a time. This assumption simplifies the model, and is the case of most BasicCAN controllers. Later in this section the model will be extended to handle a higher buffer size. We also assume a constant message length. The channel is assumed to be slotted in time, with slot size equal to the message transmission time, including the transmission of the identifier field (and therefore including the arbitration). The propagation delay is also included in the slot time, although it is almost negligible. Finally, we also consider an error-free channel.

Since CAN requires identifiers to be unique within the system, we may consider each of these identifiers as pseudo-stations that are assigned a fixed priority (the smaller the identifier value, the higher the priority). Let r_i denote the probability that a station generates a message in a slot with an identifier value i, $1 \leq i \leq NW$. From now on, we simply say that identifier i 'generates' a message. We assume that $r_i = r$ is a constant for all i. These probabilities are referred to the beginning of a time-slot, and they represent the situation for the rest of the slot. We also define the following probabilities:

1. $P_i = P\{$identifier i gets the station's buffer$\}$, that is, it has the highest priority of those identifiers generated within the station.
2. $R_i = P\{$identifier i succeeds in transmitting once it has got its station's buffer$\}$, that is, it has the highest priority of those (in the whole system) that got the buffer and are trying to transmit.

The computation of these probabilities will be discussed below. Anyway, with these definitions the behavior of identifier i may be modeled using a Markov chain as shown in Fig. 1.

Identifier i remains in the Idle (I) state if it either does not generate any message for transmission-$(1 - r)$- or it generates a message but does not have the highest priority within the station-$r(1 - P_i)$- or it generates a message, gets the buffer, wins arbitration and transmits-rP_iR_i-. It should be noted that the time slot includes arbitration and transmission of messages. When it generates the highest priority message within the station, but it loses arbitration-$rP_i(1 - R_i)$-, then it moves to the Active (A) state. In this state, the station's buffer is taken.

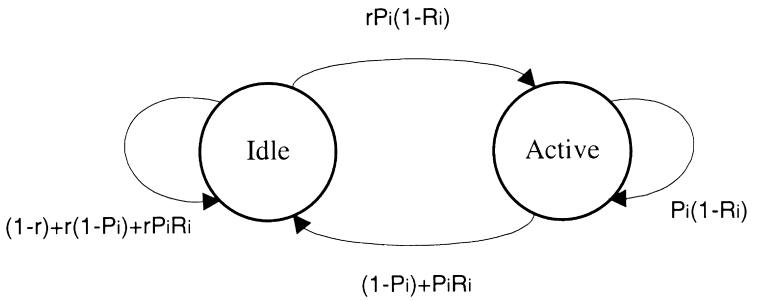

Fig. 1. State transition diagram, one message buffer per station.

The identifier leaves the Active state if any higher priority message is generated within the station-$(1 - P_i)$- in which case it must release the buffer (*pre-emption*). The message is discarded, and it may be retransmitted later, but under control of a higher software level (not the controller), so this fact is not included in our model. In case it keeps the buffer, it would transmit with probability P_iR_i. Otherwise-$(1 - R_i)P_i$- it will be retransmitted in the next slot.

Let us define $q_I(i)$ and $q_A(i)$ as the equilibrium probabilities of states Idle and Active, respectively. Then,

$$q_I(i) = \frac{1 - P_i(1 - R_i)}{1 + P_i(1 - R_i)(r - 1)}$$

$$q_A(i) = \frac{rP_i(1 - R_i)}{1 + P_i(1 - R_i)(r - 1)}.$$

At this point, we are able to compute P_i, R_i. Since identifiers are ordered by their priorities, these probabilities can be computed recursively assuming $P_1 = R_1 = 1$:

$$P_i = \prod_{n \in j,\, n < i} (1 - rP_n) q_I(n),$$

where j is the station where i is generated. That is, it is the probability that all the highest priority identifiers within the station are in the Idle state *and* that they remain in the Idle state. Of course we exclude the possibility of a transmission in that slot. Note that these probabilities depend on what we may call 'configuration', that is, the distribution of identifiers among stations. As we will see shortly, the configuration has a great effect on performance. Similarly,

$$R_i = \prod_{n=1}^{i-1} (1 - rP_n) q_I(n).$$

Using this model, the channel utilization of any given identifier can be expressed as:

$$S_i = rP_iR_iq_I(i) + P_iR_iq_A(i),$$

and the total channel utilization:

$$S = \sum_{i=1}^{NW} S_i.$$

The above model assumes that the station's buffers can store at most one message at a time, so that those messages that find the buffers full are discarded. However, the model can be easily extended to handle higher buffer sizes. This is the case of FullCAN controllers. In order to simplify the model, we suppose that there is exactly one message buffer per identifier. In other words, we assume that an identifier does not generate new messages until the current one has been transmitted, and that there is always enough space for a new message. An ideal CAN controller should have 2032 messages buffers (not extended). Of course, real controllers implement a much lower buffer size. For example, the Intel 82527 implements 15 messages per station, enough for most applications.

In this case, an identifier can be in one of three states: Idle (I), when it either does not generate any message for transmission or it simply generates a message, wins arbitration and transmits; Busy (B) when it is queued because a higher priority message is generated within the station; or Active (A) when it becomes the highest priority message within the station and it can arbitrate for the channel. The state transition diagram is shown in Fig. 2. Now, the equilibrium probabilities are:

$$q_I(i) = \frac{P_i^2 R_i}{P_i^2 R_i + rP_i(1 - P_iR_i) + r(1 - P_i)}$$

$$q_B(i) = \frac{r(1 - P_i)}{P_i^2 R_i + rP_i(1 - P_iR_i) + r(1 - P_i)}$$

$$q_A(i) = \frac{rP_i(1 - P_iR_i)}{P_i^2 R_i + rP_i(1 - P_iR_i) + r(1 - P_i)}$$

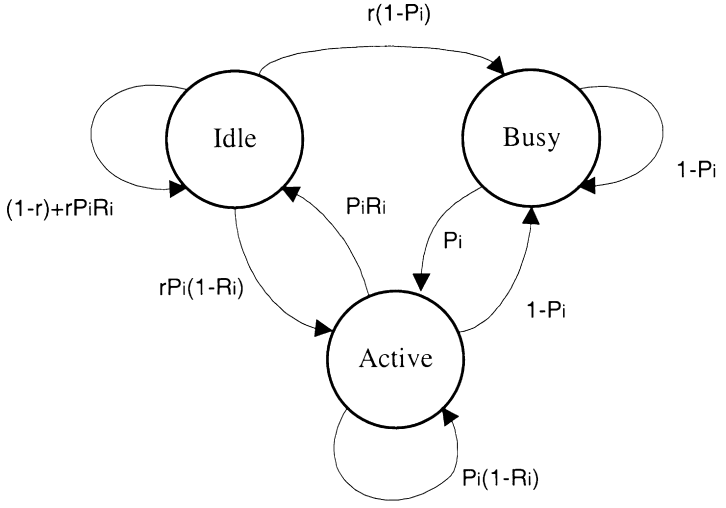

Fig. 2. State transition diagram, one message buffer per identifier.

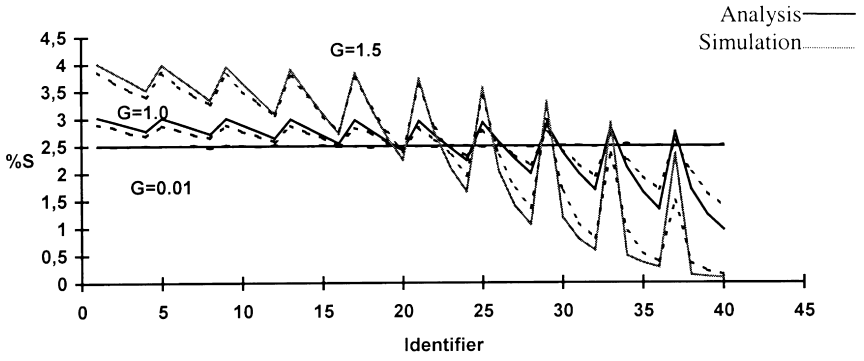

Fig. 3. Configuration A, one message buffer per station.

With regard to the probabilities P_i and R_i, we have:

$$P_i = \prod_{n \in j, n<i} (1-r)q_1(n),$$

$$R_i = \prod_{n=1}^{i-1} (1-r)q_1(n),$$

that is, using the same previous reasoning, the probability that all the highest priority identifiers stay in the Idle state excluding the possibility of a transmission. The channel utilization can be computed just as in the previous model.

3. Validation of the model

In order to validate this model, a simulation study was performed [6]. The simulation software was written using SMPL [5]. For the comparison between the models, we consider two configurations that will also allow us to show some interesting results regarding the distribution of identifiers among stations. If we assume that stations are ordered by any standard, we consider the following configurations:

1. Configuration A, where the W higher priority identifiers are assigned to the first station, the next W to the second and so on. Therefore, station j would have assigned consecutive identifiers $(j-1)W+1,(j-1)W+2,\ldots,jW$.
2. Configuration B, where identifiers are assigned alternatively among stations. Therefore, station j would have assigned non consecutive identifiers $j,N+j,\ldots,N(W-1)+j$.

Figs. 3 and 4 show the percentage of channel utilization per identifier for the case of one message buffer, obtained from both the simulation and the analytical model. We consider three values for the load $G = NWr$. We observe that the estimation using the analysis is very close to the simulation results, although a slight deviation can be observed at high loads. At low loads, the channel is evenly distributed among identifiers, but as the load increases it is expected that only higher priority identifiers obtain the bus. However, since every station is able to hold at most one message, a lower priority message may obtain a better service because an identifier has to compete first with identifiers within the station. This undesirable effect is due to the distributed nature of the arbitration mechanism, and depends on the configuration (it does not occur with configuration B). It may get worse because the time to copy a new message (e.g. in every pre-emption) to the transmit buffer is non-zero, and lower priority messages in other stations may use this time to acquire the bus. This 'priority inversion' problem [3] will be further discussed in the next section.

Figs. 5 and 6 show the results for the case of one message buffer per identifier. At low loads the results are similar to those obtained previously, so they are not shown in order to simplify the figures. Even at medium loads the analysis is not as close to simulation as with the previous model. This is probably due to the assumptions needed to keep the model simple. Anyway, the model is still useful to show

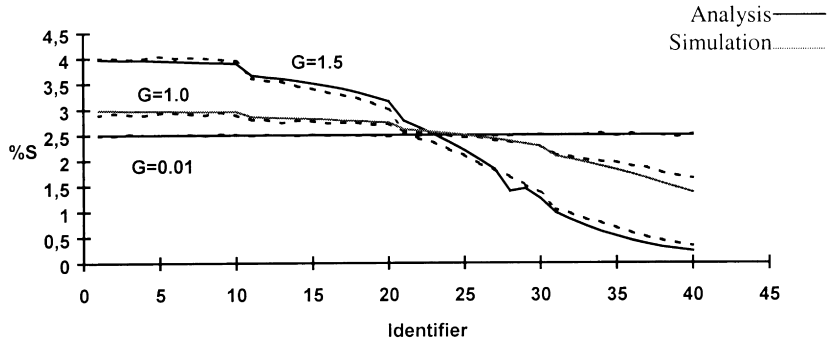

Fig. 4. Configuration B, one message buffer per station.

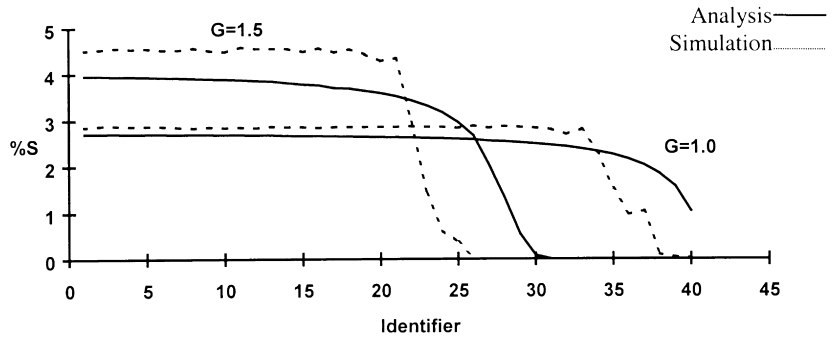

Fig. 5. Configuration A, one message buffer per identifier.

the protocol qualitative behavior; for example, how the bandwidth is distributed according to the priority level even at high loads, that is, no priority inversion is observed. These results confirm those obtained elsewhere in the literature [14].

4. Schedulability of the system

In spite of this analysis, the most important consideration when using CAN in real-time applications is the schedulability of the system, as we pointed out in the Introduction, so we should take this fact into account. We may trivially conclude that message delays are not upper bounded because the access of the lower priority messages cannot be guaranteed (except for the highest priority identifier). In general, this is the case in all simple, unfair fixed priority systems [15]. However, this problem can be solved using real-time scheduling techniques, provided that the rate at which an identifier generates messages for transmission is bounded. Note that we do not mean that messages have to be strictly periodic, although we denote 'period' as this minimum time between invocations of any given identifier.

Tindell et al. [11,12] present an analysis to bound message response times, and this worst-case analysis can be applied to assess the schedulability of the system. Their main results can be summed up as follows. The worst-case delay of a message (identifier i) is given by [11]:

$$r_i = C_{\max} + \sum_{m=1}^{i-1} \lceil \frac{r_i}{T_m} \rceil C_m.$$

where C_i is the message transmission time and T_i is the message period. The first term represents the worst-case blocking time, that is, the longest time that an identifier i can be delayed by lower priority messages, due to the fact that a higher priority message cannot interrupt a message that is already transmitting. The second term is the interference from higher priority messages that prevents identifier i from acquiring the bus.

This model assumes that the channel would not be released to lower priority messages if there are higher priority messages pending [11], that is, it does not include the priority inversion problem observed with some cases (except for messages that are already transmitting). As a result, the analysis cannot be applied to one message buffer controllers [14].

To extend the model, we have to distinguish between two cases, as in the case of 'local priority' protocols [13]. First, when an identifier i competes for the buffer within the station j, we have:

$$\sum_{m \in j, m<i} \lceil \frac{r_i}{T_m} \rceil (C_m + 2C_{\max}).$$

This term obviously depends on the configuration. Following Ref. [14], we include two additional messages due to the non-zero time to copy a new message in the transmit buffer. Each time a higher priority message pre-empts i, the bus could be claimed twice by lower priority identifiers at other stations: once when i releases the buffer and once more when the higher priority message has been transmitted and i is put back in the buffer. Note that we are considering the worst case.

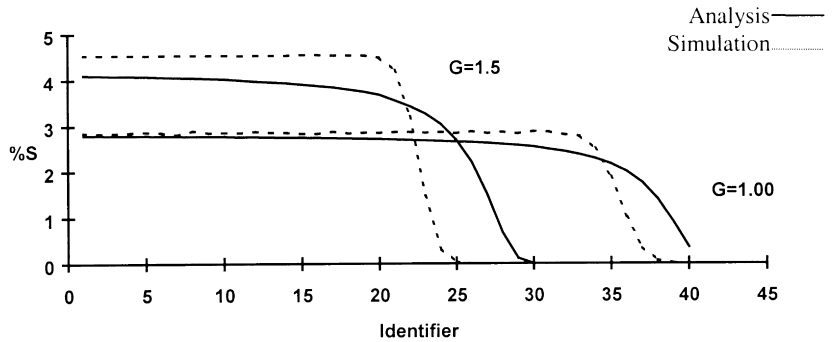

Fig. 6. Configuration B, one message buffer per identifier.

Finally, there are still two terms regarding the blocking time and the interference from higher priority messages at the other stations. Note that the blocking time should include the case when the message using the bus when i is generated belongs to the same station. In that case, when i is copied to the station's buffer the bus could be acquired by other stations, so the worst case blocking time is $2C_{\max}$. So

$$r_i = 2C_{\max} + \sum_{m \in j,\, m<i} \lceil \frac{r_i}{T_m} \rceil (C_m + 2C_{\max}) \sum_{m \notin j,\, m<i} \lceil \frac{r_i}{T_m} \rceil C_m$$

As can be seen from this equation, the worst case behavior of BasicCAN controllers is very poor. As a result, the utilization bound below which deadlines are guaranteed can be as low as 11%, depending on the application [14], not to mention the protocol overhead ($>$ 43%).

5. Conclusions

In this paper, a model for the analysis of the CAN protocol is presented, extending the analysis already developed for fixed priority multiprocessors. Two cases are considered, which correspond to the main alternatives in commercial devices: on the one hand, one message buffer as implemented in BasicCAN controllers is considered; on the other hand, we study the situation with one message buffer *per identifier* (more than one per station), as in the case of FullCAN controllers. The model allows us to obtain some useful results for designing CAN-based systems. The main conclusions are those regarding bus utilization (which is usually very low due to the protocol overhead), both total and per identifier; the effect of the controller and buffer size; the distribution of identifiers among stations, etc. The results of the analysis are validated by a SMPL-based simulation. Finally, the worst-case analysis is also discussed, following the work of Tindell et al. [12]. Our results confirm those obtained by these authors [14], in the sense that BasicCAN controllers present several drawbacks. These problems include high worst-case message delays and priority inversion in some cases. Our analysis quantify their effect on performance.

Acknowledgements

This work was supported by CICYT of the Spanish Government under project TAP93-0443.

References

[1] P. Buehring, Bit timing parameters for CAN networks, Philips Report KIE 07/91 ME, 1991.
[2] K. Hwang, F.A. Briggs, Computer Architecture and Parallel Processing, McGraw-Hill, New York, 1984.
[3] D.I. Katcher, S.S. Sathaye, J.K. Strosnider, Fixed priority scheduling with limited priority levels, IEEE Trans. Comp. 44 (1995) 1140–1144.
[4] S. Linnman, M35: the local network for electric wheelchairs and rehabilitation equipment, IEEE Trans. Rehabilitation Eng. 4 (1996).
[5] M.H. MacDougall, Simulating Computer Systems, The MIT Press, Cambridge, MA, 1987.
[6] A. Pascual, A preliminary performance study of the CAN protocol (in Spanish), Research Report, Universidad de Sevilla, 1996.
[7] Road Vehicles—Interchange of Digital Information—Controller Area Network (CAN) for High Speed Communication. ISO 11898, 1993.
[8] J. Santos et al., Priorities and protocols in hard real-time LANs; implementing a crisis-free system, Computer Communications 14 (1991) 507–514.
[9] J. Sevillano et al., A performance study of CAN in manufacturing cells, Proc. of 14th IASTED Int. Conf. Applied Informatics. Innsbruck, Austria, February 1996, pp. 12–14.
[10] K.G. Shin, C. Chou, Design and evaluation of real-time communication for FielBus-based manufacturing systems, IEEE Trans. Robot. Automat. 12 (1996) 357–367.
[11] K. Tindell, A. Burns, Guaranteed message latencies for distributed safety – Critical hard real-time control networks. Technical Report YCS229, Department of Computer Science, University of York, 1994.
[12] K. Tindell, A. Burns, A. Wellings, Calculating controller area network (CAN) message response times, Proc. IFAC Workshop on Distributed Computer Control Systems (DCCS), Toledo, Spain, September, 1994.
[13] K. Tindell, A. Burns, A. Wellings, Guaranteeing hard real time end-to-end communications deadlines, Technical Report RTRG/91/107, Department of Computer Science, University of York, 1991.
[14] K. Tindell, H. Hansson, A. Wellings, Analysing real-time communications: controller area network (CAN), Technical Report, Department of Computer Science, University of York, 1994.
[15] M.K. Vernon, S.T. Leutenegger, Fairness analysis of multiprocessor bus arbitration protocols, CSTR#744, University of Wisconsin–Madison, September, 1988.
[16] B. Wilkinson, Comments on 'Design and analysis of arbitration protocols', IEEE Trans. Computers 41 (1992) 345–351.