PAPER

Analysis and Modeling of a Priority Inversion Scheme for Starvation Free Controller Area Networks

Cheng-Min LIN[†a)], ***Member***

SUMMARY Control Area Network (CAN) development began in 1983 and continues today. The forecast for annual world production in 2008 is approximately 65–67 million vehicles with 10–15 CAN nodes per vehicle on average [1]. Although the CAN network is successful in automobile and industry control because the network provides low cost, high reliability, and priority messages, a starvation problem exists in the network because the network is designed to use a fixed priority mechanism. This paper presents a priority inversion scheme, belonging to a dynamic priority mechanism to prevent the starvation problem. The proposed scheme uses one bit to separate all messages into two categories with/without inverted priority. An analysis model is also constructed in this paper. From the model, a message with inverted priority has a higher priority to be processed than messages without inverted priority so its mean waiting time is shorter than the others. Two cases with and without inversion are implemented in our experiments using a probabilistic model checking tool based on an automatic formal verification technique. Numerical results demonstrate that low-priority messages with priority inversion have better expression in the probability in a full queue state than others without inversion. However, our scheme is very simple and efficient and can be easily implemented at the chip level.

key words: *Distributed control, priority queueing, priority inversion, controller area network, embedded control*

1. Introduction

Controller Area Network (CAN) has been one of the most successful network protocols since Robert Bosch GmbH introduced the serial bus system — CAN at the Society of Automotive Engineers (SAE) Congress in February of 1986 [2]. The CAN protocol was adopted as an international standard in 1993. Today, almost every new vehicle manufactured in Europe is equipped with a CAN network for accessing information from the electronic controller unit (ECU) deployed in an intra-vehicle network. According to the CAN specification reported by Bosch, CAN has several important properties, including prioritization messages, guarantee of latency times, configuration flexibility, etc. [3].

The priority messages property makes CAN one of the most dominant bus protocols worldwide. To achieve message prioritization, the CAN protocol employs a special circuit design called bit dominance to design its bus drivers. The bus driver circuit uses an open collector logic technique to solve the conflict problem. The bus is used to connect one or more CAN devices to provide two levels of signals: recessive and dominant. Any device may start to transmit a message if the bus is free. However, it is difficult to avoid the conflict situation. Fortunately, an identifier and an open circuit logic technique are used to solve the conflict situation in the CAN. The identifier is located at the arbitration field in sending messages. When at least two CAN devices start transmitting messages with different identifiers at the same time, the bus access conflict is resolved by the identifier because a dominant bit wins over a recessive bit.

Due to its advantages in bus access conflict resolution, CAN has been widely adopted in automobile and industry control, such as vehicles [4], [5], smart sensors [6], distributed fuel system [7], spectrometer slow control system [8], real-time communication systems [9], [10], fault confinement [11], time triggered communication [12], and remote sensing [13]. In recent years, there has been renewed interest in message prioritization following automobiles with CAN to be presented to the public. In 2003, Hasnaoui et al. [14] presented a dynamic priority policy based on reserve bits for reducing blocking probability. In 2007, Anwar and Khan [15] used the message ID window technique to provide six priorities for delivering message prioritization. The advantage of their scheme is to provide the flexibility to change the CAN message priorities on the CAN network without changing the meaning of the CAN message content. However, an extra message ID window must be attached. In 2008, Murtaza and Khan [16] proposed another solution by adding a master node deployed in a CAN bus to prevent the starvation problem occurring in other low-priority nodes. The master must try to detect a starving node and ensure that the starving node keeps participating in communication. Although the scheme can prevent the starvation problem, an extra node must be supported hence the scheme needs a higher burden and the scheme causes a master node to become a bottleneck.

As discussed above, CAN naturally has an advantage in message prioritization but the advantage causes a starvation problem. Although the proposed two schemes, message ID window and master node, can prevent the problem, they use either extra information or extra nodes to avoid generating a starving node. Hence, a simpler scheme is designed in this paper to prevent the starvation problem in CAN-based systems. Our scheme, called a priority inversion scheme, uses only an extra bit and a software counter, hence only a few resources are required and the process is easy to implement on a chip.

The rest of this paper is organized as follows. Section 2

Manuscript received September 11, 2009.
Manuscript revised December 18, 2009.

[†]The author is with the Department of Computer and Communication Engineering & Graduate Institute of Electrical Engineering and Computer Science, Nan Kai University of Technology, Tsao Tun, 542, Nan Tou County, Taiwan, R.O.C.
a) E-mail: lcm@nkut.edu.tw

DOI: 10.1587/transinf.E93.D.1504

presents a system model. The CAN architecture will be first introduced and then message prioritization will be presented. A novel design idea will be proposed and then an algorithm is illustrated the idea to be implemented in Sect. 3. Section 4 uses a probability checker tool to develop our experiments. Concluding remarks are presented in Sect. 5.

2. System Model

In this section, we first establish the system model of CAN and then introduce priority queueing system.

2.1 Controller Area Network

CAN is a multi-master message broadcast system consisting of a two wired copper bus called CAN bus and N CAN controllers denoted by C_i, where $1 \leq i \leq N$. In an automobile, these controllers are called ECUs. A controller can be layered into four layers; application, object, transfer, and physical. Application layer works on top of the protocol stack to provide various services for users. The object layer takes into account message filtering as well as message and status handling. In the transfer layer, there are several important functions, including fault confinement, error detection, message validation, acknowledgement, arbitration, message framing, transfer rate and timing. In most bottom layer, it works well in transmission medium, signal level, and bit representation.

Four different frame types are provided in the CAN bus, including data, remote, error, and overload frames. A data frame carries data from a controller to another. A remote frame with an identifier is used to request another controller transmitting the data frame with the same identifier. An error frame is sent to indicate an error occurring in the bus. When an extra delay occurs between the preceding data frame and the succeeding data or remote frames, an overload frame will be sent. Each controller can transmit message on CAN bus without taking permission from any other controller on the bus.

Designing a smart bus access mechanism is very important, especially in real-time control networks. CAN belongs to multi-master networks. Carrier Sense Multiple Access with Collision Detection and Arbitration on Message Priority (CSMA/CD+AMP) is adopted in the CAN protocol [17]. Hence, CAN's bus access is nondestructive and bitwise arbitration. Nondestructive means that a high-priority message wins the conflict and the message is not resent when two messages are sent at the same time. To complete this objective, an open collector circuit is designed in the physical layer. The logical levels 0 and 1 on the bus are sent as dominant and recessive bits, respectively. Bitwise arbitration means that the dominant bit has a higher priority than the recessive bit.

In the CAN protocol physical layer, a wired "OR" design called the open collector logic is adopted. Each transceiver in a CAN node has two ports; receiver and transmitter. In the CAN protocol, the arbitration adopts

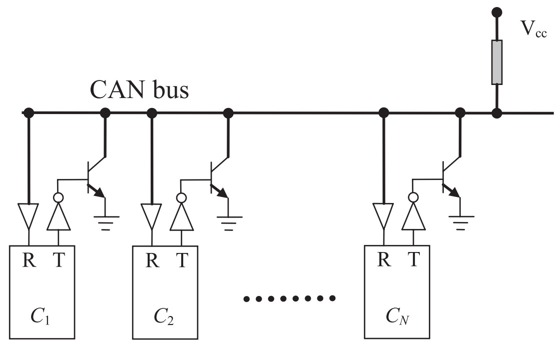

Fig. 1 The low signal is dominant designed by CAN bus.

an identifier mechanism located in the transmission frame. When a CAN node sends a message to another node through its transmitter port, an identifier is obtained from its receiver port simultaneously. Then a comparison action between the sending identifier and the reading identifier is performed. If both identifiers are different, the node will stop to send its frame. Hence, a high-priority message wins the bus access and transmitting the message again is not required.

Figure 1 illustrates the open collector logic circuit used in the CAN bus. From the figure, we know that a low signal is dominant because the bus adopts the wired OR design. Hence, when two control nodes send a high signal and a low signal at the same time. After sending a bit from a transmitter port denoted by "T" in the figure, a signal of the bit is received simultaneously by a receiver port denoted as "R".

2.2 CTMC Model for Priority Messages

Most communication systems use Markov chains to analyze performance and reliability, especially CTMCs. According to the definition presented by Kwiatkowska et al. [18], a labelled CTMC is a tuple, $\mathbf{C} = (S, \bar{s}, \mathbf{R}, L)$, consisting of four components as follows.

(1) S is a finite set of states;
(2) $\bar{s} \in S$ is the initial state;
(3) $\mathbf{R}: S \times S \rightarrow \mathbb{R}_{\geq 0}$ is the transition rate matrix;
(4) $L: S \rightarrow 2^{AP}$ is a labelling function which assigns to each state $s \in S$ the set $L(s)$ of atomic propositions (AP) that are valid in the state.

For CTMCs, we should change the transition rate matrix $\mathbf{R}$ into the infinitesimal generator matrix $\mathbf{Q}$. Hence, we calculate the matrix $\mathbf{R}$ to obtain the matrix $\mathbf{Q}$. The calculation equation is given by

$$\mathbf{Q}(s, s') = \begin{cases} \mathbf{R}(s, s') & \text{if } s \neq s' \\ -\sum_{s'' \neq s} \mathbf{R}(s, s'') & \text{otherwise.} \end{cases} \tag{1}$$

3. Priority Inversion Scheme

In this section, we will first introduce our priority scheme. We use an inversion bit to design our scheme. Hence, the overhead for our scheme is very small. An algorithm combining a counter with the inversion bit will be proposed. Finally, we prove that our scheme is starvation free.

3.1 Design

The CAN take the lead in the technology of series communication for industry control because it can easily solve conflict when at least two controllers send messages at the same time. The CAN utilizes a simple circuit that is an open collector. To meet the demand for priority solving in hardware circuits, we maintain the identifier design idea as a priority value. The CAN has two identifier formats, 11 bits and 29 bits. In most applications, 2048 and 536870912 kinds of messages are sufficient. Hence, several ways utilizing the message identifier value are designed to temporarily modify the priority to prevent starvation. For example, subtracting a certain amount of the value and using a pre-defined value such as "0". The subtracting scheme is very simple but a mortal defect, the negative priority, will be generated. Although the pre-defined value is used to solve the negative priority problem, starvation will occur for low priority message because low priority and high priority messages are inversed. We utilized one bit to design priority inversion analogous to the pre-defined value scheme. This bit is called an inversion bit with the most significant bit. When the inversion bit is set to one, all messages maintain their original priority. However, when a message must obtain a higher priority, the inversion bit is reset. Because a lower identifier has a higher priority, we put all messages in original priority from 2^{10} (2^{28}) to $2^{11}-1$ ($2^{29}-1$) using 11 (29) bits.

To guarantee starvation free conditions, we designed the lowest priority message as changing to the highest priority message when the priority inversion bit is set to one. The one's complement scheme is adapted to complete this requirement. For example, a message with an identifier of 11111111111 represents its priority as the lowest but the identifier is inverted to 00000000000 representing the highest priority. To seamlessly connect with the architecture of the current CAN, we implement our priority inversion scheme in the object layer of the CAN architecture, as shown in Fig. 2. Therefore, our priority inversion scheme can work well with the arbitration module in the transfer layer together to finish the transparent purpose. There are two main tasks in the priority inversion scheme. The first is to perform the one's complement. The other is a counting mechanism to calculate the number of transmission failures for inversion priority. The algorithm details are presented in Sect. 3.2.

From the above discussed, we generalize our scheme with the following properties:

(1) Hardware independency: our scheme does not require any hardware resource.
(2) Simplification: only one bit is required in our scheme and its calculation is very easy.
(3) Transparency: our scheme is compatible with CAN without need to redesign.
(4) Autonomy: each controller according to our scheme works autonomously.
(5) Distributed control: each controller sends its message without cooperation to the others.

3.2 Algorithm

In the previous subsection, we discussed the priority inversion mechanism design. We use a priority inversion bit that is located in the left most bit of the arbitration field in CAN communication frames. Therefore, our scheme does not need to expand any hardware resources. Our scheme uses autonomous and distributed control. Each CAN's node can execute independently. No cooperation mechanism is required. The priority inversion scheme only works on the object layer as shown in Fig. 2. The scheme is independent of the application and transfer layers. Hence, the scheme is transparent to the current CAN protocol. Another characteristic of this scheme is that only one bit required; hence, its overhead is very low. Except for one reserve bit, 1's complement scheme is used to inverse priority.

The details of our algorithm are depicted in Fig. 3. The algorithm can be divided into four parts described as follows. (1) From the data structure, each CAN node must maintain several variables, including a counter for transmitting failure, two variables storing the message identifier and the temporal identifier, a flag storing priority inversion whether it occurs or not, denoted by *cnt*, id_t, id_m, and *inv*, respectively. In addition, there is a constant value for *maxcnt* to specify the maximum number of failures as a change to invert its priority. If the *maxcnt* value is set to true, the

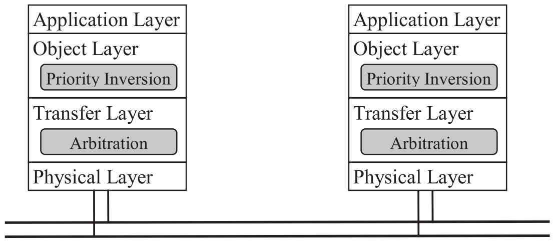

Fig. 2 Architecture layer of CAN with priority inversion.

```
(1)  Data Structure at $C_i$
     $cnt, id_t, id_m$: Integer (:=0)
     $inv$: Boolean (:=false)
     $maxcnt$ : Integer (:=MAXFAILURES)
(2)  When $C_i$ receives a message from $C_j$
     $id_t$ := GetReceiveID()
     If MSB($id_t$) is low Then $id_t$ := not $id_t$
     $id_m$ := $id_t$ & 01111111111b
(3)  When $C_i$ sends a message to $C_j$
      $cnt$ := 0
      $inv$:= false
     do
     {
            $id_m$ := GetSendID()
            $id_t$ := $id_m$ | 10000000000b
            If $inv$ is true Then $id_t$ := not $id_t$
            Send the message with $id_t$ to $C_j$
            If $cnt < maxcnt$ Then $cnt$:=$cnt$+1
            Else $inv$:= true
     }while (sending fail);
```

Fig. 3 Priority inversion algorithm.

value will not changed until the message is sent successfully. (2) When a CAN node C_i receives a message with identifier from another C_j, C_i performs the following steps. First of all, the function GetReceiveID() is called to retrieve the identifier located in the received message. If the most significant bit stored in the identifier of the received message is at a low level, the identifier must process the priority inversion action. In other words, the action of the one's complement is performed. The most significant bit will be filtered to obtain the original identifier from the object layer in C_i. (3) When a CAN node C_i wants to send a message with identifier to C_j, C_i performs the following steps. First of all, the priority inversion flag, *inv* is clear and the failure counter, *cnt* is reset. Then, the program will entry a loop until sending task is successful. The GetSendID function is called to retrieve an identifier from the message that will be sent to C_j. The identifier is set with its most significant bit to one. The flag *inv* will then be checked. If the flag is true, the priority inversion action will be performed. The message with the processed identifier is then sent to the CAN node C_j. The failure counter will be increased and the priority inversion flag is set when the failure counter value arrive at the specified maximum value.

3.3 Proof of Correctness

In this subsection, we demonstrate that our priority inversion algorithm is a starvation free scheme. **Lemma 3.1** In the CAN node physical layer, priority(msg_1) > priority(msg_2) if and only if value(msg_1) < value(msg_2).

Proof: "if": We assume that msg_1 and msg_2 have different content consisting of n bits. When msg_1 and msg_2 want to access the CAN bus at the same time, they will be sent sequentially by bit sequence. According to Fig. 1, a bit with a low signal dominates another bit with a high signal. Hence, a smaller value evaluated from msg_1 has a higher priority than msg_2.

"only if": Msg_1 has a higher priority than msg_2; hence msg_1 will win the bus control right when msg_1 and msg_2 is sent simultaneously. Hence, the value of msg_1 is smaller than that of msg_2. As discussed above, we proved the priority message mechanism. □

According to discussion above, we found that a CAN bus has naturally the characteristic of priority message due to its circuit design in its physical layer. Although starving situations do not occur as a lower traffic in the bus, a starving situation may occur in a higher traffic load. The starvation problem in priority messages is the same as that in the priority scheduling algorithm. We know that a starvation is indefinite blocking. A famous event in the starvation problem occurred at MIT in 1973 [19]. When a system is closed, a low-priority process that had been submitted in 1967 had not yet been run. From this example, we found that a starving situation will occur with a higher traffic load in CAN bus because the low-priority message will be blocked due to higher priority messages sent at the same time.

Definition 3.2 A message cannot be sent successful within a specified time interval, δ, we call this phenomenon starvation. Formally, the starvation phenomenon is described by the property $P_{\text{starving}} = \diamond^{\leq\delta}$ (send $\wedge$ O $\neg$ success).

In the above definition represented by Linear Temporal Logic (LTL), send and success is two atom propositions to be used as a send action and a successful flag, respectively. In addition, the symbols, $\diamond$, $\wedge$, O, and $\neg$ are to denote eventually, conjunction, next, and negative, respectively.

Lemma 3.3 The fixed priority system possesses the starvation characteristic if a low priority message is continuously sent but lost in the bus control right forever due to high priority messages in other ECUs.

Proof:

For a fixed priority system, a message priority is fixed. Hence, a low priority message is unsuccessfully sent, it loses the bus control right if a higher message is present.

According to the definition in [20], the mean waiting time $\overline{W_r}$ for an arriving message i with priority class r has three components:

1. The mean remaining service time $\overline{W_0}$ of the job in service.
2. $\overline{W_{qr}^{\geq}}$, the mean service time of messages in the queue that are served before the message of class r.
3. $\overline{W_{ar}^{>}}$, the mean service time of messages that arrive at the queue while the message of class r is in the queue and are served before it.

The mean waiting time for class r messages can be written as the sum of three components:

$$\overline{W_r} = \overline{W_0} + \overline{W_{qr}^{\geq}} + \overline{W_{ar}^{>}}. \tag{2}$$

The mean service time, $\overline{W_{ar}^{>}}$ approaches infinity, if the message with priority is larger than r and continuously generated before a message of class r is served. According to Definition 3.2, a starvation situation will occur if a low priority message must wait during δ with higher priority messages in other ECUs. Hence, P_{starving} is evaluated to be true. However, a fixed priority system has a starving phenomenon. □

Theorem 3.4 The priority inversion algorithm is a starvation freedom scheme.

Proof:

According to Lemma 3.3, using a fixed priority mechanism causes the starvation problem. Fortunately, the priority inversion algorithm does not belong to fixed priority schemes because changing priority will occur in the arrival failure counter at a specified value. Although a dynamic priority scheme is one method for solving the starvation problem, this scheme does not guarantee preventing the starvation problem because the highest-priority message is not blocked by other messages. In our priority inversion scheme, the lowest-priority message will be changed into the highest-priority message. For the message with priority inversion, its mean waiting time, $\overline{W_r}^*$ is very similar with Eq. (2). Only three components are required as follows.

$$\overline{W_r}^* = \overline{W_0} + \overline{W_{qr}^{\geq}}^* + \overline{W_{ar}^{>}}^*, \tag{3}$$

where $\overline{W_{qr}^{\geq}}^*$ and $\overline{W_{ar}^{>}}^*$ represent the inversion part of the mean service time for in-queue messages and the mean service time for arrival messages, respectively. It is interesting that only one component in Eq. (3) is left if only one message has the inversion phenomenon because $\overline{W_{qr}^{\geq}}^*$ and $\overline{W_{ar}^{>}}^*$ become zero. Hence, a message with priority inversion will be processed quickly because the mean waiting time is very small. Hence, the maximum waiting time never exceeds the time threshold defined in Definition 3.2. Because changing priority is temporal; no message can be permanently blocked. As discussed above, we proved that the priority inversion algorithm is a starvation free scheme. □

4. Numerical Results

In this section we consider an example of priority messages with two classes of priority. We use PRISM [21] which is a probabilistic model checking tool to evaluate the performance of the priority message example because that tool has already been apply in randomized distributed algorithms [22], communication protocols [23] and security [24]. The tool of PRISM is a probabilistic model checker using an automatic formal verification technique for the analysis of systems which exhibit stochastic behavior. In order to observe an action model, we use the tool to build a continuous-time Markov chain model (CTMC) to design two examples. One is a fixed priority queueing system without priority inversion mentioned in Sect. 2.2 and the other is a dynamic priority queueing system with priority inversion mentioned in Sect. 3.1. Table 1 shows the default value of the parameters for the two priority message examples. Notice that the M parameter only used in the priority inversion example. Its detailed description is presented in Sect. 3.2 (see *maxcnt*). In a standard CAN network [17], the maximum delay time for a message with the highest priority depends on the message with the maximum length and transmission rate is calculated 130 bit times. We assume that simulated time unit is to transmit one bit time. Hence, the service rate is defined as 1/130.

We use the PRISM language to design three and four modules for two schemes without and with priority inversion, respectively. The three modules are same and their names are highqueue, lowqueue, and canbus for two schemes. In highqueue and lowqueue modules, each has a queue with size N. The arrival rate of λ_h and λ_l is used to generate a message stored in the high priority queue and the low priority queue respectively for the two modules. The canbus module with service rate μ provides a priority bus access mechanism. The fourth module is called priorityinversion. The module only used in the priority inversion scheme use the counter M to invert priority for low priority messages.

To observe the probability, we identify a property to be expressed in a language based on the temporal logic Continuous Stochastic Logic (CSL) developed originally by Aziz et al. [25]. Next, we observe system behavior in steady state for two schemes. A property is designed to observe the probability when the queue is full. The property is listed as follows.

$$S =? [(n = N)]. \tag{4}$$

Equation (3) is applied in Sect. 4.1 and Sect. 4.2.

4.1 Without Priority Inversion

In this subsection, the current priority messages with two priorities can be evaluated. The default queue size value per controller is 20. We use PRISM language to develop a traditional CAN system consisting three modules, including a high-priority node, a low-priority node, and a bus. The two nodes have a queue with N size. The queue will be increased by one as a message is arrival. For the bus, there are two states; free and busy. When the bus is free, a message generated by the high-priority node will first execute and then one generated by the low-priority node. The actions of synchronization can be used to force two or more modules to make transitions simultaneously. For example, when a message generated by a node module is served using the mean transmitting, the bus module should be switched from free into busy.

We observe the probability in $N = 20$ (in full queue state) as shown in Fig. 4 when the high priority message arrival rate is from 0.0002 to 0.005. From Fig. 4, we know that the full queue probability for low priority messages increases when the high priority message arrival rate increases. Notice that the full queue probability for low pri-

Table 1 The default of parameters of priority messages example.

Parameters	Description	Value
N	Queue size in every CAN node	20
n	The number of message in the queue	0
M	Counter for priority inversion	10
λ_h	Arrival rate of high-priority messages	0.002-0.005
λ_l	Arrival rate of low-priority messages	0.002-0.005
μ	Service rate	1/130
f	Transmition failure counter	0
δ	Starvation detection value	100
T	Execution time	1000

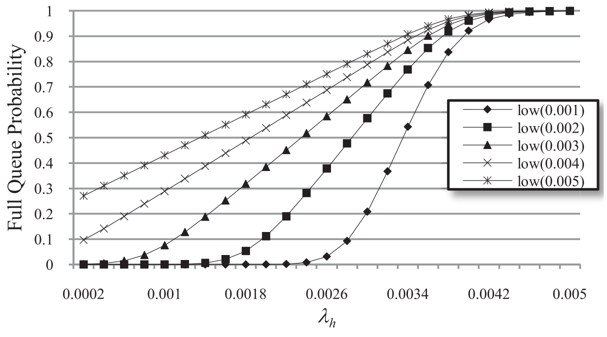

Fig. 4 The full queue probability of low priority messages quickly increases when high priority message arrival rate increases using without priority inversion scheme.

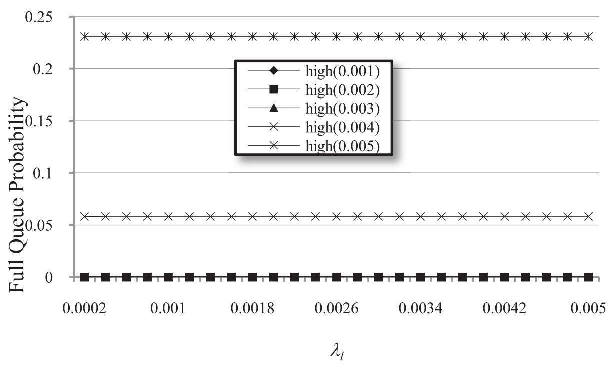

Fig. 5 The full queue probability for high priority messages maintains a horizontal level when the low priority message arrival rate increases using no priority inversion scheme.

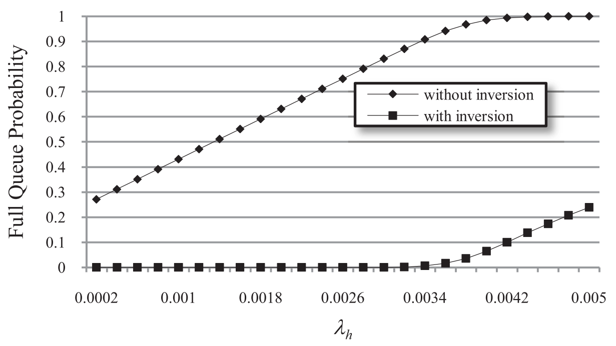

Fig. 6 The full queue probability of low priority messages increases when high priority message arrival rate increases from 0.0002 to 0.005 and low priority message arrival rate keeps at 0.005 with/without priority inversion scheme.

ority messages is close to 1 when the high priority message arrival rate is larger than 0.0042. We also observe the full queue probability, as shown in Fig. 5 when the low priority message arrival rate is from 0.0002 to 0.005. We know that the full queue probability for high priority messages is slightly impacted when the low priority message rate increases. Hence, the full queue probability for high priority messages maintains a close horizontal level and its value is under 0.25. Furthermore, three curves remain close to zero when the high priority message arrival rates are 0.001, 0.002, and 0.003, respectively. However, high-priority messages have a lower probability than low-priority messages due to execute first order.

4.2 With Priority Inversion

In the previous subsection, we evaluated the priority message experiment without priority inversion. Low-priority messages are placed in an unfavorable position compared with high-priority messages. An extreme starvation problem will probably occur. In this work, a simple priority inversion algorithm is proposed to prevent this problem. Figure 6 demonstrates that the probability in the full queue state will be reduced when priority inversion is adopted and the counter is set to 100. We extend the example mentioned in

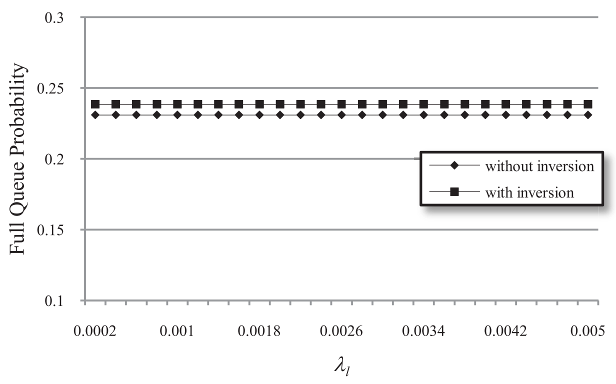

Fig. 7 The full queue probability of high priority messages increases when low priority message arrival rate increases from 0.0002 to 0.005 and high priority message arrival rate keeps at 0.005 using with/without priority inversion scheme.

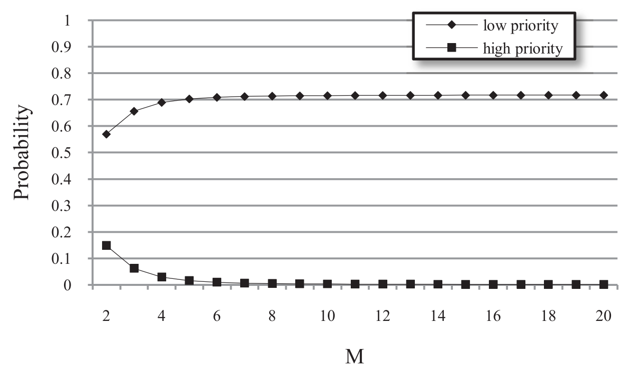

Fig. 8 The full queue probabilities between low and high priority messages with priority inversion are compared increasing M to 20 from 2 when the two message arrival rates are 0.003 in spite of high priority and low priority messages.

the previous subsection to add a counter module for priority inversion. The low-priority node wants to send a message to the bus but fails. The counter in the inversion module will be increased. The inversion time is up when the counter counts to M. The negative effects on high-priority traffic are shown in Fig. 7. From the figure, we know that this effect is very small, within 0.01. The full queue probability for high-priority messages using the priority inversion scheme is higher than not using one. The result will cause delay for high priority messages. In addition, no lost message is assumed in these experiments when the full queue probability is smaller than 1. We observe that the counter impacts on the full queue probability as shown in Fig. 8. We found that the impact on steady state behavior is close to zero when M is larger than 6.

4.3 Starvation Evaluation

Our scheme has the starvation freedom property. To evaluate if starvation occurs, we adjust the arrival rate in high priority messages and observe the starvation probability when low priority messages are fixed at a higher arrival rate 0.005. Based on Definition 3.2, we designed a starving property in

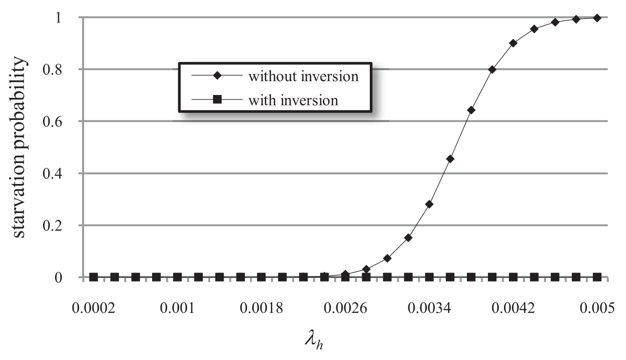

Fig. 9 The starvation probability of low priority messages in the long-run is evaluated using both schemes with/without inversion when the high priority message arrival rate is changed from low (0.0002) to high (0.005).

Table 2 Characteristic comparisons.

Schemes	hardware independence	protocol transparence	starvation freedom	Time critical	overhead
Murtaza [16]			√	√	extra node
Hasnaoui [14]	√			√	2 bits
Anwar [15]	√			√	*n* bits
Priority Inversion	√	√	√	√	1 bit

which "in the long-run, the starvation probability that the number of sending failure equals and exceeds a specified value." The probability for the starving property at steady state based on Definition 3.2 is shown as follows.

$$S = ?\ [f = \delta]. \tag{5}$$

Figure 9 illustrates the starvation probability for low priority messages when λ_h is changed from 0.0002 to 0.005, λ_l is 0.005, N is 10, M is 10, and the starving counter is 1000. These same values are smaller than those in previous experiments because the state-space explosion problem occurs. In this experiment, we found that the scheme without inversion will produce the starvation possibility when the high priority message arrival rate is larger than 0.0024. The starvation probability will reach 100% as λ_h is 0.005. In contrast, scheme with inversion do not occur in spite of λ_h being at low (0.0002) or high (0.005).

4.4 Comparisons

In this subsection, we compare our scheme and other proposed schemes mentioned in Sect. 1 as shown in Table 2. In order to fairly compare, four parameters are listed as follows.

(1) Hardware dependence vs. hardware independence: The proposed schemes can be divided into two categories. One is to use hardware devices to avoid starvation or to provide dynamic priority mechanism. The other is to modify protocol or software. A scheme using hardware component to solve the problem is called hardware dependence; otherwise called hardware independence.

(2) Protocol transparence: A scheme is transparent if the scheme follows the original defined protocol. In other words, the scheme is seamless to the protocol format is as same as defined in control area networks.

(3) Starvation freedom vs. time-critical first: Fixed priority schemes naturally starve low-priority messages. Although dynamic priority schemes can employ an adjustment priority mechanism to avoid starvation, all dynamic priority schemes are not starvation free because a scheme with starvation freedom must insure that no message is permanently blocked. Most dynamic priority schemes are time-critical first because these schemes give first time critical messages having a higher priority.

(4) Overhead: Overhead is redundant in order to complete dynamic priority or prevent starvation. This may be through hardware or software. For software, bit reservation is often used for dynamic schemes.

5. Conclusions

At the start of this paper, we stated that the CAN is dominant in the automobile and industry control because it has the advantage of message prioritization. In the CAN, the priority messages mechanism is implemented in the physical layer of the CAN protocol using open collector circuits. Although the message prioritization provides a fascinating advantage that high-priority messages certainly win when at least two CAN nodes send a message at the same time, the bigger problem of starvation potentially exists in its networks.

We discussed constructing a CAN model to evaluate the performance of our algorithm. A simple and useful scheme is provided in this paper. The scheme only uses one bit to change the original fixed priority mechanism into a dynamic process. A counter is designed to command the inversion priority to prevent the starvation created by low-priority messages. Next, we proved that our proposed algorithm is starvation freedom. We modeled our scheme using queueing systems. We found that an inverted low-priority message has a higher priority for processing. Finally, we used the PRISM tool to construct our analysis model. Through numerical results, our proposed algorithm is efficient in preventing starvation. However, the proposed algorithm is easy to implement in a chip due to its simple and efficient design.

According to the numerical results, our scheme is starvation free. Our scheme can be applied in applications with higher time critical constraints and fault tolerance. These applications include critical ECUs applied in intra-vehicle networks and smart components for industry monitoring. Our scheme's low priority messages smoothly descend throughput when a high priority component is present and continuously send its message into the CAN bus. In contrast, using the traditional scheme low priority messages have no chance to be sent because high priority messages generated a fault component dominate the CAN bus. Further work in this area will include implementing an intelligent gateway for advanced automobiles and constructing a more efficient mathematical model for verifying whether the gateway is safety, efficiency, and reliability or not. We will implement and research the technologies of embedded systems applied in automobiles and smart homes.

Acknowledgments

The author would like to thank System & Technology Corp., Taiwan for financially supporting this research under Contract No. 097006. The excellent comments of the anonymous reviewers are greatly acknowledged that have helped a lot in improving the quality and readability of this paper.

References

[1] D. Paret, Multiplexed networks for embedded systems CAN, LIN, FlexRay, Safe-by-Wire . . . , pp.14–19, Wiley, 2007.

[2] C. Dressler and O. Fischer, "CAN basics: CAN in automation history and CAN, part 1 of 8," Industrial Automation Asia, pp.22–23, Jan. 2007.

[3] R. Bosch, CAN Specification 2.0, Robert Bosch GmbH, Postfach, Stuttgart, Germany, 1991.

[4] B.J. Ahn, B.R. Park, Y.H. Ki, G.M. Jeong, H.S. Ahn, and D.H. Kim, "Development of a controller area network interface unit and its application to a fuel cell hybrid electric vehicle," SICE-ICASE International Joint Conference, pp.5545–5549, Oct. 2006.

[5] W. Liu, L. Gao, Y. Ding, and J. Xu, "Communication scheduling for CAN bus autonomous underwater vehicles," Proc. 2006 IEEE International Conference on Mechatronics and Automation, pp.379–383, June 2006.

[6] A.J. Pires, E.M. Leao, J.P. Sousa, L.A. Guedes, and F. Vasques, "Real-time communication for smart sensor networks: A CAN based solution," 5th IEEE International Conference on Industrial Informatics, vol.1, pp.553–558, June 2007.

[7] J.M. Giron-Sierra, C. Insaurralde, M. Seminario, J.F. Jimenez, and P. Klose, "CANbus-based distributed fuel system with smart components," IEEE Trans. Aerosp. Electron. Syst., vol.44, no.3, pp.897–912, July 2008.

[8] A. Bergnoli, A. Bertolin, R. Brugnera, E. Carrara, A. Cazes, F. Dal Corso, S. Dusini, G. Felici, A. Garfagnini, A. Longhin, U. Mantello, A. Mengucci, A. Paoloni, L. Stanco, V. Sugonyaev, F. Terranova, and M. Ventura, "The OPERA spectrometer slow control system," IEEE Trans. Nucl. Sci., vol.55, no.1, pp.349–355, Feb. 2008.

[9] S. Cavalieri, "Meeting real-time constraints in CAN," IEEE Trans. Industrial Informatics, vol.1, no.2, pp.124–135, May 2005.

[10] U. Klehmet, T. Herpel, K.S. Hielscher, and R. German, "Real-time guarantees for CAN traffic," Vehicular Technology Conference (VTC), pp.3037–3041, May 2008.

[11] M. Barranco, J. Proenza, G. Rodriguez-Navas, and L. Almeida, "An active star topology for improving fault confinement in CAN networks," IEEE Trans. Industrial Informatics, vol.2, no.2, pp.78–85, May 2006.

[12] M. Short and M.J. Pont, "Fault-tolerant time-triggered communication using CAN," IEEE Trans. Industrial Informatics, vol.3, no.2, pp.131–142, May 2007.

[13] E. Desa, P.K. Maurya, A. Pereira, A.M. Pascoal, R.G. Prabhudesai, A. Mascarenhas, R. Madhan, S.G.P. Matondkar, G. Navelkar, S. Prabhudesai, and S. Afzulpurkar, "A small autonomous surface vehicle for ocean color remote sensing," IEEE J. Ocean. Eng., vol.32, no.2, pp.353–364, April 2007.

[14] S. Hasnaoui, O. Kallel, R. Kbaier, and S.B. Ahmed, "An implementation of a proposed modification of CAN protocol on CAN fieldbus controller component for supporting a dynamic priority policy," 2003 38th IAS Annual Meeting, Conference Record of the Industry Applications Conference, vol.1, pp.23–31, 2003.

[15] K. Anwar and Z.A. Khan, "Dynamic priority based message scheduling on controller area network," International Conference on Electrical Engineering (ICEE '07), pp.1–6, April 2007.

[16] A.F. Murtaza and Z.A. Khan, "Starvation free controller area network using master node," ICEE 2008 Second International Conference on Electrical Engineering, pp.1–6, March 2008.

[17] W. Lawrenz, CAN System Engineering from Theory to Practical Applications, pp.86–87, Springer, 1997.

[18] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation, vol.4486 of Lecture Notes in Computer Science (Tutorial Volume), pp.220–270, 2007.

[19] A. Silberschatz and J.L. Peterson, Operating System Concepts, pp.163–164, Addison Wesley, 1988.

[20] G. Bolch, S. Greiner, H. de Meer, and K.S. Trivedi, Queueing networks and Markov chains: Modeling and performance evaluation with computer science applications, second ed., pp.272–279, Wiley Interscience, 2006.

[21] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A tool for automatic verification of probabilistic systems," Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06), vol.3920 of Lecture Notes in Computer Science, pp.441–444, Springer, March 2006.

[22] M. Kwiatkowska, G. Norman, and R. Segala, "Automated verification of a randomised distributed consensus protocol using cadence SMV and PRISM," Proc. CAV'01, vol.2102 of Lecture Notes in Computer Science, pp.194–206, Springer, Jan. 2001.

[23] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker, "A formal analysis of bluetooth device discovery," International Journal on Software Tools for Technology Transfer (STTT), vol.8, no.6, pp.621–632, Springer-Verlag, Nov. 2006.

[24] S. Basagiannis, P. Katsaros, A. Pombortsis, and N. Alexiou, "A probabilistic attacker model for quantitative verification of DoS security threats," Proc. 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC'08), pp.12–19, IEEE CS Press, 2008.

[25] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Model checking continuous time Markov chains," ACM Trans. Computational Logic, vol.1, no.1, pp.162–170, 2000.

Cheng-Min Lin was born in 1964. He received the B.S. and M.S. degrees in electronic engineering from National Taiwan University of Science and Technology, Taipei, Taiwan, in 1989 and 1991, respectively, and the Ph.D. degree in Department of Information Engineering and Computer Science, Feng-Chia University, Taichung, Taiwan. Currently, he is an Associate Professor in the Department of Computer and Communication Engineering, Graduate Institute of Electrical Engineering and Computer Science, Nan Kai University of Technology. His research interests include embedded systems, mobile computing, distributed systems, and telematics. Dr. Lin is a member of the IEEE Computer Society.