Leading Our World In Motion

2005-01-1538

SAE TECHNICAL
PAPER SERIES

Performance Analysis of Fault Tolerant TTCAN System

Aakash Arora and Syed Masud Mahmud
Electrical and Computer Engineering Department, Wayne State University

Reprinted From: In-Vehicle Networks and Software (SP-1918)

ISBN 0-7680-1633-9

9 780768 016338

**2005 SAE World Congress
Detroit, Michigan
April 11-14, 2005**

SAE International™

400 Commonwealth Drive, Warrendale, PA 15096-0001 U.S.A. Tel: (724) 776-4841 Fax: (724) 776-5760 Web: www.sae.org

The Engineering Meetings Board has approved this paper for publication. It has successfully completed SAE's peer review process under the supervision of the session organizer. This process requires a minimum of three (3) reviews by industry experts.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of SAE.

For permission and licensing requests contact:

SAE Permissions
400 Commonwealth Drive
Warrendale, PA 15096-0001-USA
Email: permissions@sae.org
Tel: 724-772-4028
Fax: 724-772-4891

All SAE papers, standards, and selected books are abstracted and indexed in the Global Mobility Database.

For multiple print copies contact:

SAE Customer Service
Tel: 877-606-7323 (inside USA and Canada)
Tel: 724-776-4970 (outside USA)
Fax: 724-776-1615
Email: CustomerService@sae.org

ISSN 0148-7191
Copyright © 2005 SAE International

Positions and opinions advanced in this paper are those of the author(s) and not necessarily those of SAE. The author is solely responsible for the content of the paper. A process is available by which discussions will be printed with the paper if it is published in SAE Transactions.

Persons wishing to submit papers to be considered for presentation or publication by SAE should send the manuscript or a 300 word abstract to Secretary, Engineering Meetings Board, SAE.

Printed in USA

2005-01-1538

Performance Analysis of Fault Tolerant TTCAN System

Aakash Arora and Syed Masud Mahmud
Electrical and Computer Engineering Department, Wayne State University

Copyright © 2005 SAE International

ABSTRACT

Continuous demand for fuel efficiency mandate "Drive-by-Wire" systems. The goal of Drive-by-Wire is to replace nearly every automotive hydraulic/mechanical system with electronics. Drive-by-Wire and active collision avoidance systems need fault tolerant networks with time triggered protocols, to guarantee deterministic latencies. CAN is an event triggered protocol which has features like high bandwidth, error detection, fault confinement and collision avoidance based on message priority. However, CAN do not ensure message latency, which is critical for real time application. TTCAN (Time Triggered CAN) removes this fallacy of CAN by providing exclusive time windows for those messages that need deterministic latencies. In addition to the exclusive windows, there are arbitration windows too, which make way for event triggered communications. In TTCAN, if an error occurs within an exclusive or arbitration window, retransmission of the message is not allowed. If the message that encountered the error is a safety critical message, then the transmission error can compromise the safety of the vehicles. In this paper, we propose a fault tolerant TTCAN system that uses a secondary bus to tolerate faults on the primary bus. To keep the cost down, we can use the same secondary bus to connect various partitions in the in-vehicle network. Each partition of the network takes care of a particular type of functionality of the vehicle. Thus the same secondary bus can tolerate faults on the primary busses of various partitions. The paper will show analysis done on a realistic TTCAN system. Our results show that even using a low bandwidth secondary bus, the performance of a Drive-by-Wire system can be significantly improved under various types of transmission errors on the primary busses.

INTRODUCTION

We are living in an era where cost of electronics in cars is 50% of total cost and is increasing by 9-16% each year [1]. The number of microprocessors and sensors keep on increasing exponentially. Current number of microprocessors in a car varies from 70 to 100 [2, 3]. Driving force behind all this is the increase in the technological innovations, 90% of functional innovation in vehicles being in the field of electronics [2, 3]. These microprocessors and sensors are used in brakes, suspension, steering, emission control, engine control, and many other critical functions inside a vehicle [4, 5]. Body electronics also make extensive use of actuators and sensors. Use of sensors and actuators is also expanding to the upcoming areas like diagnostics and entertainment.

For example, researchers at the Department of Energy's Pacific Northwest National Laboratory are developing TEDANN, or Turbine Engine Diagnostics Using Artificial Neural Networks, for the Army. TEDANN uses data from 32 existing sensors on the gas turbine engine along with 16 new sensors that have been added using a wiring harness. These sensors continually monitor engine performance and relay the information back to a computer processor for analysis. A mechanic can access TEDANN's analysis using a laptop computer. He can monitor the sensor data that has been collected (for example, engine temperatures, fuel levels, pressure, throttle and speed) and view diagnostic reports without having to remove the engine from the hull. Also, the tank's driver can be alerted to critical engine problems with an onboard display [6].

In order to be effective, these sensors, actuators and microprocessors in a vehicle must communicate using appropriate network protocols. A fine comparison of 40 such protocols that exist since early 1980's can be found in [7], [8] and [9]. By far CAN (Controller Area Network) has been the most dominating protocol in Europe and is now being used in USA too. The fact that last year almost 300 million CAN microchips were sold [10] establishes the worldwide acceptance of CAN. CAN was invented by Robert Bosch GmbH in mid 1980's. The basic features of CAN like high speed serial interface, low cost physical medium, short data lengths, fast reaction times, multi-master and peer-to-peer communication, error detection and correction [11] make it a common choice for many automobile designers and vendors.

Factors like cost, stability, reliability and safety demand a distributed safety critical real-time computing platform [3]. But the event-triggered nature of CAN prevents it from being used for deterministic real-time communications. Message latency is not guaranteed in

CAN and increases with higher bus message traffic load [7]. This pushed research on time based scheduling of messages for in-vehicle networks in the early 1990's. As a result, various time-triggered architectures started to emerge. One of these, TTCAN, is based on unchanged standard CAN protocol [12]. Much literature is now available that describes TTCAN in detail [12-14].

TTCAN defines a system matrix that consists of basic cycles. Each Basic cycle further consist of *time windows*. These windows are categorized into *exclusive windows*, *arbitrating windows* and *free windows*. A *frame synchronization entity* has all the time triggered information and is responsible for triggering specific messages in their corresponding exclusive time windows. Event triggered messages follow standard CAN arbitration and are scheduled for arbitration windows. Free windows are reserved for future expansion of network. A global timer synchronizes all local timers at the start of each basic cycle using a message known as the reference message.

Figure 1 illustrates the abovementioned scheme. TTCAN is now accepted as ISO CD 11898-4(draft). Software independent hardware for testing TTCAN is already available [15, 16].

Fault Tolerance in TTCAN

Using TTCAN it is possible to carry out deterministic communication without losing the event triggered nature. This is possible because of the existing arbitrary windows and exclusive windows.

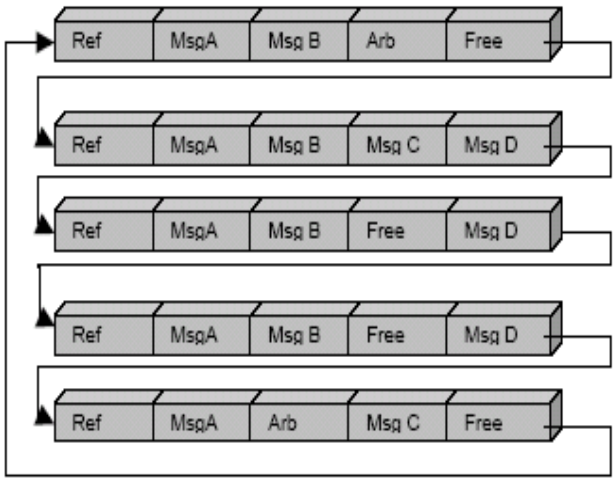

Figure 1: System Matrix

However, the above described scheme has a flaw in it. Standard CAN protocol retransmits a message if an error occurs during transmission of the message. To protect the definitive latency of messages this feature was taken out in TTCAN. TTCAN does not allow automatic retransmission of messages during exclusive windows and arbitrating windows [13]. This implies if error occurs during these windows, messages will be lost. Since, as discussed in the beginning, communication of safety critical messages is one of the main purposes of using TTCAN. Loss of safety critical messages defeats the purpose of a new protocol, and much more than that puts the life of passengers at risk. Though research is ongoing, no systematic fault tolerance schemes exist for TTCAN [16].
Table III [26], makes it clear that without fault tolerant TTCAN it is possible to have up to 2480 failures in an hour. In such a case the life of the passenger is at risk. We simply cannot afford to have such high failure rates. Thus introduction of fault tolerance in X-by-Wire systems is of extreme importance.

Error Rate in TTCAN

According to [23] for X-by –wire systems a failure rate of better than 10^{-9} is required. But in reality not much actual data for error conditions and rates is available [24]. For most analytical purposes, bit error rates of 10^{-4} (worst case) to 10^{-6} have been considered. The most practical data is available in [26] as shown *Table I*. These data were collected under conditions shown in *Table II*.

Table I: Bit error rates

	Benign environment	Normal Environment	Aggressive Environment
Bits Transmitted	2.02×10^{11}	1.98×10^{11}	9.79×10^{10}
Bit errors	6	609	25239
Bit error rate	3.0×10^{-11}	3.1×10^{-9}	2.6×10^{-7}

Table II: Testing environment

Available Bus Bandwidth	1Mbps
Used Bus Bandwidth	250Kbps
Length of CAN bus	30m
Time slot	400 Microseconds
Data frame	8 bytes

Table III: Failures per hour

Bit error rate	Failure per hour
10^{-4}	2840
10^{-5}	286
10^{-6}	2.87

The paper cited above mentions that these tests were carried out in academic conditions (benign), in normal factory conditions (normal) and in a factory with arc welding taking place at a distance of 2 meters (aggressive).

Comparison of previous works

Other Time Triggered Architecture (TTA) based protocols like TTP/C and Flexray provide fault tolerance by providing redundant channels [16, 17]. The whole scheme of TTP/C and comparison with other TTA based protocols with TTP/C is described in [16, 17].

Since late 1990's, studies have been carried out to come up with a systematic fault tolerant approach for TTCAN. Most of these look at redundant buses from different angles. One research suggests using a redundant bus having time slots that match the time slots on the primary bus [18]. Additional slots in each basic cycle are required. These additional slots are known as diagnostic and monitor time slots. These time slots detect errors on busses and based on these errors action are taken. Redundant bus is also used for load balancing. One approach suggests using two busses, one for time and safety critical messages and other for low priority messages. Another approach [19] also suggests redundant TTCAN bus architecture. It describes topologies that might be used and then goes on to describe various methods for synchronization of the redundant busses with the primary bus.

All these approaches require synchronization and do not discuss as to how the basic error containment mechanism, used by standard CAN, will be handled (error active, error passive and bus off stages of a node). In certain cases, it is suggested to adopt fail silent approach that will not help in managing the error counters of the nodes.

Another model is used in [25] where no redundant bus is used at all; instead attempts have been made to ensure fault tolerance by sending each frame twice. In the above-mentioned paper, comparisons have been made between the cases where a frame is sent only once and the case where frames have been sent twice. The results show improvement in the fault tolerant behavior (by decreasing the probability of failure) in the case where frames are sent twice.

In this paper here, we will try to induce fault tolerance in a TTCAN system by approaching it in two different methods as discussed below.

In both the proposed approaches we make an assumption that for any practical fault tolerant system, *occasional delivery failures are acceptable and expected* and that hard deadlines cannot be met in any form of electrical communication that is subject to unpredictable faults [25].

Method 1:

Proposed Architecture

The physical layer contains the transceivers and the transfer medium; above these is the TTCAN controller that enforces the protocol. The controller is supervised by a software layer which supervises receive and transmit functions. The TTCAN controller maintains two receive and transmit queues. The primary and secondary transmit queues are represented by P_Tx_Q and S_Tx_Q, respectively. Similarly, the primary and secondary receive queues are P_Rx_Q and S_Rx_Q. Above mentioned architecture is shown in Figure 2.

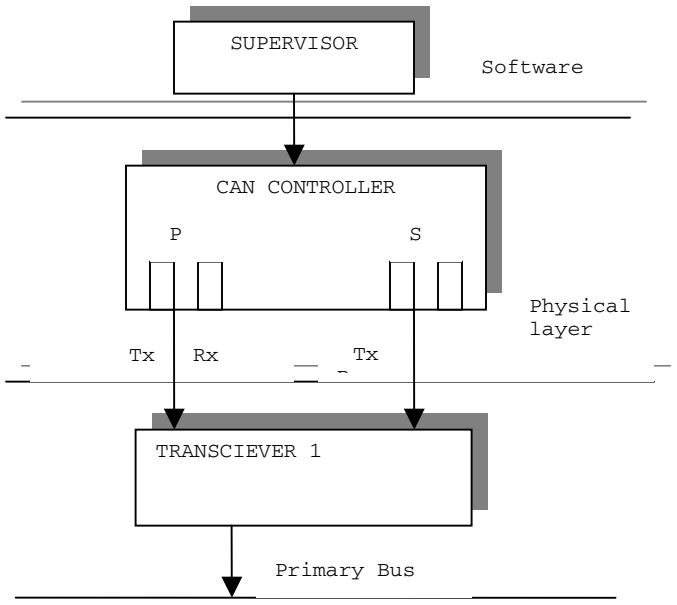

Figure 2: Proposed architecture

Algorithm

The proposed algorithm for ensuring the delivery of safety critical messages is built on the top of TTCAN. In the proposed system-matrix design, each basic cycle contains a time window at its end. We will call this window the MAILBOX WINDOW. If any message in a basic cycle is not transmitted due to an error, the supervisor schedules it for mailbox window of current basic cycle. In a real-time scenario, this window acts like a mailbox for the messages that couldn't go through. Using the data provided above, it can be proved easily that there cannot be more than one error message in a basic cycle. If the bit error rate is considered to be in the order of 10^{-7}, for two messages to be corrupted in one basic cycle, a basic cycle should be 2×10^{7} messages long, i.e., 2.2 hours long (as per message and bus conditions of [26]). Thus we can comfortably consider at the most one error message per basic cycle. Hence one mailbox window per basic cycle is sufficient. The proposed system matrix is considered in Figure 3.

Figure 3: Proposed system matrix

Architecturally, only changes required are in the role of supervisor and adding extra a pair of Tx and Rx queues,

as shown in Figure 2. When there is no error the whole system behaves in normal fashion, as described in [12-14].
The frame synchronization entity controls the whole system of exclusive, arbitrating and free windows. But in case of error, the role of supervisor increases. As an error occurs during the transmission of a message in primary bus, the supervisor sends an error frame on the primary bus while it queues the failed message in S_Tx_Q. Generally most CAN chips maintain their Tx queues on a FIFO basis [21]. In our case also, the supervisor schedules the transmission of the messages queued in S_Tx_Q for the mailbox window of a current basic cycle. The S_Tx_Q is flushed on start of each system matrix to prevent the overflow of messages from one system matrix to another, if it happens in a remote case. The following flowchart in Figure 4, describes the entire algorithm.

Analysis

The performance parameters monitored in the proposed scheme are probability of failure and delay in message transmission. The aim here is to minimize these parameters to achieve reliable performance.

As per [25], the probability of failure can be calculated as described below.

If there are *m* events in time *t*, according to Poisson distribution, the probability of successful delivery P_t:

$$P_t = \frac{e^{-\lambda t}(\lambda t)^m}{m!}$$

Where λ represents failures per second

In our case m=0

Thus P=1- $e^{-\lambda t}$ (1)

Thus probability of unsuccessful delivery is

$$P = 1 - P_t$$
$$P = 1 - \frac{e^{-\lambda t}(\lambda t)^m}{m!}$$

Also *t* is the time for transmission of a message. As per [27]

$$t_{data} = (l_{fix} + l_{data} + 1 + \left\lfloor \frac{l_{fix} - l_{stuff} + l_{data}}{l_{stuff} - 1} \right\rfloor + l_{efs}).t_{bit} \quad (2)$$

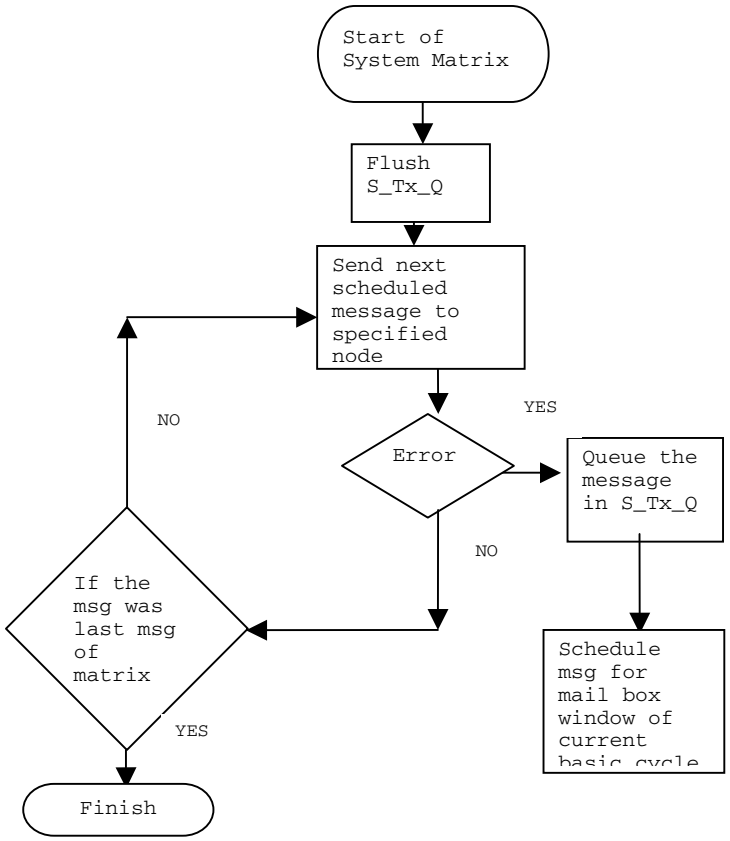

Figure 4: Proposed Fault Tolerant algorithm

Where
l_{fix} =length of fixed size bits in a message subject to bit stuffing
l_{data} =length of data field
l_{efs} =length of bits which are not subjected to bit stuffing
l_{stuff} = number of stuff bits
t_{bit} =bit timing

Also for error frames

$$t_{error} = (l_{.flag} + l_{del}).t_{bit} \quad (3)$$

Length of a time window in the worst case, $C_{i,}$ can be calculated by adding (2) and (3).

$$C_i = t_{window} = t_{data} + t_{error} \quad (4)$$

Substituting the value of C_i in (1)

$$P = 1 - e^{-\lambda C_i} \quad (5)$$

We can extend (5) to state that the probability of an unsuccessful delivery in *n* attempts is

$$P = (1 - e^{-\lambda C_i})^n \quad (6)$$

Now by adding a mailbox window in a basic cycle with *N* time windows we can decrease this probability by following order

$$P = (1 - e^{-\lambda Ci})^{n+1/N} \quad (7)$$

Mathematically this can be viewed as giving each time window another *1/N* attempt of to deliver a message.

For the analysis purposes we have taken λ as 0.7, 0.08 and 0.00079. These values are proved in [26]

If a mailbox window is not considered, a message that encountered error will be transmitted after a time period equal to its periodicity. However if the mailbox window is present in the basic cycle, this message can be transmitted earlier.
This improvement in quality of service is expressed as following:
Consider a basic cycle in which window *i* has length in time as t_i. If an error occurs in the n^{th} window of a basic cycle then the message will be delivered in the mailbox window. Thus the delay in time is

$$D = \sum_{i=n+1}^{N} ti = \sum_{i=0}^{N} ti - \sum_{i=0}^{n} ti \quad (8)$$

If periodicity of the message is T_i, then improvement is:

$$\text{Improvement} = \frac{D}{Ti} \quad (9)$$

Results and Discussions

The results have been shown in Figure 5, Figure 6 and Figure 7. Figure 5 illustrates the decrease in probability of unsuccessful delivery with and without mailbox window. This is studied over various message lengths (to consider effect of Ci). Speed of bus considered is 512 Kbps and the number of windows in basic cycle is 10. Bus load can be reduced almost 50% by using this technique as compared to double scheduling method suggested in [25].

The graph clearly shows that the addition of mailbox window brings the probability of unsuccessful delivery down as compared to the case when mailbox window not considered.

Figure 6 compares the effect of length of basic cycle on the probability of unsuccessful message delivery. From the figure, it is clear that with the decrease in the length of basic cycle, the probability of failed delivery decreases.

Figure 7 compares various value of λ as per Table 3, to study the effect on probability of failed delivery. As expected, with decrease in λ, P decreases.

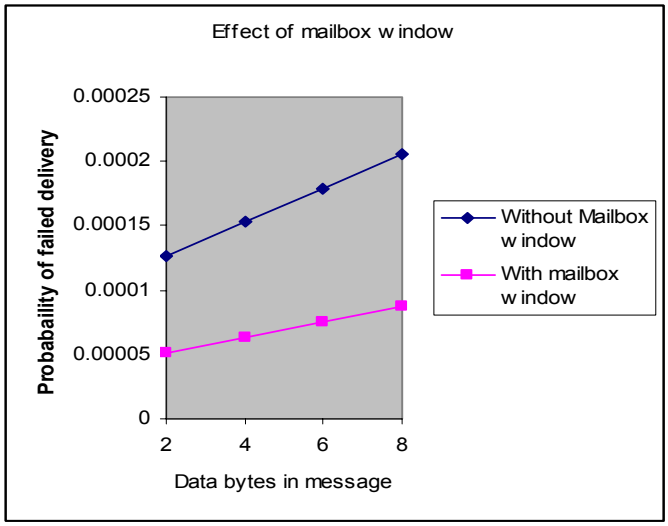

Figure 5: Effect of mailbox window

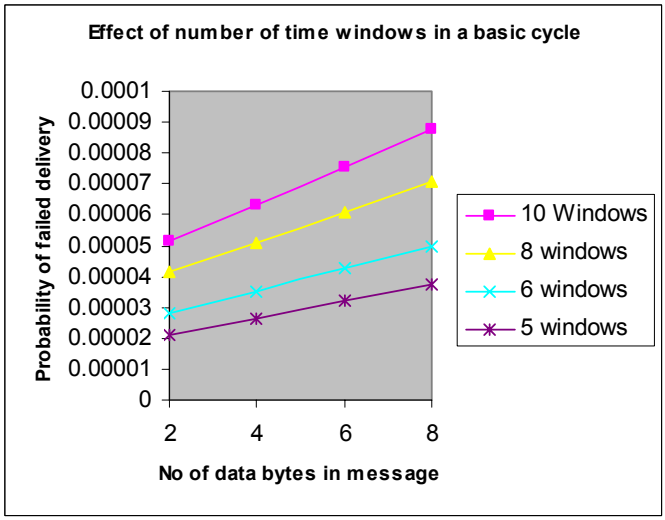

Figure 6: Effect of basic cycle length (mailbox present)

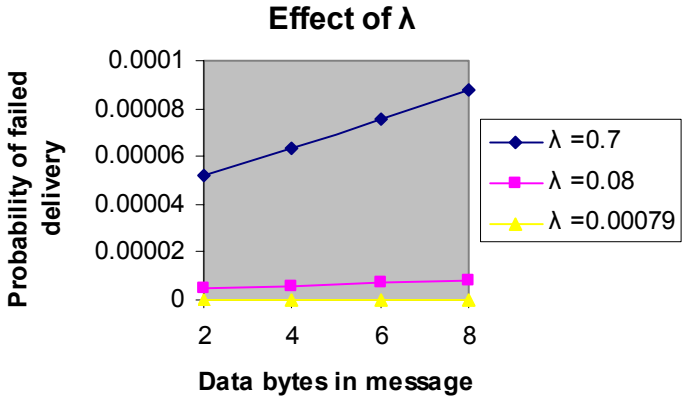

Figure 7: Effect of λ (in presence of mailbox)

From the results shown in Figure 5-7, it can interpreted that the proposed system is able to induce systematic fault tolerance by delivering all safety critical messages

with in the same basic cycle. The error handling mechanism of standard CAN is not compromised. The system does not require any redundant bus or custom built hardware thus substantially reducing the development as well as implementation costs.

Method 2:

Here we consider a normal TTCAN with a non-synchronized redundant bus as shown in Figure 8. No mailbox window is considered in this system matrix.

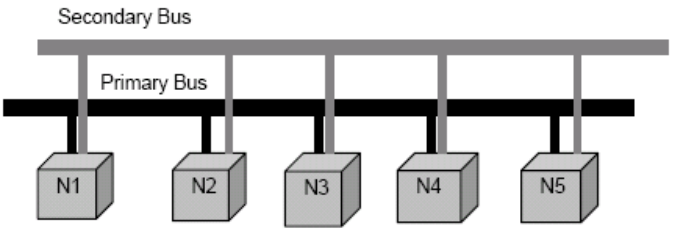

Figure 8: TTCAN system with redundant bus

The redundant bus can be a CAN bus, LIN bus or a MOST bus. It need not be a dedicated bus; it can be used on a bandwidth sharing basis. The speed of the redundant bus can differ from the speed of the primary bus. No synchronization is required.

Just like previous method, this method also accepts the graceful degeneration of system for the sake of fault tolerance.

Proposed Architecture

This architecture considers two CAN controllers at a node; one for each bus. Each CAN controller has a transceiver. Frame synchronization entity keeps sending messages as per predetermined time sequence. In case of errors, it notifies the supervisor which immediately puts error message on primary bus and tries to send this failed message through the second CAN controller. This architecture is shown in Figure 9.

Analysis

Our interest here is in the delay involved in case of an error. Let's assume in a basic cycle an error occurs in time window *n* at time *t1*. The supervisor will redirect this message to secondary bus using the second controller instantaneously. The best-case scenario is that the secondary bus is fault free and is not pre-occupied thus readily available for transmission of this message (worst case scenarios are discussed later). Depending upon the speed of the secondary bus the message will be received at time *t2* by receivers. If the expected time of delivery without occurrence of error were *t3*, then

Delay =d = t2-t3. (10)

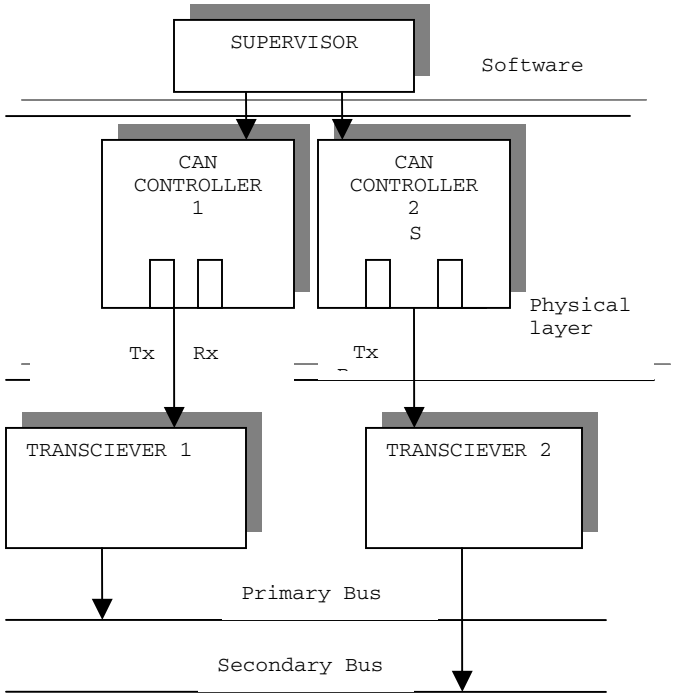

Figure 9: Proposed TTCAN architecture with redundant bus

This delay is illustrated in Figure 10.

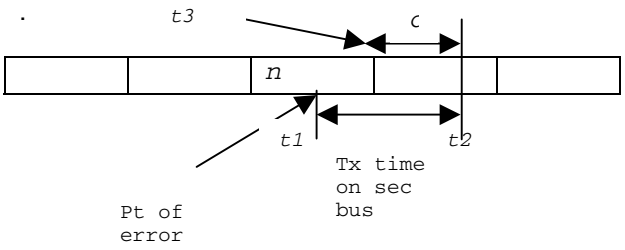

Figure 10: Calculation of delay

In the worst case scenario we will have to consider that messages have delay due to wait because of reasons like other messages in queue, successive network faults (burst), jitter, etc. A detailed explanation of these is available in [25].

Results and Discussion

Best-case scenario

As mentioned here the secondary bus is readily available for transmission of message that was faced with error.

We have considered messages of different data frame sizes being transmitted over a 512Kbps bus. The occurrence of error is considered midway between normal message transmissions. On error, these messages are considered to be sent over the secondary bus, as mentioned in analysis. We have considered various speeds of secondary bus. The results based on equation 10 are shown in Figure 11.

Table IV: Worst case response time from [25] for different message lengths.

Length (Microsecond)	Period (Microsecond)	Deadline (Microsecond)	WCRT (Microsecond)
288	2000	2000	828
328	4000	4000	1168
328	4000	4000	1508
528	8000	8000	2048
248	12000	12000	2608
528	240000	240000	2320

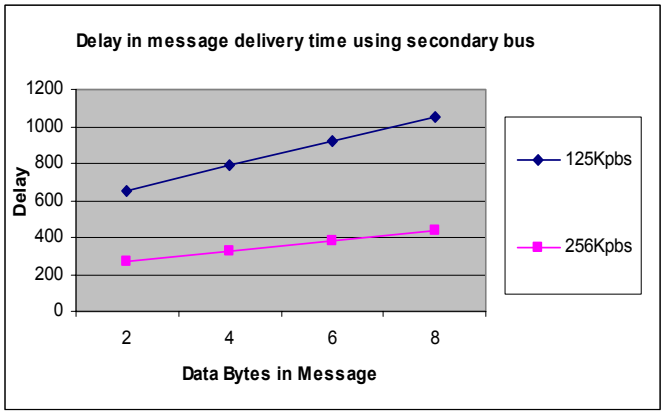

Figure 11: Delay analysis with different secondary bus speeds (delay in microseconds)

As shown in Figure 11, it is possible to resend the safety critical message through a secondary bus within acceptable limits.

Worst Case Scenario

As mentioned earlier, for worst-case analysis, it is necessary to consider factors like the wait times, the faults of secondary bus, their frequency, bus utilization and the jitter involved. Considering secondary bus of 256 Kbps and 41% bus utilization, the worst-case response times for various messages on secondary busses are displayed in table IV [25].

Results of both best and worst case scenarios indicate that worst-case delay is in order of 2 milliseconds. From a real world perspective these should be totally acceptable. For example, if a car moving at 75 mph receives a message that faced error, 2 milliseconds after its deadline, it would have moved only 2.6 inches in that time. When compared to not receiving brake messages at all, and putting life of passenger at risk, this improvement is significant.

In addition to that, both of the messages suggested above do not compromise the basic fault containment mechanism of CAN. Thus the error active, error passive and bus off modes are still in play taking care of faulty nodes.

Conclusion

For the safety of passengers it is critical to have fault tolerant systems. The paper proposes two methods of inducing systematic fault tolerance in a TTCAN system. One uses a special system matrix design and the other uses non-synchronized redundant bus architecture. The proposed schemes have been verified analytically under varying worst-case window timing for different data loads, basic cycle lengths, fault per second rates, and bus speeds. The results are positive and encouraging. These methods do not compromise the existing CAN fault containment. Also, proposed methods are relatively easier to implement and more cost effective as compared to prevailing techniques. The efforts to implement these methods on hardware are going on at our laboratory.

REFERENCES:

1. Karen Parnell, “Automotive Electronics Digital Convergence-How to Cope with Emerging Standards and Protocols”, AMAA 2004, Berlin.

2. Stephen Channon and Peter Miller,” The Requirements of Future In-Vehicle Networks and an Example Implementation”, SAE Technical Paper Series 2004-01-0206.

3. Rienhard Maier, et al,” Time Triggered Architecture: A Consistent Computing Platform “, IEEEmicro July/Aug 2002.

4. Patrick Leteinturier, et al,” TTCAN from applications to products in automotive”, SAE Technical Paper Series 2003-01-0114.

5. Maria Bruce,” Distributed Brake–By-Wire Based on TTP/C”, ISSN 0280-5316 ISRN LUTFD2/TFRT-5668 SE.

6. http://sciencedaily.com/releases/1998/11981103141 5.htm

7. S Shaheen, D Heffernan and G Leen,” A Comparison of Emerging Time Triggered Protocols for X-by-wire Control Networks”, Proc. Instn Mech. Engrs Vol 217 PartD: J.Automobile Engineering.

8. Christopher A. Lupini,”Multiplex Bus Progression 2003”, SAE Technical Paper Series 2003-01-0111.

9. G Leen and D Heffernan,” Expanding Automotive Electronic Systems”, IEEE Computer Jan 2002 P.88

10. Naill Murphy," A Short trip on the CAN bus", Embedded System Programming (8/11/03), embedded.com.

11. M.Farsi, et al," An overview of CAN", Computing and Control Engineering Journal, June 1999.

12. Florian Hartwich, et al," Integration of Time Triggered CAN (TTCAN_TC)", SAE Technical Paper Series 2002-01-0263.

13. Thomas Fuehrer, et al," Time Triggered Can (TTCAN)", SAE Technical Paper Series 2001-01-0073.

14. Holger Zeltwanger,"Time-Triggered communication on CAN", SAE Technical Paper Series 2002-01-0437.

15. http://www.can.bosch.com/content/TT_CAN.html

16. www.tttech.com/technology/docs/protocol_comparisons/TTTech-Comparison_TTP-TTCAN-FlexRay.pdf

17. http://www.tttech.com/technology/docs/fault_handling/TTTech-Fault-Handling-TTA.pdf

18. Matjaz Colnaric, Domen Verber," Communication Infrastructure for IFATIS Distributed Embedded Control Application", RTN 2004 - 3rd Int. Workshop on Real-Time Networks, Catania, Italy.

19. B.Müller, et al,"Fault Tolerant TTCAN networks", 8th iCC Las Vegas, 2002.

20. Aakash Arora, et al," A Fault Tolerant Time Triggered Protocol for Drive-By-wire Systems", 4th Annual Intelligent Vehicle Systems Symposium, June 2004, Traverse city, Mi.

21. http://www.engin.umd.umich.edu/ceep/reports/200MidYearRichardson01.html

22. Richard T. McLaughlin and Chris Quigley,"Analysis and Diagnostics of Time Triggered CAN (TTCAN) Systems. SAE Technical Paper Series 2004-01-0201

23. Cedric Wilwert,et al, "Impact of Fault Tolerance Mechanisms on X-by-wire System Dependability ",TRIO report 2003.

24. Guillermo Rodriguez-Navas, et al, "Harmonizing Dependability and Real Time in CAN Networks", RTLIA2003 - 2nd International Workshop on Real-Time LANs in the Internet Age.

25. Ian Boster, Alan Burns, et al,"Comparing Real-Time Communication under Electromagnetic Interference", 16th Euromicro Conference on Real-Time Systems (ECRTS'04), 2004 Catania, Italy

26. Joaquin Ferreira , Arnaldo Oliveria ,et al," An Experiment to Assess Bit Error Rate in CAN", RTN 2004 - 3rd Int. Workshop on Real-Time Networks, Catania, Italy.

27. José Rufino,"An Overview of the Controller Area Network" Proceedings of the CiA Forum - CAN for Newcomers, January 1997, Braga, Portugal.

ABBREVIATIONS

CAN: Controller Area Network
TTCAN: Time Triggered CAN
TTP/C: Time Triggered Protocol, Class C
LIN: Local Interconnect Network
MOST: Media Oriented System Transport