Towards analyzing the fault-tolerant operation of Server-CAN

Thomas Nolte†, Guillermo Rodríguez-Navas‡, Julián Proenza‡, Sasikumar Punnekkat†, Hans Hansson†
†MRTC, Department of Computer Science and Electronics, Mälardalen University, Västerås, Sweden
email: {thomas.nolte, sasikumar.punnekkat, hans.hansson}@mdh.se
‡Departament de Matemàtiques i Informàtica, Universitat de les Illes Balears, Palma de Mallorca, Spain
email: {guillermo.rodriguez-navas, julian.proenza}@uib.es

Abstract

This work-in-progress (WIP) paper presents Server-CAN and highlights its operation and possible vulnerabilities from a fault tolerance point of view. The paper extends earlier work on Server-CAN by investigating the behaviour of Server-CAN in faulty conditions. Different types of faults are described, and their impact on Sever-CAN is discussed, which is the subject of on-going research.

1 Introduction

The usage of communications in embedded real-time systems is nowadays very common as these systems are typically distributed. A number of communication standards are available, e.g., [5, 8, 9, 12]. One of the more common real-time networks for embedded systems is the Controller Area Network (CAN) [6]. This network can be found in many application domains; however, automotive is its stronghold. The requirements set by distributed applications on their network include fault-tolerance and real-time. Many protocols have been developed on top of CAN to enhance these requirements [1, 7, 11]. Server-CAN [10, 11] is an approach with real-time capabilities and fairness properties, scheduling both periodic and aperiodic messages using servers. This work-in-progress (WIP) paper extends the work on Server-CAN by investigating its operation from the point of view of fault tolerance.

Server-CAN has been proposed to support communications in flexible open systems where users of the system might be added, removed and updated during runtime. This flexibility is provided by the Server-CAN admission control mechanism. Servers are providing bandwidth isolation among users of the network. The network is scheduled as one resource using server-based techniques and a centralised scheduler. Server-CAN can therefore easily support mode changes where the system go from one operating mode to another, possibly completely change the usage of the network. Moreover, the adding, removing and updating of users does not affect the guarantees provided to the other users. In total, the above inherent properties of Server-CAN, such as flexibility and robustness, if combined with appropriate mechanisms for ensuring continued and consistent transmissions, can provide fault-tolerant real-time communications over CAN.

The system may suffer from different types of faults, namely, node software faults, node physical faults, channel permanent faults and channel transient faults. The impact of these faults on Sever-CAN is discussed. Inherent mechanisms of Server-CAN in order to achieve fault-tolerance, as well as mechanisms which could be easily incorporated, are also discussed.

The paper is organised as follows: Server-CAN is presented in Section 2, with a focus on its relevant, from a fault-tolerance point of view, protocol mechanisms. Then, Section 3 presents the fault model used in this paper and discusses Server-CAN behaviour in presence of errors. Finally the paper is concluded in Section 4.

2 Server-CAN

In Server-CAN, bandwidth is allocated to users of the network by the usage of servers called network access servers (*N-Servers*). Each node has one or more N-Servers allocated to it. The N-Servers have exclusive associated bandwidth in terms of capacity, C, and a period time, T. Moreover, all N-Servers have an associated deadline in order to be scheduled for message transmission. Time is divided into Elementary Cycles (ECs), similar to the FTT-CAN [1]. The length of an EC is denoted T_{EC}. The N-Server period is required to be an integer multiple of T_{EC}.

All N-Servers have a local message queue, in which all its user messages are stored. A *user* is a stream of messages, e.g., the sending of messages by an application, and can be of either periodic or aperiodic nature.

The scheduling is performed at a specialised master node, called the *M-Server*. The M-Server keeps all information regarding the N-Servers in the system. The M-Server is updating the N-Server deadlines according to the scheduling policy in use. As soon as an N-Server is being scheduled for message transmission, the N-Server selects a message from its local message queue. Since each N-Server has ex-

clusive right to a share of the total system bandwidth, all users sharing an N-Server will share this portion of bandwidth. Hence, depending on the type of queue used at the N-Server, e.g., FIFO or priority-based, different guarantees of timeliness can be offered. Suppose a priority-based queue is used, then the users will experience a service, in terms of timeliness, similar to the one of an exclusive network, essentially with the only difference being the lower bandwidth offered. Hence, the timely behaviour will, compared to an exclusive CAN network, be divided by C/T, i.e., the server's share. A variant of the response-time analysis for fixed priority systems could be used to calculate the timing properties.

The M-Server and all N-Servers are sending their messages to their corresponding CAN controller, where all messages are scheduled according to the native CAN message arbitration mechanism. In this paper, all CAN controllers are assumed to have an infinite message buffer that is ordered based on message identifiers (i.e., message priorities).

2.1 Scheduling mechanism

The server-scheduling mechanism is depicted in Figure 1.

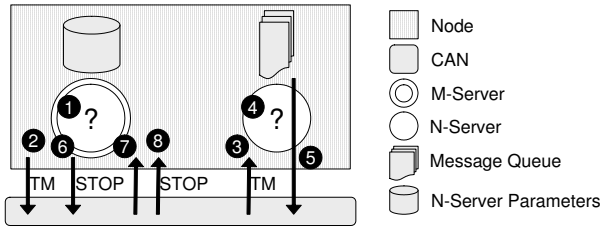

Figure 1. Server-scheduling mechanism.

The scheduling is performed at the M-Server according to the Earliest Deadline First (EDF) policy (Figure 1:1, i.e., (1) in Figure 1). A schedule is created containing the N-Servers with the earliest deadlines, filling up one EC. As this is done, the schedule is put into a trigger message (TM) and sent to all the N-Servers in the system (Figure 1:2).

When the N-Servers receive the TM (Figure 1:3), they will read it to see whether or not they are allowed to send a message (Figure 1:4). If they are, their message is immediately queued for transmission (if existing) at the CAN controller (Figure 1:5). Otherwise, if not scheduled, the node has to wait for the next TM to see if it will be scheduled.

In order to terminate the EC, the M-Server is also sending a STOP message to itself (Figure 1:6). However, a small delay before sending STOP is required. This is needed to make sure that the STOP message is not sent before at least one of the other nodes have both processed the TM (in order to find out whether it is allowed to send or not), and (if it is allowed to send) enqueued its message in the corresponding CAN controller. The STOP message is of lowest priority possible, acting as an indicator for when all the nodes that were allowed to send a message within the active EC actually have sent their messages. If the M-Server receives the STOP message, all nodes that were allowed to send messages within the EC have already sent their messages. This since the STOP message, with its low priority, is the last message to be sent within the EC.

The M-Server is always reading all the messages that are sent on the CAN bus (Figure 1:7), i.e., the M-Server is polling the bus. This in order to update its server variables based on the actual traffic sent on the bus. Since servers are scheduled for message transmission even though they might not always have any messages to send, this has to be taken care of by updating the server parameters accordingly. There are different ways of updating the server-parameters in the case when the server did not send a message. Depending on how the server-parameters are updated the server will have different real-time characteristics [10, 11].

When the M-Server reads the STOP message (Figure 1:8), the EC is terminated, and the next EC is initiated based on the updated server-variables. Hence, the actual length of the EC might be less than T_{EC} due to slack [10, 11].

Using the Server-CAN concept, N-Servers can potentially join and leave the system arbitrary as long as the total utilisation by all the N-Servers in the system (bandwidth demand) is less or equal to the theoretical maximum. This joining and leaving is controlled by the admission protocol presented in Section 2.2.

2.2 Admission control

As an option to add flexibility to Server-CAN, an admission control mechanism can be used to dynamically add and remove N-Servers at run-time. In order for the admission control to work, each node in the system is required to have an N-Server that can be used to transmit protocol messages, e.g., an N-Server that is used for non real-time traffic.

Each N-Server has an N-Server ID i, and is allowed to send messages with specified identifiers m_i^j. Hence, each N-Server i is associated with a set of message identifiers M_i and will only allow transmission of messages with identifiers $m_i^j \in M_i$. Each node n is allocated a specific message identifier m_n used for protocol specific messages. These message identifiers are known and allocated to nodes at the initiation of the system. Hence, the M-Server will read these messages as protocol messages, automatically decoding the message.

When implementing the admission control, three requests are encoded into a single *request-message*. These request messages are used to add an N-Server, remove an N-Server, and update N-Server parameters. Moreover, three corresponding *reply-messages* are used to acknowledge the request. Hence, two types of messages are involved in the admission control mechanism, namely request- and reply-

messages. For example, the M-Server receives a request, performs some admission control, possibly updates its parameters and finally sends an appropriate reply-message.

2.3 Bandwidth sharing

Server-CAN allows the implementation of bandwidth sharing mechanisms. By having the M-Server monitoring the traffic it is possible to share bandwidth between N-Servers, even changing their N-Server parameters. Bandwidth sharing is done directly in the M-Server using bandwidth sharing algorithms, e.g., [2], and the changing of N-Server parameters (C_j and T_j) is done using the admission control presented in Section 2.2.

2.4 Mode changes

When the system performs a mode change, the M-Server changes its server parameters from one set to another. The mode change is triggered by the reception of a *mode-change message*. Upon the reception of this message the M-Server changes to the mode indicated in the message.

3 Fault tolerance in Server-CAN

Message omissions are recovered by retransmissions in general, provided by the built-in fault tolerance ability of native CAN protocol. Server-CAN builds on several mechanisms (both inherent and supplemented) in order to facilitate a fault-tolerant operation.

3.1 Fault model

The fault model used is that the network can suffer from four types of faults: (1) *Node software faults*, (2) *Node physical faults*, (3) *Channel permanent faults*, and (4) *Channel transient faults*. Each of these faults are discussed together with its impact on the communications when using Server-CAN, and how the Server-CAN protocol is affected. In some cases, feasible solutions for overcoming these faults are also discussed.

3.1.1 Node software faults

By using N-Servers as an application's interface between the application and the network, bad and unexpected behaviour of one application will not propagate to other applications on other nodes. Applications on the same node might suffer, and there might be a possible overload in the badly behaving application's N-Server. Also, propagation of errors from the badly behaving application could corrupt the N-Server.

N-Servers are implementing a software bus guardian preventing babbling idiots. Solutions to the babbling idiot problem have been presented for CAN [3] by the usage of extra hardware. Server-CAN is a software solution to this problem. However, compared with hardware bus-guardians, Server-CAN only solves babbling idiots caused by software faults. Also, Server-CAN does not present fault independence with respect to the rest of the node and is therefore vulnerable to propagation of errors from applications running on the same node.

Node software faults typically stems from software design errors. Here, it is assumed that all software, including the M-Server and the N-Server, are properly designed and subjected to validation tests and if possible, formally verified. The M-Server and the N-Server are not complex and therefore unlikely to exhibit design faults. However, other applications might suffer from software faults and then it is important to determine how these faults affect the Server-CAN communication.

3.1.2 Node physical faults

The M-Server is a single point of failure in the system, and the probability of a physical fault in the M-Server is not negligible. If the M-Server crashes there will be no transmission of the TM, i.e., no schedule is sent to the N-Servers in the systems and the system could in the worst-case be blocked. For correct operation, the M-Server must always recover. A solution to this is to have replicated M-Servers to prevent the scheduler of the network to disappear. This replication of the M-Server can be done similar to the handling of master-nodes in FTT-CAN [4, 13].

The replication of the M-Server is achieved by having one or more backup M-Servers (called B-Servers) in the system. These hot standby B-Servers keep monitoring the network as normal M-Servers, updating its server states, but they do not send TMs. Moreover, as N-Servers join or leave or change their properties, the B-Servers update their information as well. The consistency of this information must be guaranteed, and solutions similar to those existing for FTT-CAN [13] could be used. All B-Servers can verify that the TM contains the correct information. If not, a synchronization of M-Server data can be done. When there is no transmission of a TM by the M-Server, a B-Server takes on the role as the M-Server and transmits its current TM.

From a system point of view, the N-Server is not a single point of failure. However, it is a single point of failure from its user point of view.

3.1.3 Channel permanent faults

Channel permanent faults include link partitions, stuck-at-dominant, etc. The single link of a bus topology is a single point of failure. In topologies different from a bus, e.g. a star, a faulty link does not cause a global failure of the system. These types of faults are very important and usually addressed by bus replication [14]. Hence, in this paper it is assumed that the channel is free of permanent faults or is able to tolerate them by its own means.

3.1.4 Channel transient faults

These faults are due to Electro Magnetic Interference (EMI) and cause *message duplications* and *message omissions*,

and can be either *consistent* or *inconsistent* [15]. Message duplications cause a message to be transmitted twice at the cost of loss of bandwidth, and message omissions cause a message not to be transmitted at all.

Using the Server-CAN protocol, the M-Server is responsible for scheduling all N-Servers and sending the schedule to the N-Servers using the Trigger Message (TM) and terminating the Elementary Cycle (EC) using a STOP message. Hence, protocol specific messages sensitive to channel transient faults are the TM and the STOP messages, for which the implications of a channel fault during their transmission have to be investigated. Also, since these faults can happen in different combinations, detailed analysis of their impact on the fault-tolerant operation of CAN and Server-CAN is essential. Inconsistent message omissions may jeopardize the consistency among the M-server replicas. Next section is devoted to outline how these transient faults may affect M-server and B-servers consistency.

3.2 Consistency of M and B-Servers

Due to the complexity introduced by the Server-CAN protocol, the following mechanisms are subject to inconsistency among the M and B-Servers, and have to be analysed in detail:

• **Admission control** - The admission protocol involves message passing. Hence, it is vulnerable to channel transient faults. The admission protocol involves the transmission of request- and reply-messages. To ensure consistency, similar techniques as for FTT-CAN can be used [13].

• **Bandwidth sharing** - As bandwidth sharing mechanisms involve changing M-Server parameters, special consideration needs to be taken in order to avoid inconsistencies among the replicated M-Servers. Here, the same messages as when updating N-Server parameters are used. Hence, bandwidth sharing suffers from the same fault scenarios as admission control.

• **Mode changes** - Changing mode involves message passing and is therefore vulnerable to channel transient faults.

4 Summary and position of work

In order to use Server-CAN in safety-critical applications its fault tolerant operation has to be analysed. In this paper the fault model intended to be used has been presented. Multiple combinations of channel transient faults can occur and their impact on the Server-CAN protocol has to be analysed in detail. Protocol messages can be both omitted and duplicated in either a consistent or inconsistent way.

Babbling applications are tolerated by Server-CAN as long as they are caused by software faults. We also observe that some node physical faults may cause a global failure. In particular, the M-Server is a single point of failure so M-Server replication using B-Servers is required in order to tolerate physical faults. We observe that B-Servers require consistency and transient channel faults are an impairment to this requirement. Suitable mechanisms must be incorporated in order to guarantee consistency of M- and B-Servers under faults and we are currently analysing this issue in detail.

A possible advantage of Server-CAN is its inherent ability in improving the fault tolerance capabilities of the system, by having a flexible method for taking care of channel faults, as they can be scheduled as aperiodic messages without having the need to reserve bandwidth a priori. Together with replicated M-Servers, the Server-CAN approach could provide a fault-tolerant solution for applications where dependability is a primary requirement.

References

[1] L. Almeida, P. Pedreiras, and J. A. Fonseca. The FTT-CAN Protocol: Why and How. *IEEE Transaction on Industrial Electronics*, 49(6), December 2002.

[2] G. Bernat, I. Broster, and A. Burns. Rewriting history to exploit gain time. In *Proceedings of RTSS'04*, pages 328–335, Lisbon, Portugal, December 2004.

[3] I. Broster. *Flexibility in Dependable Real-time Communication*. PhD thesis, Dept. of Computer Science, August 2003.

[4] J. Ferreira, P. Pedreiras, L. Almeida, and J. Fonseca. Achieving fault tolerance in FTT-CAN. In *Proceedings of WFCS'02*, pages 125–132, Västerås, Sweden, August 2002.

[5] FlexRay Consortium. http://www.flexray.com/.

[6] ISO 11898. Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High-Speed Communication. ISO Standard-11898, Nov 1993.

[7] ISO 11989-4. Road Vehicles - Controller Area Network (CAN) - Part 4: Time-Triggered Communication. ISO Standard-11898-4, December 2000.

[8] LIN Consortium. LIN - Local Interconnect Network. http://-www.lin-subbus.org/.

[9] MOST Cooperation. MOST - Media Oriented Systems Transport. http://www.mostcooperation.com/.

[10] T. Nolte, M. Nolin, and H. Hansson. Real-Time Server-Based Communication for CAN. *IEEE Transactions on Industrial Informatics*, 1(3), August 2005.

[11] T. Nolte, M. Sjödin, and H. Hansson. Server-Based Scheduling of the CAN Bus. In *Proceedings of ETFA'03*, pages 169–176, Lisbon, Portugal, September 2003.

[12] Robert Bosch GmbH. BOSCH's Controller Area Network. http://www.can.bosch.com/.

[13] G. Rodríguez-Navas, J. Rigo, J. Proenza, J. Ferreira, L. Almeida, and J. A. Fonseca. Design and Modeling of a Protocol to Enforce Consistency among Replicated Masters in FTT-CAN. In *Proceedings of WFCS'04*, Sept. 2004.

[14] J. Rufino, P. Veríssimo, and G. Arroz. A Columbus' egg idea for CAN media redundancy. In *Digest of Papers, The 29th International Symposium on Fault-Tolerant Computing Systems*, pages 286–293, USA, June 1999. IEEE.

[15] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcast in CAN. In *Proceedings of FTCS-28. Munich (Germany)*, June 1998.