A Communication Protocol for Hard and Soft Real-Time Systems

Christer Eriksson, Henrik Thane* and Mikael Gustafsson
Department of Real-Time Computer Systems
Mälardalen University, Sweden
* The Royal Institute of Technology, Sweden
Email: {cen,mgn}@mdh.se, *henrikt@damek.kth.se

Abstract

The distributed real-time community is mainly divided into two camps: hard real-time and soft real-time; hybrid systems have been considered but not to any great extent. In this paper we propose a hybrid communication protocol, which forms a foundation for interconnection of nodes in a distributed hybrid real-time system. The protocol will enable both hard and soft real-time frames on a broadcast communication bus, yet still guarantees the hard real-time behaviour. The protocol will utilise the communication bandwidth more efficiently, which is relevant to cost-sensitive embedded applications.

1: Introduction

When implementing hard real-time services for safety-critical embedded systems the time-triggered approach is often preferred -- the input from the environment can only propagate into the system at pre-defined points in time, and output can only propagate from the system at pre-defined points in time [Kop91,Lev95]. Thus, the input/output state-space will shrink and the verification of the system will be significantly simpler. Precisely time-triggered actions are also essential for distributed control applications [Tör96].

Many applications have however, not solely hard real-time or soft real-time requirements, but a mixture of both kinds. There are instants when a function with soft real-time requirements, in an otherwise hard real-time environment, can be more efficiently implemented in a soft real-time manner than in a hard real-time fashion. For example, may long time planning in a mobile robot application be implemented as a soft real-time task [Nil95].

When applications are cost-sensitive, the hard real-time requirement to accommodate for the worst case execution time, yielding a low CPU utilisation, may seem like a waste. There is thus a need for a hybrid, enabling both hard real-time and soft real-time requirements to be implemented in a satisfactory way.

Some work has been done on integrating hard and soft real-time on a single node [Ham92,Rub95]. One hybrid handles the hard real-time services via the time-triggered approach and the soft real-time services via the event-triggered approach. The time-triggered approach uses a priori constructed schedules, which are used as dispatch-tables during run-time. The soft real-time approach uses on-line scheduling. To generalise this hybrid into a distributed real-time system the communication system is required to function in the same manner.

Outline

In section 2 the basic principles will be presented. In section 3 we will cover the state of the art. In section 4 we will describe the proposed protocol and the clock synchronisation. In section 5 we will discuss redundancy issues and in section 6 we will discuss implementation issues. Finally, we will make some conclusions.

2: Basic principles

In this paper we will pursue this idea of a hybrid communication system. We will present a protocol, enabling both hard and soft real-time frames on a broadcast communication bus, yet still guarantees the hard real-time requisites. We will also cover the subject of clock synchronisation but not the subject of membership. The protocol will utilise the communication bandwidth more efficiently, which is relevant to cost-sensitive embedded applications. The actual implementation of the protocol interconnects the nodes via CAN.

The Controller Area Network (CAN) is developed by Intel and Bosch [CAN92], and is a broadcast communication bus used for connecting nodes and devices in embedded systems. The CAN bus uses a CSMA/CA protocol, solving bus contention by arbitrating between frames' unique identifiers/priorities. The CAN bus has become very popular in the embedded control systems community; not surprisingly since it is a standard, having an advanced error correction design and a low price.

Traditional *Time Division Multiple Access* (TDMA) protocols [Kop94a], statically divide a computer network's capacity into a number of bus-slots. Each node in a system is allocated *one* bus-slot, with a corresponding right to transmit *one* frame. This approach poorly utilises the transmission media -- the bus. Unused bus-slots cannot be reused by other nodes. The worst case response-time is one *round*, i.e., the accumulated time-period, of having let all designated nodes have a chance to transmit one frame.

Our approach is in essence a TDMA protocol with a CSMA/CA protocol interior to every bus-slot. This approach has the benefit of allowing *several nodes* to send *several frames* in one bus-slot; yielding a short response-time and allows soft real-time frames to be sent when spare frames exist, i.e., when the hard real-time application has not used all frames in a bus-slot.

The traditional TDMA protocol can be seen as special case of this protocol, i.e., only one node is allowed to send one message in one bus-slot.

This protocol requires that a global time-base is established and that all clocks are synchronised with a known precision. A global time base is imperative since the protocol is synchronous (TDMA) and that the kernels on each node are all subjects to a global time-triggered schedule. Our clock synchronisation protocol uses a distributed fault-tolerant scheme. Using special characteristics inherent to all broadcast buses, i.e., that all nodes receive a frame at the same time, and by using a daisy-chain concept [Lön95] we can provide a fault-tolerant clock-synchronisation in every bus-slot. The clock synchronisation algorithm will be covered in section 4.2.

3: State of The Art - Communication principles and clock synchronisation

3.1: TTP - Time Triggered Protocol

TTP is a TDMA protocol. If a node has no data to send in its allotted bus-slot, an empty frame is transmitted instead. These periodic emissions contain acknowledgement information and is viewed as a life-sign by the membership service [Kop94a].

Clock synchronisation is accomplished by knowing all frames' arrival time a priori. The difference between the actual arrival time and the a priori scheduled arrival time is a measure by of how much the sender's and the receivers' clocks differ. The set of collected information is then sufficient to perform a fault-tolerant clock synchronisation[Kop87,Kop94a]. This is done at every re-synchronisation point, likely once every round. TTP uses a *Fault-tolerant average algorithm* (FTA) [Lam85] to compute a global time. It discards the m slowest and m fastest readings and average the rest, in order to cope with m faulty readings.

3.2: DACAPO

DACAPO uses, as TTP, a TDMA scheme and utilises the time difference between a received message and the pre-destined arrival time as a measure by how much two clocks differ. However when TTP uses a Fault-tolerant average algorithm [Lam85] to compute the global time, DACAPO uses a *Daisy-Chain algorithm* (DC). The principle behind this algorithm is that all clocks adjust to the time of the currently sending node [Lön95]. The obvious benefit, is that all clocks are synchronised every time a frame is transmitted and not once every round, as is the case with TTP.

Fault-tolerance is realised by having a *reception window* centred around every sample point. This window is as wide as the expected maximum clock skew. The reception window filters through only the frames arriving inside the window and thus ignores probably faulty clocks.

3.3: VIA Basement

The protocol uses a TDMA scheme too, but allow in contrast to TTP and DACAPO, a bus-slot to contain several frames and allow bus-slots to vary in length [Ahl95a, Ahl95b]. The basic idea in Ahlgren's work is very similar to our work (mixing hard and soft real-time frames on the bus) but does not consider fault-tolerance issues, such as membership, redundancy and fault tolerant clock synchronisation. Clock synchronisation is accomplished via a master-slave concept. Ahlgren's work has neither been implemented to the same extent as our work.

3.4: The Real-Time Priority Bus

This protocol differs considerably from the previously described protocols. When the other protocols are in essence time-triggered this protocol is event-triggered. The *real-time priority bus* concept uses an extension of fixed priority pre-emptive scheduling, and addresses scheduling of frames on shared broadcast buses [Tin95a]. The approach requires the communication bus to support priorities.

The protocol works as follows: each node has a queue of out-going frames, which are sorted by priority. Bus contention is solved by arbitrating between frames' priorities. The assignment of the frames' priorities is made off-line. If tasks residing on different nodes has precedence relations with an associated message the clocks of the nodes must be synchronised. A general analysis has been derived for this protocol and a more specific analysis has been developed for the CAN communication protocol [Tin95a, Tin95b].

Using an event-triggered concept imposes serious restraints on how to implement redundancy [Kop94b]. It also limits the confidence gained by testing since the state-space, which has to be covered is far greater than when using a time-triggered concept [Lev95].

4: Protocol description

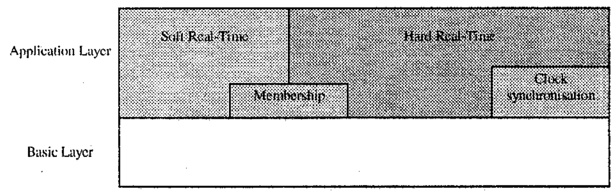

Figure 1: The protocol hierarchy.

The proposed protocol assumes that a broadcast bus is used, where the transactions are atomic, i.e., that every node or none receives a frame.

The protocol consists of two layers: the basic layer and the application layer (figure 1). We will in this paper only cover the basic layer and clock synchronisation. The basic layer is in essence the data-link layer of the OSI stack, merely handling transmission and reception of frames over the network. The application layer handle time-constrained and non time-constrained messages of greater complexity than one frame. For example, may the soft real-time part of the application layer provide services for a TCP/IP implementation.

4.1: Basic layer

The basic layer handles two types of frames, *hard real-time frames* (HRTF) and *soft real-time frames* (SRTF). The modes of transmission range from broadcast to unicast, depending on if the receivers are configured to sift through or not sift through frames.

HRTFs have timing requirements, which must be met. To satisfy these timing constraints, HRTFs are scheduled pre run-time using an off-line scheduler. SRTFs are scheduled on-line, using for example, the real-time priority bus concept [Tin95a, Tin95b] and information provided by the off-line scheduler about the spare-capacity in bus-slots.

4.1.1: Description of the basic protocol

In each bus-slot a fixed number of n_i frames can be sent, where i is the slot index. A frame is an atomic data-unit that can be transmitted on the bus. The bus-slots can have varying lengths, depending on the application requirements. The duration of a bus-slot is denoted t_i. The application may allocate hard real-time frames (HRTFs) using the entire or a subset of all the frames in a bus-slot. These HRTFs are transmitted first. The frames sent in a bus-slot can originate from several different nodes. When all globally scheduled HRTFs have been successfully transmitted, soft real-time frames (SRTF) can be transmitted. Since a broadcast bus with atomic transactions must be used, we can be certain that a consensus exists regarding if all frames have been transmitted or not.

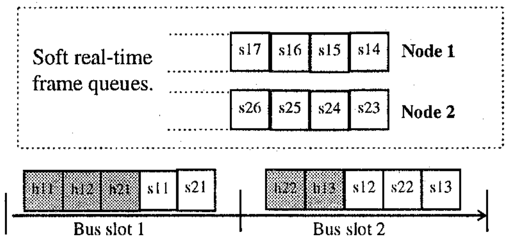

Figure 2: Frames in slot 1, 2 and pending SRTFs on each node.

Example 1: Assume a system that consists of two nodes and two bus-slots. Each bus-slot contains five frames.

- In bus-slot 1: node 1 sends HRTFs h11 and h12; node 2 sends a HRTF h21.
- In bus-slot 2: node 2 sends HRTF h22; node 1 sends a HRTF h13.

There are two frames left for SRTFs in bus-slot 1, and three left in bus-slot 2. The SRTFs are, as mentioned previously, scheduled on-line and picked from a queue on each node. SRTFs transmitted and pending are illustrated in figure 2.

4.1.1.1: The bus-slots are divided into zones

Every bus-slot *i* of length t_i is divided into 4 zones. The reasons for having this division, is partially due to the precision of the global time base, partially due to termination of ongoing transmissions, and due to the need to establish a consensus in the system before changing bus-slots or changing system modes.

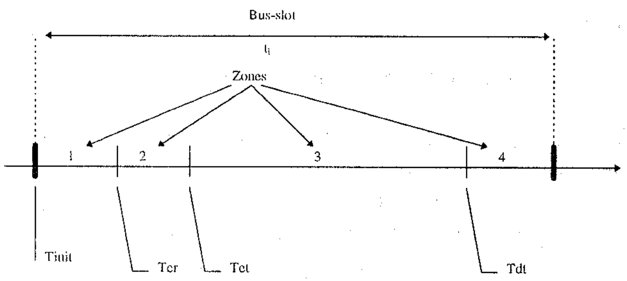

Figure 3: The four zones of a bus-slot.

Zone 1

At instant *Tinit* does a zone change take place, from zone 4 to zone 1.

No traffic on the bus is allowed during zone 1, reception and transmission of frames are disabled. Pending HRTFs from the previous slot are downloaded to the communication controller. A HRTF frame waiting to be transmitted must be available for transmission during the previous bus-slot, and will be available to the receiver at the start of the following bus-slot. It will take a minimum of one bus-slot (t_i) and a maximum of two bus-slots(t_i+t_{i+1}), from the production of the data until the delivery of the data.

Zone 2

At instant *Ter* zone 1 changes to zone 2. Reception of HRTFs is permitted, if they are scheduled, otherwise reception of SRTFs is allowed. Due to the precision of the global time-base, can the transmissions of frames not start at the same time as the reception of frames. The duration of zone 2 must be at least as long as the maximum clock skew,δ , between any two clocks in the system. If this is not accommodated for, frames could be lost if a receiver is lagging behind.

Zone 3

At instant *Tet* zone 2 changes to zone 3 and transmission of frames is allowed. Any time during zone 3 can transmission and reception of SRTFs be enabled if all HRTFs are sent and received.

Zone 4

At instant *Tdt* a zone change takes place from zone 3 to zone 4, and transmissions of HRTFs and SRTFs are disabled. Due to the precision of the global time base, can reception of frames not terminate at the same time as the disabling of transmissions. Neither can we disable reception until the last sent frame has propagated through the network. This sums up to a duration of zone 4 to at least δ +*Ft* (where Ft is the transmission time of one frame). If this is not accommodated for, frames could be lost due to a lagging transmitter.

4.1.1.2: Error services

A vital feature in any communication system is to provide error notification to the application. Our system provide services for error notification when:

- an omission failure is discovered, i.e., all HRTFs have not been successfully transmitted or received in a bus-slot. Causes may be transient faults on the bus, a fail-silent node or an awry clock synchronisation.
- a frame is received in zone 1. A cause may be a faulty clock.
- a SRTF is received when not all HRTFs have been received or transmitted. A cause may be a faulty agreement of the members in the system (membership).

4.2: Clock Synchronisation Protocol

All clock synchronisation protocols must: (1) gather information about the clock skews between nodes in a system, (2) calculate a global time-base using the gathered information, and (3) adjust the local clocks according to the global time-base. If the clock synchronisation protocol is fault-tolerant too, it must also discern which clock skews are faulty and discard the faulty values.

Our clock synchronisation protocol is a generalisation of the protocol proposed by Gergeleit and Streich [Ger95]. Their clock synchronisation scheme uses a master-slave concept, while our protocol uses a rotating master scheme. This eliminates a single point of failure.

The protocol is founded on a few basic assumptions:

- A broadcast bus must be used. Every node in a system receives a frame at the same time with a known tolerance, τ.
- Every successful transmission and reception of frames are acknowledged. The delay between transmission and reception acknowledgement must be fixed.

The protocol works as follows:

1. An initiator node, Y_n sends a frame in a bus-slot n. The master in bus-slot n, M_n time-stamps the arrival-time of the frame, t_m. Every slave, S time-stamps the arrival-time, t_s of the frame.
2. The master M_n in slot n sends its time-stamp t_m in slot n+1. When the slaves receive M_n's time-stamp, t_m the slaves calculate the clock skew $\delta = (t_m - t_s)$ and adjust their clocks according to δ. The Master, M_n's "time-stamp frame" is now a new initiator Y_{n+1}.
3. The master sets $\delta=0$ and does therefore not adjust its clock.

The set of masters could either be a subset or the entire set of nodes in the system. The set of masters must be chosen in relation to the specified failure hypothesis. Fault-tolerance is achieved by only using clock skews interior to a specified tolerance, δ_{max}.

5: Redundancy issues

A possibility for a time triggered protocol to handle omission failures is to use redundancy. Redundant HRTFs could be replicated according to the following principles:

- replicate a frame k times in a bus-slot to handle k-1 failures.
- replicate a frame in k bus-slots to handle k-1 failures
- replicate a frame on k buses to handle k-1 failures.

The actual replication of HRTFs should be handled by an off-line scheduler.

6: Implementation

The implementation of the protocol is based on the Intel 82527 CAN controller. The CAN protocol fulfils the requirements listed in section 4 except for the requirement that the controller must be deterministic. One of the major drawbacks with current CAN implementations, is that the application software has no or limited control of the automatic re-transmissions of frames. It would have been useful if the controller had provided services for enabling and disabling re-transmissions, or provided notification by interrupt if a re-transmission was commencing, so that further re-transmissions could be terminated by the controlling software.

Another problem regarding CAN, however academic, is that a receiving node could have a Byzantine behaviour [Lam85]. For example if a controller-interface is flawed, a receiver could terminate every sent message by sending an active error frame. A nasty scenario, is when a controller terminates every other message by sending an error frame, which results in it never becoming error passive. Another malicious behaviour might be when a faulty receiver terminates 127 frames before it becomes error passive.

7: Conclusion

In this paper we have pursued the idea of a hybrid communication system. We have presented a protocol, enabling both hard and soft real-time frames on a broadcast communication bus, which guarantees the hard real-time requisites.

The approach was in essence a TDMA protocol with a CSMA/CA protocol interior to every bus-slot. This approach has the benefit of allowing several nodes to send several frames in one bus-slot, both hard real-time frames and soft real-time frames. Comparing the proposed protocol with a traditional TDMA scheme, the protocol has a better utilisation of the media by allowing soft real-time frames on the bus. Comparing the proposed protocol with an event triggered concept it is more deterministic. This protocol is very flexible, it can either be configured as a strictly TDMA kind of protocol, strictly as an event-triggered kind of protocol or anything in between.

8: Acknowledgement

This work has been supported by the Swedish Technical Board (NUTEK) and the Research Council of Västmanland.

9: References

[Ahl95a] B. Ahlgren. VIA BASEMENT DRTS Requirements document: Communication Services. ProVIA-93303, Version 2, Stockholm February, 1995.

[Ahl95b] B. Ahlgren. VIA BASEMENT DRTS Requirements document: Communication Protocol Specification.. ProVIA-93304, Version 2, Stockholm February, 1995.

[CAN92] Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High Speed Communication, ISO DIS 11898, February 1992.

[Ger95] M. Gergeleit, H. Streich. Synchronising High-Resolution Clocks via the CAN-Bus. Second International CAN Conference. Heathrow, England, October 1995.

[Ham92] D.K.Hammer and O.S. van Roosmalen. An Object-oriented Model for the construction of dependable distributed systems, proc. 2nd International workshop on object orientation in operating systems. Paris 1992.

[Kop87] H. Kopetz. Clock Synchronisation in Distributed Real-Time Systems, IEEE Trans. Computers, Aug. 1987.

[Kop91] H. Kopetz. Event-Triggered versus Time-Triggered Real-Time Systems. Lecture Notes in Computer Science, vol. 563 Springer Verlag, Berlin, 1991.

[Kop94a] H. Kopetz and H Grünsteidl. TTP - A Protocol for Fault-Tolerant Real-Time Systems. IEEE Computer, January, 94.

[Kop94b] H. Kopetz. A Communication Infra Structure for a Fault-Tolerant Distributed Real-Time System. Proceedings 1994 IFAC workshop on Distributed Computer Control Systems, Toledo, Spain, September 1994.

[Lam85] L Lamport et.al, Synchronising clocks in the presence of faults, journal of the ACM, Vol. 32, No. 1, Jan 1985.

[Lev95] N. Leveson. Safeware – System Safety and Computers. Addison Wesley, 1995.

[Lön95] Lönn H et. al, Synchronisation is Safety-Critical Distributed Control Systems for Vehicle Dynamics", Tech Report, Chalmers, 1995.

[Nil95] M. Nilsson, A New Approach to Extreme Fault Tolerance: "Conscious" Real-time Control. In proceedings of SNART'95, Chalmers Institute of Technology, August 1995.

[Rub95] Rubus OS, Real-Time Operating System, Tutorial. Arcticus Systems AB Datavägen 9A, 175 62 Järfälla, Sweden 1995.

[Tin95a] K. W. Tindell et. al. Analysing Real-Time Communications: Controller Area Network (CAN). Proceedings Real-Time Systems Symposium, Puerto Rico, December 1994.

[Tin95b] K. W. Tindell et. al. Analysis of Hard Real-Time Communications. Journal of Real-Time Systems, vol. 9, no. 2, September 1995.

[Tör96] M Törngren. Fundamentals of Implementing Real-Time Control Applications in Distributed Computer Systems. To appear in the Journal of Real-Time Systems, 1996.