AN EVENT-TRIGGERED COMMUNICATION PROTOCOL FOR INTELLIGENT REAL-TIME CONTROL

Jason J. Scarlett and Robert W. Brennan

Department of Mechanical and Manufacturing Engineering, University of Calgary, 2500 University Dr. N.W., Calgary, Alberta, CANADA, T2N 1N4

Abstract: In this paper we propose an event-triggered communication protocol with dynamic priority setting for high-integrity systems. The dynamic priority scheduling involves two components: a component based on node transmission history, and a static node identifier. In order to investigate the characteristics of the proposed protocol, the worst-case performance is analysed and simulation experiments are conducted. The results show that an upper bound on the message delay time can be established with this approach. *Copyright © 2005 IFAC*

Keywords: communication protocols, distributed control, intelligent control.

1. INTRODUCTION

Computers and computer networks have become an integral part of our daily lives. The most famous network is of course the Internet. Its development dates back to 1969 when the ARPAnet was first operational. Today the Internet remains the most easily identifiable computer network, but other networks are springing up all around us. These smaller networks are evident as embedded systems, and are playing an increasing but hidden role. Consumer products such as printers, answering machines, ATMs, refrigerators, thermostats, and even wristwatches are all equipped with networking capability to increase their functionality and intelligence (Moschovitis, 1999). Safety concerns arise when these embedded systems are used in safety-critical applications such as automobiles, trains, aircraft, medical equipment, and industrial machinery.

The suitability of a specific protocol for safety-critical applications must consider a wide range of issues. Many existing protocols are already being used in safety-critical applications. In this paper, we begin with a brief review of the main safety-critical protocols (TTCAN, FTT-CAN, TTP/C, Byteflight, and FlexRay) along with a summary of some of the safety techniques implemented in these protocols. Common issues in safety such as redundancy, data validation, fault isolation, and timing aspects are found in existing protocols.

In Section 3, an analogy is given outlining common assumptions that are made about the deterministic nature of event-based and time-triggered systems, and a new protocol is suggested that challenges some of the commonly accepted ideas regarding determinism and event-based protocols.

Next, we investigate the characteristics of this new protocol in Section 4. We begin with an analytical investigation into the worst-case performance of the protocol then test our results with a set of discrete-event simulation experiments. This paper concludes with an overview of our work on exploring the potential for this new protocol.

2. REAL-TIME COMMUNICATION PROTOCOLS

A variety of communication protocols are currently being used in safety-critical applications that fall into two main categories: event-triggered and time-triggered protocols. In this section, we briefly summarise the main protocols used in industry.

Currently, the main communication protocols used for real-time embedded systems are TTCAN (Marsh, 2003), FTT-CAN (Ferreira et al., 2001), TTP/C (Marsh, 2003), Byteflight (Kopetz, 2001), and FlexRay (Kopetz, 2001).

Closely related to each of these recent protocols is the Control Area Network (CAN) protocol, developed by Bosch (1991). CAN is an event-triggered protocol that does not guarantee message delivery times. Driven by the need for a deterministic protocol for use in safety-critical systems, Bosch developed Time Triggered CAN (TTCAN) (Marsh, 2003). TTCAN extends CAN by implementing a dual scheduling window system. In this scheme, time windows for both periodic and spontaneous messages are created allowing both time-triggered and event-triggered messages to coexist on the same network (Shaheen et al., 2003).

FTTCAN was proposed by Joaquim Ferreira, Paulo Pedreiras, and Luís Almeida from the Universidade de Aveiro, Portugal in 2002 (Ferreira et al., 2001; Almeida et al., 2002). FTTCAN extends the dual scheduling window to a new level. The newly defined time windows are not only able to handle both time-triggered and event-triggered messages, but are dynamically resizable. This allows the protocol to adapt to the present traffic conditions on the bus.

TTP/C uses TDMA (Time Division Multiple Access) scheme for bus access (Marsh, 2003). Each node is assigned a window during which they have exclusive broadcast rights. Currently, this protocol does not define support for event-triggered messages. In order to support event-triggered messaging, an additional protocol such as CAN must be implemented within TTP/C (Kopetz, 2001).

The Byteflight protocol was developed by BMW together with several semiconductor companies for safety-critical applications in automotive vehicles (Kopetz, 2001). This protocol is not a true time-triggered protocol because it does not rely on global timing synchronization (Shaheen et al., 2003). Instead, Byteflight provides access to the bus in a FTDMA (flexible time division multiple access) scheme.

BMW, DaimlerChrysler, Motorola, Philips, GM and Bosch are currently working together on yet another protocol aimed at the automotive industry (Kopetz, 2001). Like TTCAN and FTT-CAN, FlexRay uses the dual time window approach. This protocol however uses two different bus access methods for the different windows. Asynchronous messages are handled by the FTDMA Byteflight protocol and the synchronous messages are handled by a TDMA scheme (Shaheen, 2003).

3. EVENT-TRIGGERED AND TIME-TRIGGERED COMMUNICATION PROTOCOLS

Much of the discussion about choosing a protocol begins with the assumption that time-triggered protocols are the only ones suited to safety-critical applications. This assumption is based on the belief that time-triggered schemes are deterministic (higher degree of predictability) and event-triggered schemes are not (Claesson et al., 2003). The following comparison will illustrate why this assumption may not be the right one to take when approaching safety-critical systems.

One of the first things to clarify is determinism. Determinism can be defined as:

- The ability for a system to respond with a consistent, predictable time delay between input and response (Taylor, 2003).
- A system whose time evolution can be predicted exactly.
- A system in which the output can be predicted with 100 percent certainty.

These definitions have three things in common. They all deal with a system, they all attempt to predict the outcome, and they all do so with a degree of certainty. For safety-critical systems message latency must be predictable.

In time-triggered systems deterministic latency is achieved by avoiding collisions all together. In doing so, time-triggered systems are able to guarantee the transmission of a message at a given point in time. This ability to guarantee or predict message delivery makes time-triggered systems suitable for safety critical applications.

In the case of event-triggered systems, the latency depends on the frequency of collisions, and on the arbitration technique used to resolve these collisions. Some argue that it is not possible to predict the latency of event-triggered systems because of the uncertainties involved with arbitration. Another way to state this is that in an event-triggered system, the message latency depends on the volume of network traffic. As network traffic increases, the latency also increases. This variation introduces a sense of uncertainty that some claim cannot be tolerated in a safety-critical environment. On the other hand, a purely time-triggered system will always be able to guarantee message latency independent of the traffic level (within capacity limits), bringing a sense of predictability to the network. This seems quite logical, but the following example illustrates why this should not be the sole basis of describing a safety critical system.

Compare the transportation option available to a commuter in large city. Let us assume that the commuter has two options available for the daily commute; a car or public rail transport. During light traffic times of the day, the commuter is likely to opt for the car because of the shorter commute time (all other things being equal). During heavy traffic times of the day, the commuter is likely to opt for the rail transport system because of the shorter commute time. This is quite similar to the choice between event-triggered (car) and time-triggered (rail) systems. For periods with low volume of traffic an event-triggered system will outperform a time-triggered system. During the periods of high volume traffic, the event-triggered (car) system will result in potentially greater delay (commute) time than the time-triggered system. At this point many have concluded (prematurely) that the time-triggered approach is best for safety-critical applications because its performance is more predictable.

We ask then, if the commuter has a medical emergency (safety-critical) which transportation system will they rely on? In reality they would rely on a special type of event-triggered transport that could be describes as having a higher priority while on the roadways (i.e. an ambulance). This ensures a quick transport time, and improves the degree of uncertainty in the transport time even during periods of high volume.

Traditionally, the uncertainty in message delivery makes time-triggered the preferred option. But clearly, in the analogy, introducing a priority to an event-triggered system may be able to address the issue of uncertainty. Current safety-critical communication protocols do not include an event-triggered protocol that employs dynamic message priorities to deterministically describe the messaging delays.

4. AN EVENT-TRIGGERED PROTOCOL WITH DYNAMIC PRIORITIES

Event-triggered protocols have the advantage of potentially fast response times. These response times are however linked to the level of traffic on the network. What we seek is a method to decouple these two characteristics from each other. Additionally, we understand that under maximum loading the time-triggered method appears to be the optimal solution since it maintains deterministic message delays. Note that we have not considered the effects of exceeding network capacity.

The protocol suggested here is meant to illustrate that the traditional view, that an event-triggered system cannot be deterministic, is false. This false conclusion has in turn lead to the false conclusion that event-triggered systems should not be used for safety-critical systems.

The basic idea is to constrain the system in such a way that every node is guaranteed an opportunity to transmit at least one message in a given period of time. This period of time is essentially equal to the time-cycle defined in time-triggered protocols. In doing this, the protocol is able to guarantee performance (i.e., maximum message delay) equal to a similar time-triggered protocol.

4.1 Fixed priority scheduling.

Collisions in an event-triggered protocol such as CAN are dealt with using bit-wise arbitration. Essentially this allows the message with the highest priority to continue it's transmission without delay during a collision. All other colliding messages must try again at a later time. With CAN, the message priority is assigned based on the originating node's priority. These node priorities are determined pre-run-time and are static. This use of static message (node) priorities prevents delay times from being predicted deterministically.

For example, if two messages collide, the losing message must wait until the winning message is finished it's transmission before trying to retransmit. When the winning message is finished, the losing message may try again only to find that it is competing with another message of higher priority. It is clear to see that the lower priority message has a distinct disadvantage. During times of high traffic volume, the lower priority message(s) will see increased average delay times greater that that of higher priority messages. These delay times are deterministically unbounded.

One very interesting point to note is that the node with the highest priority will always win arbitration. This means when the highest priority node wishes to transmit a message, it will have to wait, only until the current message transmission is complete. In this way the maximum message delay is equal to that of the longest message and is deterministic.

4.2 Dynamic priority scheduling.

Dynamic priority scheduling allows the maximum delay time to be deterministically predicted. In order to achieve this, the priority setting scheme must guarantee that each node will not have to wait longer than one cycle to broadcast. This maximum delay time is equivalent to the delay time a time-triggered protocol can guarantee.

The dynamic priority scheduling proposed here involves two components. The first component is a calculation based on the message history within the node. The second component is a unique static identifier like the one used in the fixed priority scheduling method. Together these two components make up a dynamic priority code that is computed by each node.

The calculated component is used to increase a node's priority the longer it waits and is based on the time since the node last sent a message. An important feature of this is that no clock synchronization is required between the nodes as is the case with time-triggered systems.

The static component is assigned pre run-time as a unique identifier for each node. It is required when two messages collide that have the same time priority. This happens only when two nodes that have not transmitted for greater than the cycle time attempt to transmit at the same time. In this case both nodes would have the highest time priority, and the static component would then be used to settle the arbitration. In the next section, we look at this protocol in more detail and determine its worst-case performance.

4.3 The worst-case scenario.

For simplicity constraints are made on the system's messaging abilities. All messages have the same *worst-case transmission time*:

$$C_n = C \;\forall\; N \tag{1}$$

where N is the *total number of nodes* in the system.

The *system period*, T_{system}, is defined as the total time it takes if each node was to transmit once:

$$T_{system} = N \cdot C \tag{2}$$

This definition is derived from a time-triggered system where each node is given one time slot during a transmission cycle.

Next, we define the *node idle time*, L_n, which is calculated based on the difference between the *current time*, T_{now}, and the *time that a node's last message transmission ended*, TL_n.

$$L_n = T_{now} - TL_n \tag{3}$$

As noted in the previous section, a node's *dynamic priority*, DP_n, is based on the node message history and a static identifier. The first component, the *time priority*, TP_n, represents the dynamic portion of DP_n. By definition, the TP_n is equal to N immediately after a node transmittes a message. The more time that passes, the higher the *time priority* becomes until it reaches a value of 1. The second component is the static *node priority*, NP_n.

$$TP_n = \text{Max}\ \{\text{RoundUp}\ [(T_{system} - L_n)/C], 1\} \tag{4}$$
$$NP_n = n \tag{5}$$
$$DP_n = \text{Concatenate}\ (TP_n, NP_n) \tag{6}$$

In order to determine the maximum delay time, we consider the worst potential delay time of a message. For the worst-case scenario we choose the node that has been assigned the lowest node priority of N. In choosing this node, we ensure that this node will be the last to transmit during a tie (i.e., of two nodes with equivalent time priorities, the node with a node priority of N is guaranteed to lose). We also choose the node to have the lowest time priority at the current time. This is equivalent to saying that the node has just finished transmitting a message.

To ensure the maximum amount of delay to our node, we choose the network loading to be as high as possible. To do this, two things are required. Firstly, all other nodes must currently have a message waiting. And secondly, all of the other nodes have a time priority of 1. This results in a network where, all the nodes with a node priority between 1 and N-1 have a message waiting and a time priority of 1.

Since the worst-case node has the lowest time priority, it follows that all of the other nodes will win arbitration, and continue to win arbitration even when the worst case node has a time priority of 1 (remember that the worst case node has a node priority of N and hence it loses all ties). Since there are N-1 nodes with a higher time priority than the worst case node, the worst case node will not be allowed to transmit during the first (N-1)C time units.

If the worst case scenario described above occurs at an initial time of 0, the current time is after N-1 nodes have transmitted their messages, then the first transmission completes after C time units and the idle time of the first node to send a message given by (3) is:

$$L_1 = T_{now} - TL_1 = (N-1)C - C = (N-2)C$$

Since all of the remaining nodes transmitted after this first node, their idle time must be less than the first transmitting node's idle time. Therefore, for all nodes except the worst case node,

$$L_{1,\ldots N-1} \leq L_1$$
$$L_{1,\ldots N-1} \leq (N-2)C$$

Calculating the time priority using (4) and (2) for these nodes gives:

$$TP_{1,\ldots N-1} = \text{Max}\ \{\text{RoundUp}\ [(T_{system} - L_{1,\ldots N-1})/C], 1\}$$
$$TP_{1,\ldots N-1} \geq \text{Max}\ \{\text{RoundUp}\ [(T_{system} - (N-2)C)/C], 1\}$$
$$TP_{1,\ldots N-1} \geq \text{Max}\ \{\text{RoundUp}\ [T_{system}/C - N + 2], 1\}$$
$$TP_{1,\ldots N-1} \geq \text{Max}\ \{\text{RoundUp}\ [N \cdot C/C - N + 2], 1\}$$
$$TP_{1,\ldots N-1} \geq \text{Max}\ \{\text{RoundUp}\ [2], 1\}$$
$$TP_{1,\ldots N-1} \geq 2$$

Calculating the idle time using (3) and time priority using (4) and (2) for the 'worst case node' gives:

$$L_W = T_{now} - TL_W = (N-1)C - 0 = (N-1)C$$

$$TP_W = \text{Max}\ \{\text{RoundUp}\ [(T_{system} - L_W)/C], 1\}$$
$$TP_W = \text{Max}\ \{\text{RoundUp}\ [(T_{system} - (N-1)C)/C], 1\}$$
$$TP_W = \text{Max}\ \{\text{RoundUp}\ [T_{system}/C - N + 1], 1\}$$
$$TP_W = \text{Max}\ \{\text{RoundUp}\ [N \cdot C/C - N + 1], 1\}$$
$$TP_W = \text{Max}\ \{\text{RoundUp}\ [1], 1\}$$
$$TP_W = 1$$

Since TP_W is higher than $TP_{1,\ldots N-1}$ at $T = (N - 1)C$, the 'worst case node' is guaranteed to transmit at $N \cdot C$ time units (i.e., there is a maximum period of $(N - 1) \cdot C$ between messages).

For a system with 5 nodes, Table 1 depicts the scheduling of messages in the worst-case scenario. At time equal to 0, the worst-case node (node 5) has just completed transmitting a message. Immediately following, all 5 nodes are waiting to send a message. The first four nodes have a time priority of 1 while the worst-case node has a time priority of 5. This results in the first four nodes being scheduled in the next four time slots followed by the worst-case node in the fifth time slot. The total time delay between the beginning of successive messages from the worst-case node is shown as 5*C*.

Table 1 Worst case scheduling for a 5 node system.

Node	Dynamic Message Priority					
1	-	**11**	51	41	31	21
2	-	12	**12**	52	42	32
3	-	13	13	**13**	53	43
4	-	14	14	14	**14**	54
5	**15**	55	45	35	25	**15**
Broadcasting Node	5	1	2	3	4	5
Transmsn. End Time	0	C	2C	3C	4C	5C

4.4 Experiments.

For simulation purposes the system is modelled consisting of five nodes with equal exponentially distributed mean interarrival times. Message length is constant for all nodes and is set to one time unit. Scheduling of the bus is handled by one of three methods: time-triggered (TTCAN), event-triggered (CAN), and dynamic priority event-triggered scheduling.

Initial simulation results for time-triggered and event-triggered scheduling methods are consistent with the expected outcome. In each case the nature of the simulation delay time is consistent with the commonly expected nature of the scheduling method.

Figure 1 shows the simulation results for the time-triggered method. Specifically, the maximum delay time experienced by any message from any node is five time units. It can also bee seen that each node performs equally.

In contrast, static event-triggered scheduling (Figure 2) shows that each node does not perform equally well. Nodes with higher static priorities perform better. In fact, the node with the highest static priority demonstrates its dominance with a maximum delay time that never exceeds one time unit (seen as the single nearly horizontal line). The non-deterministic nature is also evident by the increasing delay time as message interarrival times decrease. For these simulations, the maximum delay time for the lowest priority node was 170 time units.

As illustrated in Figure 3, the dynamic priority scheduling method achieves its primary goal. The figure shows that the maximum delay time never exceeds five time units, and hence can be described deterministically. Unlike the static priority method, the delay times for the dynamic priority method appear to be 'clipped' as the message interarrival time decrease. This is precisely the behaviour we desire. Two other benefits are also seen. Firstly, the average delay times are now more consistent between the different nodes meaning that no node is 'outperforming' any of the other nodes. Secondly, the average delay time is better than those shown by the time-triggered method.

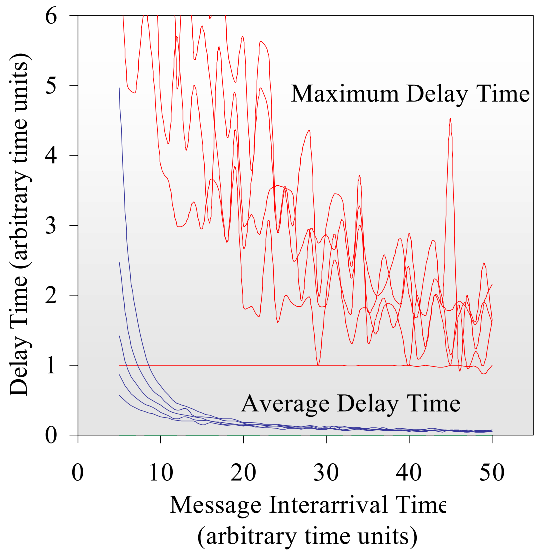

Fig. 1. Time-triggered performance.

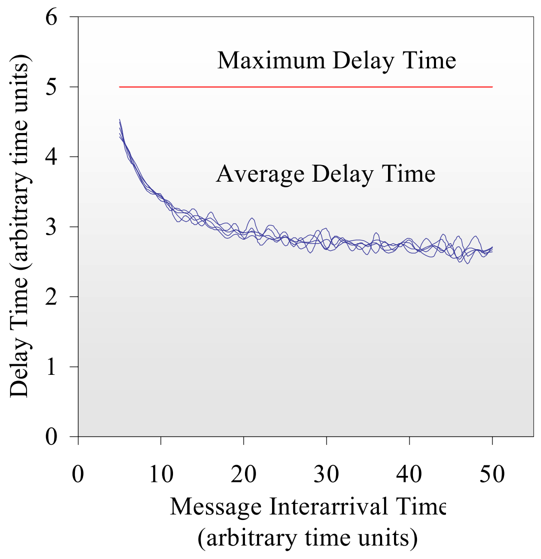

Fig. 2. Event-triggered performance.

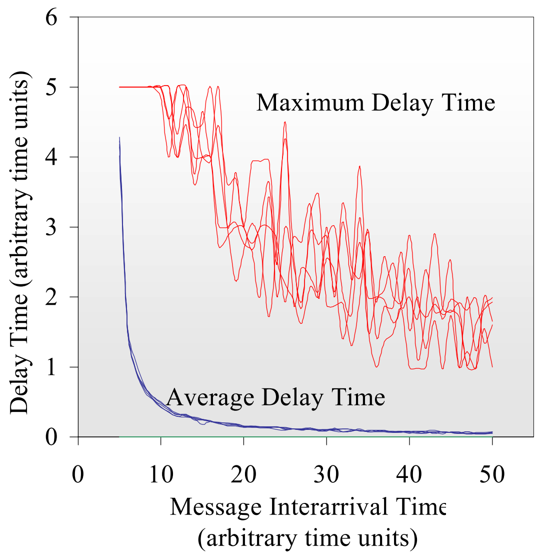

Fig. 3. Event-triggered, dynamic priority performance.

5. CONCLUSIONS

The proposed protocol assigns each node a priority that is used to arbitrate collisions (as in CAN bit-wise arbitration). Unlike the protocols described previously however, these priorities change dynamically. Once a node transmits a message, the node lowers its priority for a specified amount of time. In doing this, the node guarantees that it is not able to monopolize the bus because other nodes now have a higher priority than it does. This in turn guarantees that every node will not have to wait any longer than one cycle to broadcast.

A side benefit of this method is that it automatically provides a method to control babbling idiot failures. This could be accomplished for example by implementing a bus guardian of the loosely coupled or close coupled type described in (Broster and Burns, 1998). Implementing the bus guardian in this way could prevent a node from attempting to transmit multiple successive messages of the highest priority. The bus guardian would effectively limit the node by allowing it to only send a message with the lowest priority immediately after a higher priority message has been sent. The second message would then be transmitted immediately only if the bus is idle. The architecture of a bus guardian is one of the remaining issues to be researched further.

Another issue related to bus guardians is the potential for including the dynamic message priority setting intelligence solely within the bus guardian itself. In effect the bus guardian would be responsible for setting the message priority.

In this paper, we compared the proposed event-triggered dynamic priority protocol with existing protocols using a discrete-event simulation model (implemented in Arena (Kelton et al., 1998)). Initial analysis has shown that a fundamental assumption about event-triggered protocols is flawed. Specifically, with dynamic priorities we are able to set an upper bound on the delay time of the first message in queue at each node. This makes the proposed dynamic priority event- triggered system deterministic.

Future work will focus on evaluating the performance of the system with variable message lengths and transmission errors.

REFERENCES

Audsley, N., A. Burns, M. Richardson, K. Tindell and A. Wellings. (1993) Applying New Scheduling Theory to Static Priority Pre-Emptive Scheduling. *Software Engineering Journal*, 8(5): 285-292.

Almeida, L., Pedreiras, P., Fonseca, J., (2002) The FTT-CAN protocol: why and how, *IEEE Transactions On Industrial Electronics*, 49 (6), pp. 1189-1201.

Broster, I., Burns, A., (2001) The Babbling Idiot in Event-triggered Real-time Systems, *Proceedings of the Work-In-Progress Session, 22nd IEEE Real-Time Systems Symposium*, YCS 337.

Claesson, V., Ekelin, C., Suri, N., (2003) The event-triggered and time-triggered medium-access methods, *Proceedings of the IEEE International Symposium on Object-Oriented Real-Time Distributed Computing* (ISORC'03).

Ferreira, Pedreiras, Almeida and Fonseca (2001) The FTT-CAN protocol for flexibility in safety-critical systems, *IEEE Micro*, pp. 81-92.

Holonic Manufacturing Systems (2004) *Web Site*, http://hms.ifw.uni-hannover.de/.

Kelton, W., Sadowski, R., Sadowski, D. (1998) *Simulation with Arena*, New York: McGraw-Hill.

Kopetz, H., (2001) *A comparison of TTP/C and FlexRay*, TU Wien Research Report 2001/10.

Marsh, D., (2003) Network protocols compete for highway supremacy, *EDN Europe*, pp. 26-38.

McFarlane, D.C., and Bussmann, S. (2000) Developments in holonic production planning and control, Production Planning & Control, 11(6), pp. 522-536.

Moschovitis, C.J.P. (1999). History of the Internet. ABC Clio Ltd.

Robert Bosch GmbH. (1991). Bosch CAN Specification version 2.0.

Shaheen, S., Heernan, D., Leen, G., (2003) “A comparison of emerging time-triggered protocols for automotive X-by-wire control networks,” IMechE J. Automobile Engineering, 217(Part D), pp. 13-22.

Taylor, P., (2003) “Real-time, determinism and Ethernet.” Retrieved November 24, 2003 from <www.bara.org.uk/encyclopedia/ethernet/Hirschmann2.pdf>