An Analysable Bus-Guardian for Event-Triggered Communication

I. Broster, A. Burns
Real-time Systems Research Group
Department of Computer Science
University of York, UK.
{ianb,burns}@cs.york.ac.uk

Abstract

We present a guardian-based approach to detecting 'babbling idiots', faulty nodes which erroneously consume extra resource in an event triggered system. In general, one cannot detect all babbling idiots, but the maximum effect of undetected faults is bounded and small, and therefore can be taken into account in worst case response time analysis to guarantee that a babbling idiot cannot cause a timing failure elsewhere in the system. The approach is applied specifically to the CAN protocol to protect against faulty nodes transmitting message frames too often. We show that the overhead of including the effect of undetected frames into the worst case response time analysis is small enough to be of practical value.

1. Introduction

It has long been a criticism of event-triggered systems that they are unable to detect or tolerate *babbling idiot failures* or commission faults. A commission failure occurs if a process (or node) erroneously produces a result (or transmits a message, causes an event *etc.*) if none should have been produced. In a real-time system, this extends to an event that occurs too *early*.

Informally, the babbling idiot failure is where a process/node repeatedly suffers commission faults and therefore consumes more resources than it would normally use. This can lead to a resource starvation elsewhere. For example, consider a set of nodes communicating through a shared bus; if one node suffers a babbling idiot failure and begins to transmit extra messages onto the bus then it may starve the other correct nodes on the bus of network bandwidth. Protecting the shared resource from abuse by an incorrectly behaving node is the focus of this paper.

Time-triggered bus protocols such as TTP [12] dictate that a node is only allowed to write to the bus at predefined times. Therefore it is relatively easy to detect if a node becomes a babbling node by watching when it writes to the bus. Furthermore, it is possible to build a *bus guardian* [21] which only connects the node to the bus at these predefined times, hence ensuring that the node can never write to the bus when it should not do so. This creates a fail-silent architecture, which has been shown to be a very useful way of designing both time-triggered and event-triggered systems.

With event-triggered communication, it is not possible to exactly specify when a message will be sent. This flexibility is an advantage in many systems, but it makes babbling idiot detection (and hence the design of a bus guardian) significantly more difficult.

In this paper, we consider the implications of a bus guardian in an event-triggered communication system and propose an architecture, algorithm and timing analysis to detect the babbling idiot failure. The aim of the guardian system is to prevent a babbling node from starving other nodes on the bus of available bandwidth by enforcing fail-silence as soon as a babbling node is detected.

We show that although not all commission faults are guaranteed to be detected, by using the approach in this paper, the maximum effect of undetected commission faults is bounded. Therefore, this bounded effect can be introduced into a worst case response time analysis. This can ensure that even if a babbling idiot is present in the system, the interference it may cause to other parts of the system is not sufficient to lead to a timing failure.

There has been significant research concerning the effectiveness and applicability of the event-triggered approach in dependable systems [25, 19, 18, 8]. The existence of an effective Bus Guardian for event-triggered communication removes one barrier [17], (perhaps the final barrier) to the use of such networks in high-integrity applications.

This paper provides a discussion of architectures in Section 3, then Sections 4 and 5 explain how a bus guardian may be constructed for event-triggered communication. This theory is applied to CAN in Section 6. Section 7 provides two contrasting examples of applications, showing that the approach is practical. Conclusions are in Section 8.

2. Bus Guardians in Event-triggered Communication

The flexibility of an event-triggered communication bus such as CAN [3, 11] may provide many advantages over a time-triggered communication bus. For example, it allows very efficient error correction mechanisms to tolerate network-born faults such as electromagnetic interference which causes corruption of bits on the bus. An event-triggered bus also provides the ability to efficiently support non-periodic data, by transmitting such data on demand (only when it needs to be transmitted) rather than requiring a periodic slot reservation as an implementation in the time-triggered paradigm might do.

At the cost of this flexibility comes some uncertainty. It is clear that if there is a system requirement to transmit at arbitrary times then the guardian cannot know, in advance, the times at which a node will transmit.

A bus guardian is a device designed to protect a bus from failure of some component attached to the bus. The concept of a bus guardian to isolate a node in the event of failure is not new, it has been used at least as far back as the FTMP architecture in 1978 [9] to safely interface nodes to a multiple-redundant bus. Delta-4 [15] used a similar strategy, restricting node communication via 'network attachment controllers'. It is common for a bus guardian to either disconnect a node from the bus, or to shutdown the node, in order to prevent the faulty node from interfering with the correct operation of the other nodes in the system.

The causes of 'babbling idiots' are generally unknown (the fault is rarely observed) however the consequences of the problem may be severe. The two most likely causes of errors in the time domain are thought to be related to either erroneous internal states or errors in the clock source of the communication controller [2]. Both of these could lead to incorrectly timed access to the communications media. Alternatively an incident [10] where EMI (electromagnetic interference) caused a permanent change to a programmable ROM in a crane illustrates another possible cause of a babbling idiot failure.

In automotive applications, electronic control is taking an increasing role in safety-critical functions, brake-by-wire and steer-by-wire are expected to appear in production vehicles within the next 10 years [13]. The need for the braking and steering control messages to be transmitted irrespective of failures in other nodes is directly addressed in the requirements of the emerging FlexRay automotive bus protocol [2]: "For highly dependable and fault-tolerant applications, an independent bus guardian to prevent the monopolisation of the communication medium by a communication controller is required". As FlexRay is largely time-triggered, a FlexRay bus guardian is easy to construct.

Despite the need for babbling idiot protection in event-triggered communication, there has been little effort to provide an independent bus guardian for event-triggered bus communication to guard against commission failures. Tindell [24] suggested that specialised driver hardware may provide a robust defence against software failure leading to babbling idiot failure. Some very early work which forms the foundation of this paper was also published [4]. However, this is the extent of previous work on babbling idiot protection in event-triggered communication.

3. Architectures

This section discusses possible architectures for implementing a bus guardian, including a brief review of other approaches to building such devices. When designing a bus guardian, one of the most difficult problems is to ensure that there are no common failure modes between the bus guardian and the node(s) that it is protecting; a guardian would be of no use if it fails whenever the node that it is guarding fails. Some potential sources of common failures are: protocol implementation, protocol errors, clocks, CPU/hardware, power supply, operating system/software, common data, *etc.* Independent hardware with no common components and design diversity can be used to help avoid common failure modes.

However, some common elements cannot easily be removed. For example: it is apparent that the guardian and the node should have independent clocks in order to prevent a fast/slow clock being a common failure mode. However, if the clocks are totally independent then we must assume that the guardian and the node may experience clock drift relative to each other. Therefore we might impose some clock synchronisation mechanism upon both the guardian and the node and this introduces either a common component or a dependency between the guardian and the node.

The design of a bus guardian, in common with the design of any fault tolerant system, is full of many compromises between independence, coverage of faults and simplicity. Clearly, in any architecture, it is necessary for the guardian to have some *a priori* knowledge of the intended timing behaviour of the node(s) it is guarding.

3.1. Closely Coupled Guardian

The architecture in Figure 1 uses a very close coupling between the node and guardian. It is similar to the *network attachment controller* approach in Delta-4 [15] and the hardware supported approach for CAN [3] suggested by Tindell [24].

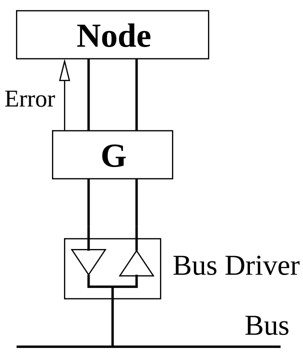

Figure 1. Closely Coupled Guardian.

In this architecture, the guardian (**G**) is really a highly reliable network interface with additional logic to prevent the node from transmitting too frequently. It must understand the network protocol which adds complexity to the guardian. As the guardian provides the interface to the network, faults in the guardian could still lead to babbling, but redundant hardware could be used to provide fail silent dependability. Independence can be criticised in this model due to the close coupling of the node and the guardian.

3.2. Loosely Coupled Guardian

Alternatively the slightly less closely coupled strategy in Figure 2 can be used. It is similar to the techniques used by TTP [21] and FTMP [9]. The node always listens to the bus, but transmission is only allowed under the control of the guardian.

Normally the bus drivers are switched ‘off’; before attempting to transmit to the bus, the node must inform the guardian that it wishes to transmit. If the guardian agrees with the request then it enables the bus driver. Afterwards, the guardian disables the driver. This scheme provides significantly more isolation and

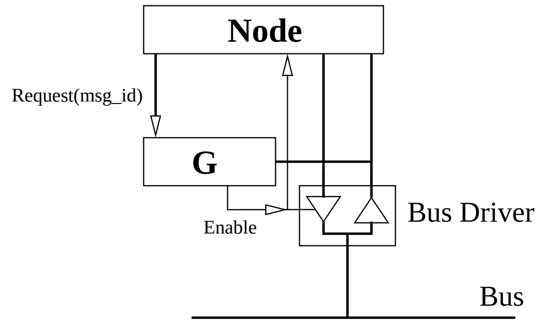

Figure 2. Loosely Coupled Guardian.

provides fail silence: in order for any message to appear on the bus, both the guardian and the node must concur. However, as illustrated by the TTP guardian, clock synchronisation is crucial and could be a cause of a common mode failure.

3.3. Independent Guardian

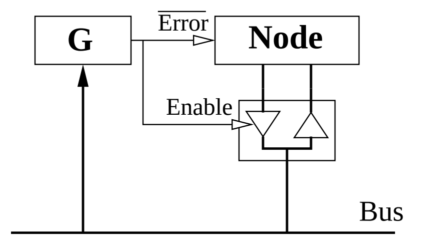

Figure 3. Independent Guardian.

A third strategy is shown in Figure 3 which provides even more independence between the node and the guardian by having the guardian as a completely separate node connected directly to the bus (in a read-only configuration). Using only information from the bus, the guardian monitors transmissions and detects babbling nodes.

If a node is found to be babbling, then the guardian is able to affect (*e.g.* shutdown or disconnect) a node using a direct link to the node (this being the only connection between the guardian and the node). For a system where nodes are widely distributed, a guardian will only have a direct connection to nodes in the same physical location, therefore one guardian is required per ‘cluster’.

This model requires no special features of the processor node itself, but the guardian is a fairly complex device because it must understand the network protocol.

A disadvantage of this approach is that the guardian is only able to detect a babbling node a *after* it has transmitted an incorrect message onto the bus. However, as the analysis later in the paper shows, this is not a significant drawback. Additionally, this architecture is not able to defend against one node masquerading as another, although additional logic can be supplied to protect against this.

The advantages of independence of the node and guardian are clear in the third architecture, where the node and the guardian are completely separate nodes. The guardian plays a passive role in network activities, but is able to disconnect/disable a node that it is guarding if it detects any violation. This architecture is the one that will be used in the following analysis.

4. Definitions and Model

An event-triggered bus is considered (such as a CAN bus), characterised by the bus scheduling decisions being made at run-time based on which nodes wish to transmit at any given time. The particular scheduling regime is not significant at this stage, but fixed priority (as for CAN) or earliest deadline first scheduling schemes are both entirely suitable. Analysis for a fixed priority scheme is considered later.

The instant that a node requests a message transmission is termed a 'trigger event', and let S_i be the absolute time that trigger event i occurs. The message is then queued for transmission according to the bus scheduling scheme.

A system of sporadic streams is considered, where there is a minimum 'inter-arrival' time, T, between the generation of any two consecutive trigger events in a stream. This does not imply that the frames will be separated by this time when they are transmitted on the bus, merely that the node should not request two message transmissions closer than the minimum inter-arrival time. The *sporadic invariant* can be written:

$$\forall i \;\; S_{i+1} - S_i \geq T \tag{1}$$

For clarity, only one stream is initially considered, but later analysis will be applied to multiple streams.

The sporadic model is chosen because it is a general model for which the worst case response time of a message in a stream can be calculated. When considering worst case analysis, the sporadic model also incorporates the periodic model, so the analysis in this paper applies equally well to periodic messages with period T and sporadic messages with minimum inter-arrival time T.

A commission failure has occurred when the sporadic invariant condition in equation (1) is broken. Since the independent nodes architecture in Figure 3 is used, equation (1) cannot be verified directly because S_i is known only to the node where the event occurred and not to the guardian. Instead, the guardian node listens to the bus traffic observing when each activity (message frame) finishes. Let E_i be the observed finalisation time of the activity triggered at S_i.

For any given stream, the guardian knows (in advance) the following 'common knowledge' [7] which is determined off-line:

R is the worst case response time of the activity triggered by event i, *i.e.* the finalisation time may occur any time between S_i and $S_i + R$. In the case of CAN, this is calculated using the fixed priority worst case response time formulation, which has been well documented elsewhere [23]. It is expected that there are mechanisms in the system to ensure that R_i cannot be exceeded, one scheme for ensuring this in CAN is described elsewhere [5]. We assume that the only possible failure is the babbling idiot failure.

T is the known minimum inter-arrival time between trigger-events.

The babbling idiot problem presents itself in subtly different ways in CPU scheduling and bus scheduling. With CPU scheduling, the operating system has knowledge of the details of each trigger event and therefore can detect the babbling idiot at source by monitoring the spacing between consecutive trigger events. The implicit assumption is that the operating system can be trusted, and the incorrect trigger events are caused by either something external to the system, or a software failure external to the operating system.

On the other hand, in a bus system, the protection is to be applied to the bus against the failure of a node. No trust can be placed in the node. The values of S_i cannot be known to any external guardian. Hence the guardian only has the ability to detect the messages after they are transmitted, *i.e.* it knows E_i. In specific cases it may be able to detect a message transmission earlier (for CAN, a message transmission can be identified immediately after the arbitration field which occurs at the start of the frame).

5. Approach to Detection

In this section, the guardian algorithm is described. The overall approach is to monitor the finalisation times $(E_i, E_{i+1}, \ldots)$ and for each message in a stream, the guardian will calculate a threshold value,

$G_{i+1}, G_{i+2}, \ldots$, which represents the earliest acceptable value for $S_{i+1}, S_{i+2} \ldots$. Since $S_i \leq E_i \leq S_i + R$, if $E_{i+1} < G_{i+1}$ then some babbling in the stream has occurred. This is described in more detail below.

The threshold values can be regarded as conservative 'estimates' for the earliest possible time of the trigger events. 'Conservative' in this context means that a threshold/estimate must always be less than or equal to the time of the actual trigger event. The activity of the guardian is described by considering the first invocation in a stream, then the subsequent invocations.

First Invocation. Consider some invocation i which is either the first invocation, or which follows a long gap where there were no events. The guardian observes that the invocation completes at time E_i. The maximum time between release (trigger event) and completion is R, therefore the guardian can deduce a bound on the earliest possible time that this invocation was released:

$$S_i \geq E_i - R \tag{2}$$

Therefore, if the sporadic invariant condition (1) is preserved, then the guardian knows that the earliest time that it should expect the next invocation is $G_{i+1} = E_i - R + T$.

Subsequent Invocations. For subsequent invocations, there are several possibilities.

1. If the next invocation (E_{i+1}) is observed to arrive before G_{i+1} then the guardian has detected a babbling node and will disable the node.
2. Otherwise, the guardian *assumes that the node is not babbling* and calculates the next threshold by considering the earliest possible trigger event that could give rise to this observation. From the bound in (2) the earliest possible trigger event for E_{i+1} is at $E_{i+1} - R$.
   - If $E_{i+1} - R$ occurs on or after the threshold G_{i+1} (which is an estimate of S_{i+1}) then the next threshold can safely be set to $E_{i+1} - R + T$. Figure 4 shows this in a diagrammatic form.
   - If $E_{i+1} - R$ occurs before the threshold G_{i+1} then since we assume that the node is not babbling, the guardian concludes that G_{i+1} is already a better estimate for S_{i+1} than $E_i - R$, so it sets the next threshold to be $G_{i+1} + T$. Figure 5 shows this.

The rules for calculating G_{i+1} can be stated succinctly as:

$$G_0 = -\infty$$
$$G_{i+1} = \max(G_i + T, E_i - R + T)$$

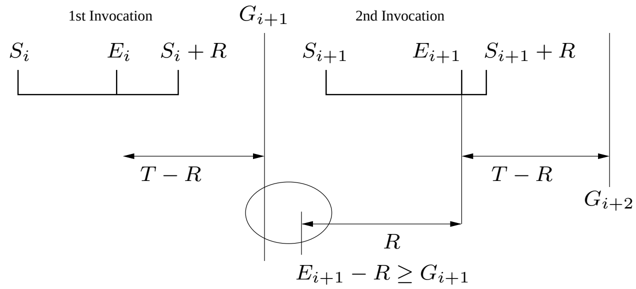

Figure 4. New Threshold Based on New Observation.

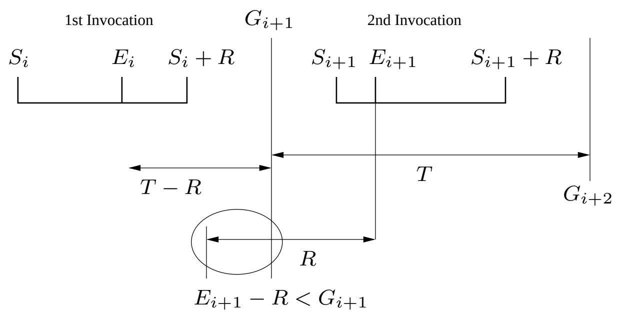

Figure 5. New Threshold based on Previous Observations.

or

$$G_{i+1} = \max(G_i, E_i - R) + T \tag{3}$$

The guardian rules are easily implemented. Figure 6 presents the guardian in algorithmic form.

5.1. Undetectable Babbling

The guardian described in the previous section cannot detect all commission failures since knowledge of E_i is not sufficient to derive an exact value for S_i. Recall

$G \leftarrow -\infty$;
while true **do**
  E ←Get_New_E();
  if $E < G$ **then** -- babbling detected
    Shutdown();
  else
    $G \leftarrow \max(G, E - R) + T$
  end if
end while

Figure 6. Guardian Implementation.

the *assumption* in the previous section that if $E_i \geq G_i$ then the node is not babbling—it is possible to create scenarios where $E_i \geq G_i$ yet the sporadic invariant (1) is broken—this assumption is at the root of undetectable babbling. Nevertheless, the following questions can be answered:

1. What is the worst case traffic that the guardian will allow onto the network?
2. When will babbling be detected?

In this paper, the concern with babbling nodes is the effect that they will have on the available bandwidth for other messages, therefore the answer to the first question is particularly crucial.

In scheduling analysis we are concerned with the load asserted by each source of work within the current time interval of interest. Let t denote the length of this interval. For example, in fixed priority scheduling, we know that a sporadic stream with minimum inter-arrival time T (or periodic stream with period T) will, in the worst-case, produce the following load within t:

$$\text{Load} = \left\lceil \frac{t}{T} \right\rceil C$$

where C is the time necessary to send each message.

For an arbitrary stream protected by a guardian, the worst case load will occur when the guardian allows work through at the maximum volume and then detects a babbling node and kills the node. Let F be the time needed for this detection. It follows that

$$\text{Load} = (\text{Max. number of releases in } t)C + F \qquad (4)$$

This represents the worst case traffic that the guardian will allow onto the network (regardless of whether a node is babbling or not). An exact formulation for the maximum number of releases in t is given in the following theorem:

Theorem 1. *The maximum number of acceptable releases, in any time interval of length t, generated by a guarded message stream with minimum inter-arrival interval time, T, and a worst case response time of R is given by*

$$g = \left\lceil \frac{t+R}{T} \right\rceil \qquad (5)$$

Discussion. Before providing a proof of this result, we first give some intuition behind the formulation. In the worst case, the start of the interval will coincide with a message being sent; moreover this message is the first that has been transmitted for some time. The guardian will then allow the second message through if it occurs after $T - R$, this being the minimum gap at any time. Following this minimum gap subsequent messages can arrive no sooner than one every T. If any message arrives later than it cannot be compensated by a subsequent earlier release (compensated in the sense of to give more overall messages in any interval). This behaviour is identical to that which is exhibited by a stream (or task) suffering release jitter [22]:

$$\text{Load} = \left\lceil \frac{t+J}{T} \right\rceil C$$

where J is the jitter.

With the guardian the jitter is equal to R, and hence equation (5) would be expected to represent the worst-case. To prove the theorem we first need the following result.

Lemma 1. *The general form for the guardian threshold equation* (3) *is*

$$G_i = \max_{j \in \{0..i-1\}} (E_j - R + (i-j)T) \qquad (6)$$

Proof. By induction. The base case is trivially derived from equation (3) where $i = 0$ and $G_0 = -\infty$.

The inductive step, applying equation (3) to equation (6).

$$\begin{aligned} G_{i+1} &= \max(\max_{j \in \{0..i-1\}} (E_j - R + (i-j)T), \\ &\qquad E_i - R) + T \\ &= \max(\max_{j \in \{0..i-1\}} (E_j - R + (i+1-j)T), \\ &\qquad E_i - R + (i+1-i)T) \\ &= \max_{j \in \{0..i\}} (E_j - R + (i+1-j)T) \qquad \square \end{aligned}$$

Proof of Theorem 1. We may assume that in the worst case, this interval t begins at an observation time E_a, so the observation is included in the interval. So, the interval is $[E_a, E_a + t)$.

If $t < T - R$ then there will be only one such message, because $T - R$ is the minimum gap. Theorem 1 holds for such values of t. For values of t greater than $T - R$ there may be two or more messages. Let the last message in the interval have the guardian threshold time G_b. So, $G_b < t + E_a$, or

$$G_b - E_a < t \qquad (7)$$

So, considering G_b,

$$G_b = \max_{j \in \{0..b-1\}} (E_j - R + (b-j)T)$$

and appealing to the property of the max function,

$$\forall j \in \{0..b-1\},\ G_b \geq E_j - R + (b-j)T$$

Therefore, considering the case $j = a$, it is true to say

$$G_b \geq E_a - R + (b - a)T$$

hence

$$b - a \leq \frac{G_b - E_a + R}{T}$$

Appealing to inequality (7),

$$b - a < \frac{t + R}{T}$$

Let g be defined as the maximum number of messages in the interval; both a and b are in the interval so

$$g = b - a + 1$$

$$\therefore \quad g = \left\lceil \frac{t + R}{T} \right\rceil \qquad \square$$

Therefore, by equation (4), the worst case traffic that a guardian will allow onto the network is bounded.

Returning to the second question of when the babbling will be detected: a commission failure will be detected immediately after the first invocation that exceeds the limit given in Theorem 1.

6. Analysis of CAN

The previous section showed that the worst case undetected babbling is bounded. Note that until now there has been no requirement for any particular bus system or scheduling algorithm (although fixed priority was assumed in some discussion). In this section, we incorporate the previous results to CAN worst case response time analysis.

6.1. CAN Specifics

Firstly, note that we may exploit some of the specific properties of CAN. A stream is identified by the *arbitration field* which is transmitted at the beginning of each frame. Therefore, the guardian is able to identify the stream immediately after the arbitration field has been transmitted. When a node babbles, it can be shutdown/disconnected within the 12 bits of the arbitration field (or the 29 bits for the longer arbitration field in extended CAN). Following this, there may be up to 29 bits for an error frame to be transmitted before the bus becomes available again.

Therefore the worst case load (interference) that a stream j (babbling or otherwise) may impose during an interval t is

$$I(t) = \left\lceil \frac{t + R_j + \Delta}{T_j} \right\rceil (C_j + S) + F \tag{8}$$

where F is the longest possible time taken to detect and recover from a babbling node; $F = 41\tau$ for CAN, where τ is the time of one bit on the bus; $S = 3\tau$, the time taken to transmit the interframe space.

Note that a slight change in notation is required now because we are dealing with multiple streams. In this section, subscripts ($_i$) will be used to denote a stream, rather than invocations within a stream.

The addition of Δ in the ceiling function is to ensure that edge effects [1] cannot lead to an optimistic analysis as the protocol is non-preemptive. In practice, it usually makes no difference to the results if Δ is not included. It is normal to use $\Delta = \tau$.

6.2. Worst Case Response Time

Equation (8) can be incorporated into the usual WCRT analysis of equations for CAN. The worst case response time for a frame in a stream of priority i on CAN with no babbling idiot (or any other faults) is given in equation (9) which is based on equations by Tindell [23].

$$R_i = \max_{\forall k \in \mathsf{lp}(i)} (C_k) + S + C_i + I_i(R_i - C_i) \tag{9}$$

Where $I_i(t)$ is the total interference from higher priority frames; C_i is the worst case time to transmit a frame k, given by equation (10) [16, 14] and is related to the number of bytes, b, in the frame.

$$C_i = \left(44 + 8b + \left\lfloor \frac{34 + 8b - 1}{4} \right\rfloor \right) \tau \tag{10}$$

Equation (9) can be solved iteratively by forming a recurrence relation in the familiar way for worst case response time equations [6].

In the absence of any babbling, interference is calculated by summation of higher priority interference using equation (11).

$$I_i(t) = \sum_{j \in \mathsf{hp}(i)} \left\lceil \frac{t + \Delta}{T_j} \right\rceil (C_j + S) \tag{11}$$

Incorporating babbling protection from equation (8), the worst case interference that traffic from higher priority nodes may impose if they are babbling is given by equation (12)

$$I_i(t) = \sum_{j \in \mathsf{hp}(i)} \left(\left\lceil \frac{t + R_j + \Delta}{T_j} \right\rceil (C_j + S) + F \right) \tag{12}$$

Single Failure Assumption. The probability of all nodes babbling simultaneously is likely to be very small. Further, if such an event were to occur then it would be an indication of some more serious underlying problem in which case a guardian would be useless. Therefore, we make the reasonable assumption that only one stream will babble at any time, all other streams behave as expected. This leads to the following equation for worst case interference.

Let $b \in \mathsf{hp}(i)$ be the stream that is babbling, so

$$\begin{aligned} I_{i,b}(t) = & \sum_{j \in \mathsf{hp}(i)-\{b\}} \left\lceil \frac{t+\Delta}{T_j} \right\rceil (C_j + S) \\ & + \left\lceil \frac{t + R_b + \Delta}{T_b} \right\rceil (C_b + S) + F \end{aligned} \tag{13}$$

which we would wish to maximise over b:

$$I_i(t) = \max_{b \in \mathsf{hp}(i)} I_{i,b}(t) \tag{14}$$

Note that following the single failure assumption, in equation (14) for the value of R_b we may use the normal worst case response time of stream b without considering interference from babbling nodes. If stream b is babbling then no other stream is babbling; therefore the worst case response time R_b is no worse than the normal worst case response time.

7. Examples and Evaluation

In this section, the worst case response time analysis of the previous section is applied to two CAN message sets. We show that the increased worst case response times incorporating a babbling node are small enough to be practical. Two contrasting examples are shown, one with a large random message set, and one based on the smaller, frequently referenced SAE benchmark [20] reported by Tindell [23].

A large periodic message set with 80 different streams and a utilisation of 46% was generated with random periods distributed uniformly between 5 ms and 100 ms. This message set was chosen to show that the approach is scalable for a large number of message streams. The bus speed is 500 kbits/s. For reasons of space, the full message set is not tabulated here, but shown as a graph in Figure 7. Note that priority 1 represents the lowest priority.

The graph shows that for the single failure assumption, the increase in worst case response time by including babbling idiot protection is very small. Further, worst case response time with the worst case babbling overhead does not grow significantly faster than normal worst case response time increases at lower priorities.

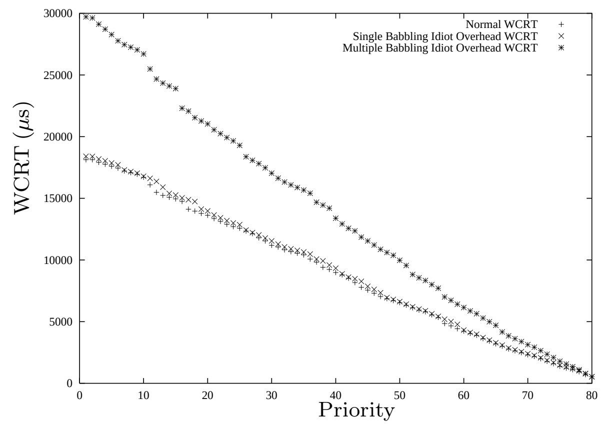

Figure 7. Worst Case response times, comparing usual WCRT analysis with Babbling Idiot overhead.

For multiple failures (all nodes babble in the worst possible way), the overhead is more significant, but only growing to just over 1.6 times the normal worst case response time for the lowest priority stream, which is still a realistic and manageable overhead.

The second example, which we will examine in more detail, is based on the benchmark produced by the Society of Automotive Engineers (SAE) [20]. This is a well-standing and often quoted set of messages that might be transmitted within a vehicle. This message set has frequently been used as a benchmark to evaluate CAN. We will use a modified form of the benchmark derived from Tindell [23] which has 17 periodic message streams.

This benchmark is normally considered at the very low data-rate of 125 kbit/s, where it produces a very high bus utilisation of around 85%. Although all deadlines are normally met in this benchmark, it has very little spare capacity for error correction of any kind, and hence would probably not be used in a reliable real-time system. In this paper, we will use a faster rate of 250 kbit/s which is still well below the maximum data-rate of CAN, and still able to provide a very stable communication channel.

The benchmark and the results of applying the babbling idiot worst case response time equations are shown in Table 1. For comparison, the 'Normal' column shows the worst case response time if no node is babbling, the '1 Fail.' column shows the worst case response time if a single babbling failure of the worst possible magnitude occurs somewhere else in the system, and the 'Max Fail.' column shows the worst case response time if all nodes in the system babble in their worst possible way. The results confirm that incorporating a single babbling failure into worst case response time analysis does not significantly increase the worst case response times. Incorporating multiple failures increases the worst case response times significantly, but not to the extent that the analysis is useless. For this task set, all frames are still guaranteed to meet their deadlines if all other nodes babble (although as previously indicated, this is not a realistic scenario).

i	C_i (μs)	T_i (ms)	Normal R_i (μs)	1 Fail. R_i (μs)	Max Fail. R_i (μs)
17	248	1000	708	708	708
16	288	5	1008	1172	1172
15	248	5	1268	1432	1596
14	288	5	1568	1732	2060
13	248	5	1828	1992	2484
12	288	5	2128	2292	3208
11	448	10	2508	2672	4912
10	248	10	2768	2932	5796
9	288	10	3068	3532	6520
8	288	10	3368	3832	7584
7	248	100	3628	4092	9428
6	368	100	3968	4432	9932
5	248	100	4228	4692	10356
4	248	100	4488	4952	10780
3	328	1000	4748	5212	11204
2	248	1000	5008	6892	11628
1	248	1000	5020	6904	11804

Table 1. SAE Benchmark Message Set.

8. Conclusion

This paper has shown that it is possible to provide protection from babbling idiots and commission failures in event-triggered real-time communication. The approach uses an external guardian which is a completely independent node on the bus. Although the guardian cannot detect all babbling failures, the worst case babbling that cannot be detected by a guardian is identifiable. The interference that this babbling may induce on lower priority frames is bounded and can therefore be incorporated into worst case response time analysis. This paper shows how that can be done for a specific bus, CAN. The worst case response time for any frame taking into account the additional overhead from any undetected babbling is only slightly larger than the worst case response time if babbling cannot occur. This is shown by means of two contrasting examples.

The approach in this paper is presented in a generic manor that is applicable to many event-triggered buses. It may be possible to apply this approach directly to EDF scheduled buses, as well as to internet-based real-time protocols. This may be considered in future work.

The overall impact of the analysis reported in this paper is to confirm the role event-triggered protocols can have in high integrity applications.

References

[1] I. J. Bate. *Scheduling and Timing Analysis for Safety Critical Real-Time Systems.* PhD thesis, Dept. of Computer Science, University of York, York, YO10 5DD, 1999.

[2] R. Belschner, J. Berwanger, C. Ebner, H. Eisele, S. Führer, T. Forest, T. Führer, F. Hartwich, B. Hedenetz, R. Hugel, A. Knapp, J. Krammer, A. Millsap, B. Müller, M. Peller, and A. Schedl. *FlexRay Requirements Specification.* FlexRay, www.flexray.com, v2.0.2 edition, April 2002.

[3] Bosch, Postfach 50, D-700 Stuttgart 1. *CAN Specification*, version 2.0 edition, 1991.

[4] I. Broster and A. Burns. The babbling idiot in event-triggered real-time systems. In G. Fohler, editor, *Proceedings of the Work-In-Progress Session, 22nd IEEE Real-Time Systems Symposium, YCS 337*, pages 25–28. IEEE, Department of Computer Science, University of York, 2001.

[5] I. Broster and A. Burns. Timely use of the CAN protocol in critical hard real-time systems with faults. In *Proceedings of the 13th Euromicro Conference on Real-time Systems*, pages 95–102, Delft, The Netherlands, June 2001. IEEE.

[6] A. Burns and A. J. Wellings. *Real-Time Systems and Programming Languages.* Addison Wesley, 3rd edition, 2001.

[7] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, July 1990.

[8] H. Hansson, C. Norström, and S. Punnekkat. A simulation based approach for estimating the reliability of distributed real-time systems. In *Proceedings of ETFA*, Proceedings of ETFA, Oct 2001.

[9] A. L. Hopkins, T. B. Smith, and J. H. Lala. FTMP - a highly reliable fault-tolerant multiprocessor for aircraft. *Proceedings of the IEEE*, 66(10):1221–39, October 1978.

[10] IEE. EMC and functional safety. IEE guidance document, IEE, Sept 2000. Available from http://www.iee.org.uk/PAB/EMC/core.htm.

[11] International Standards Organisation. *ISO 11898. Road Vehicles—Interchange of digital information—Controller area network (CAN) for high speed communication*, 1993.

[12] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Kluwer Academic, 1997.

[13] G. Leen and D. Heffernan. Expanding automotive electronic systems. *IEEE Computer*, 0018-9162/02:88–93, Jan 2002.

[14] T. Nolte, H. Hansson, C. Norström, and S. Punnekkat. Using bit-stuffing distributions in CAN analysis. In *IEEE Real-Time Embedded Systems Workshop at the Real-Time Systems Symposium*, London, UK, 2001.

[15] D. Powell. *Delta-4: A Generic Architecture for Dependable Distributed Computing.* Number 818/2252 in Research Reports, ESPRIT 818/2252. Springer-Verlag, 1991. ISBN 3-540-54985-4.

[16] S. Punnekkat, H. Hansson, and C. Norström. Response time analysis under errors for CAN. In *Proceedings of RTAS 2000*, pages 258–265, Washington DC, 2000. IEEE.

[17] J. Rufino. *Computational System for Real-time Distributed Control.* PhD thesis, Universidade Técnica de Lisboa Instito Superior Técnico, July 2002.

[18] J. Rufino, P. Veríssimo, and G. Arroz. Defining a CAN-based infrastructure for fault-tolerant real-time distributed computing. In *Proceedings of the 19th Real-Time Systems Symposium*, Work In Progress, pages 27–30, Madrid, Spain, Dec 1998. IEEE. (published as Technical Report UNL-CSE-98-002, from University of Nebraska-Lincoln, Department of Computer Science and Engineering).

[19] J. Rufino, P. Veríssimo, and G. Arroz. Embedded platforms for distributed real-time computing: Challenges and results. In *Proceedings of the 2nd International Symposium on Object-oriented Real-time distributed Computing*, pages 147–152, Saint Malo, France, May 1999. IEEE.

[20] SAE. Class C application requirement considerations. Technical Report J2056/1, Society of Automotive Engineers, 1993.

[21] C. Temple. Avoiding the babbling-idiot failure in a time-triggered communication system. In *Proceedings 28th Annual International Fault Tolerant Computing Symposium, FTCS'98*, 1998.

[22] K. Tindell. *Fixed Priority Scheduling of Hard Real-Time Systems.* PhD thesis, Department of Computer Science, University of York, 1994.

[23] K. Tindell, A. Burns, and A. J. Wellings. Calculating controller area network (CAN) message response times. *Control Engineering Practice*, 3(8):1163–1169, 1995.

[24] K. Tindell and H. Hansson. Babbling idiots, the dual-priority protocol, and smart CAN controllers. In *Proceedings of the 2nd International CAN Conference*, pages 7.22–28, 1995.

[25] P. Veríssimo, J. Rufino, and L. Ming. How hard is hard real-time communication on field-buses? In *Digest of Papers, The 27th International Symposium on Fault-Tolerant Computing Systems*, pages 112–121, Seattle, Washington, USA, June 1997. IEEE.