Self-organized Message Scheduling for Asynchronous Distributed Embedded Systems

Tobias Ziermann[1,⋆], Zoran Salcic[2], and Jürgen Teich[1]

[1] University of Erlangen-Nuremberg, Germany
{tobias.ziermann,teich}@informatik.uni-erlangen.de
[2] The University of Auckland, New Zealand

Abstract. A growing number of control systems are distributed and based on the use of a communication bus. The distributed nodes execute periodic tasks, which access the bus by releasing the messages using a priority-based mechanism with the goal of minimal message response times. Instead of randomly accessing the bus, a dynamic scheduling of messages technique based on adaptation of time offsets between message releases is used. The presented algorithm, called DynOAA, is executing on each node of the distributed system. It takes into account the current traffic on the bus and tries to avoid simultaneous release of messages by different nodes, hence reduces the likelihood of conflicts and need for repeated release. In this paper, we first address single bus (segment) systems and then extend the model and the offset adaptation algorithm to systems that use multiple buses (segments) connected by a communication gateway. A rating function based on the average of maximum response times is used to analyze DynOAA for the case of CAN-bus systems based on bit-accurate simulations. Experiments show the robustness of the algorithm (1) in case of fully asynchronous systems, (2) ability to deal with systems that change their configuration (add or remove message release nodes) dynamically and (3) model systems containing multiple bus segments connected by a gateway. The approach is also applicable to other priority-based bus systems.

1 Introduction

In this paper, we target distributed control embedded systems based on the use of multiple computing nodes connected by a communication bus. Typical applications include automotive systems, industrial automation, home automation, healthcare systems and robotics. The common characteristic of such applications is that the communication over the bus is triggered periodically and the amount of data exchanged is relatively small. However, the time between sending data from the source node to receiving it on the destination node, called

⋆ This work was supported in part by the German Research Foundation (DFG) under contract TE 163/15-1.

J.M. Alcaraz Calero et al. (Eds.): ATC 2011, LNCS 6906, pp. 132–148, 2011.

message response time, is crucial and subject to real-time constraints. An example of communication bus and protocol perfectly suited for this task is the Controller Area Network (CAN) [1]. In this paper, we will focus on CAN, but the methods introduced are in principle suited for any priority-based communication protocol.

The design of the communication system in distributed control systems is a very complex task, which is very often performed using hand-based procedures, which is error prone, and time consuming when needed to repeat because of the change of system configuration and requirements. The use of automated methods and tools is a better option, but fails if the design is faced with large design space, which increases the computational complexity of the task, hence making design tools effectively unusable. Also, the use of design tools is based on the assumption that all design parameters and inputs are known in advance, which most often is not the case. For example, message release offsets [7], an efficient technique to reduce response times of the communication bus, can be calculated in advance, but the assumption that their statically calculated values will be the best throughout system operation results in a pessimistic solution, with non-optimal use of the system bus. We want to avoid these disadvantages of offline methods by considering the current traffic situation and adapt online.

We recently proposed [12] a solution of dynamically adapting the message release offsets by individual nodes (and software tasks) of the distributed system, which resulted in the reduced message response times for CAN-based systems. Compared to our initial work, this paper presents several new contributions: (1) The formal model is refined and extended to systems with multiple bus segments. (2) The rating function of the schedule is further improved to allow a better comparison between schedules of different approaches and scenarios. (3) We loosen the assumption that the monitoring period of DynOAA is synchronized and demonstrate that it has only little influence on the performance of the method. (4) We conducted experiments which incorporate dynamic changes of the system configuration during run-time and show the robustness of DynOAA. (5) The behavior of the system is experimentally analyzed for the case that only a fraction of all nodes apply DynOAA. Finally, (6) first preliminary results for using DynOAA on systems with multiple bus segments are obtained and reported.

The rest of the paper is organized as follows. In Section 2, we define the problem and position our work in the context of related work. Section 3 gives the details of the proposed dynamic adaptation method, the dynamic offset adaptation algorithm (DynOAA), and illustrates its operation for single-segment systems. Section 4 further extends DynOAA to allow the use in multi-segment systems and evaluates it by the developed simulator. Conclusions and future work are given in Section 5.

2 Problem Definition and Related Work

In this section, we outline the assumptions for the modeling of CAN-based systems and position our work to the existing related work. The measure of performance in the form of a rating function is also introduced.

2.1 Context

The CAN specification [1] defines the data link layer and roughly describes the physical layer of the ISO/OSI-Model. At the data link layer, a message-oriented approach is chosen. Four different types of frames are used to transfer messages: Data frames, Remote Transmit Request frames, Overload frames and Error frames. The most important is the data frame that is used for data exchange. Each data frame has its unique identifier. This 11 bit long identifier defines the message priority by which the bus access is granted. Bus arbitration is done by Carrier Sense Multiple Access with Bitwise Arbitration (CSMA/BA). The method of bitwise arbitration can be described as follows: Each node that would like to have access to the bus, starts sending its message as soon as the bus is idle for the time of 3 bits. Every sent bit is also watched. When the sent bit differs from the watched one, then a message with higher priority is also sending and transmission is stopped. After sending the identifier, only the message with the highest priority is left and has exclusive bus access.

The major advantage of using CAN is that its huge popularity, particularly in the automotive sector, which allows mass production of the CAN controllers and their integration with the microprocessors used in the computation nodes, resulting in very cheap solutions. Another advantage is availability of software built for the CAN infrastructure [9], which is the result of many years of use of CAN. On the other hand, a big disadvantage is the limited maximal data rate of 1 Mbit/s, which is the relic of the early conception of the bus, and significantly limits its application domain. From this reason, other communication buses such as Flexray [6] and real-time Ethernet [5], have been proposed. However, the introduction of new buses is related to a significantly increased cost to design and manufacturing of new computation nodes, which includes both hardware and software which already exist for CAN-based systems. Our work therefore goes in the other direction, where we are trying to breathe new life into CAN-based systems by allowing better use of the available data rate and increasing the performance of systems built on the CAN bus without changes of the system infrastructure.

The main problem caused by the limited bandwidth is the increase in response times, defined as the time between attempts to release the message to the time when actual message transfer begins, especially when the workload of the system increases (typically above 50%). This increase is caused by simultaneous attempts of multiple tasks to access the bus and release a message. Several approaches try to optimize the scheduling of messages on CAN by adjusting the priority of the messages. One way is to divide the message streams into different categories and schedule them with known scheduling techniques such

as earliest deadline first [13,4]. Another way is to use fuzzy logic to select the priority [2]. The different priorities are established by using several bits from the identifier. This has the disadvantage that the number of available identifiers is reduced from 2048 to 32 and 128, respectively. Already in current automotive applications more than 128 identifiers are needed, which makes this suggestion infeasible.

Therefore, we are using a different, less interfering, approach: If tasks send messages periodically, which is most often the case in distributed control systems, simultaneous access can be avoided by adding an appropriate offset to a message release time. An approach that assigns statically calculated fixed offsets of messages that are assumed to be synchronous by using off-line heuristics is proposed in [8]. Although the approach results in better response times, it does not take into account the dynamics of the system operation and resulting traffic. Instead, the approach is based on a-priori given, static assumptions. Additionally, it doesn't consider the asynchronous nature of the nodes (and tasks) that communicate over CAN bus, which is caused by each node running on its own clock. Because of different clocks and clock drifts, the optimal offsets ideally should change dynamically and adapt to the changing conditions on the bus. This is further supported by the fact that the start of the tasks is not synchronized, resulting in additional randomization of initial offsets. In order to solve this, the dynamic offset adaptation algorithm (DynOAA) has been proposed [12] recently as the solution to the changes of the recent bus traffic, resulting in better response times and also in an increased level of scheduling fairness.

2.2 System Model

The system we are targeting can be described by a set of nodes communicating over the bus as shown in Fig. 1. One or more tasks on each node may initiate a communication, i.e. release messages.

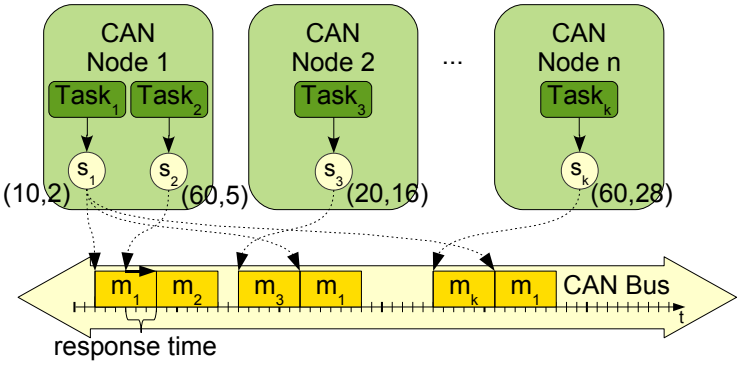

Fig. 1. CAN-Bus based system model

In our model, we abstract the tasks by considering only the mechanism used to release *messages* called a *stream*. A stream s_i can be characterized by a tuple (T_i, O_i) with $0 \leq O_i \leq T_i$, that is, by a period T_i (time between any two consecutive messages generated by stream s_i) and an offset O_i. The offset is relative to a global time reference. It can therefore drift over time, because the local time reference can differ from the global one. The *hyper-period* P is the least common multiple of all periods $lcm\{T_1, T_2, ..., T_k\}$. Assuming a synchronous system, the schedule is finally periodic with the hyper-period. A *scenario* consists of k streams. We assume the priorities are set by the designer, typically according to the stream period so that a rate monotonic scheduling is achieved.

A *message* m_i is a single release or CAN frame of the stream. The time between a message release and the start of its uninterrupted transfer over the bus is the *response time* of the message. We don't add the constant non-preemptive time to transfer the message to the response time, because then a response time of zero is always the best possible case for every message independent of its length. This will simplify later comparisons between different schedules. In Fig. 1, for example, the response time of message m_2 is three time slots, because it is delayed by the running message m_1. The *worst case response time* $WCRT_i(b, e)$ of a stream i during a certain time interval starting at time b and ending at time e is the largest response time of all messages of the stream recorded during that time interval. For example, an analytical approach would calculate $WCRT_i(0, \infty)$.

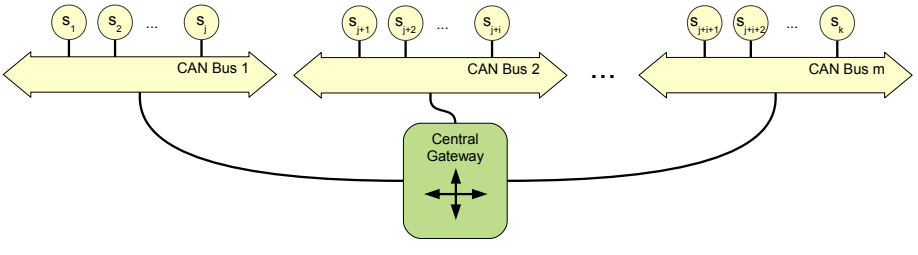

Fig. 2. System model for a multi-segment Controller Area Network

In this paper, we extend this model by allowing m buses connected by a gateway. Each stream is connected to exactly one bus and can send messages to one or multiple buses, as depicted in Figure 2. Therefore, a multi-segment stream s'_i is characterized by a tuple (T_i, O_i, B_i, D_i), that is additionally by a source bus B_i, it is connected to and a set of destination buses D_i. The response times of a multi-segment stream are the times between a message release and the start of its uninterrupted transfer on the destination buses. The worst case response time is, analog to the single-segment case, the largest of these response times. A message that is transmitted on a different bus then the source bus will be called a *routed message*. For the gateway, we make the following assumptions:

- The gateway is central, so it is connected to every bus. It has to be ensured that all connection requirements of the streams are fulfilled. Delays caused by computational load on the gateway are neglected. This is a reasonable assumption, because the operation speed of the gateway is multiples of the communication protocol.
- The gateway uses priority-based transmission with unlimited buffers. This means, in contrast to a first-in-first-out strategy, when several routed messages are pending for transmission, the message with the highest priority will get transmitted.
- A routed message will be pending as soon as the complete message on the source bus has been transmitted.

In our model, we assume discrete time with a minimal system time resolution defined a priori. All stream characteristic times are multiples of this minimal time resolution.

2.3 Rating Approach

In [12] a new metric for comparison of schedules in the form of a rating function is introduced and used to show the advantage of an online approach. We will use this as a basis for comparison. In this paper, we extend the definition by dividing the rating function by the number k of streams that release messages, thus enabling comparisons of schedules for different number of streams. It is defined as follows:

$$r(t) = \frac{\sum_{i=1}^{k} \frac{WCRT_i(t-P,t)}{T_i}}{k} \tag{1}$$

In words, the rating represents the average of all streams maximum response time during the last hyper-period relative to the stream's period. In the case of multi-segment CAN systems, we distinguish between two cases: (1) if the whole system is rated, the WCRTs as described in Section 2.2 are considered and (2) for comparison of a single segment in a multi-segment system, the modification of the rating is explained in Section 4.

3 Single-Segment Scheduling

In our approach, we exploit the dynamic adaptation of message release offsets over time as the traffic on the CAN bus changes. Because it cannot be assumed that there is a single global observer that knows the complete system status and operation, the decisions to change offsets are based on traffic monitoring carried by the individual streams. The algorithm, as well as examples of its operation, are presented in Section 3.1. The robustness to an asynchronous and a changing system are shown in Section 3.3 and 3.4, respectively.

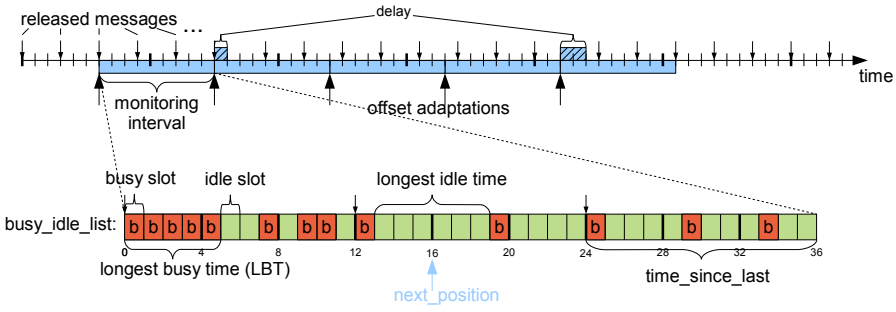

Fig. 3. DynOAA illustration - timing diagram and busy_idle_list on a single node

3.1 DynOAA

The dynamic offset adaptation algorithm (DynOAA) [12] is run on each node independently and periodically. For the sake of completeness, we will illustrate the operation of DynOAA in short as shown in Figure 3. In the upper part of the figure, on the top of the time line, the periodically released messages of the stream are indicated by small arrows. The larger arrows on the bottom of the time line indicate the instances when the adaptations start or when DynOAA is run.

Before the adaptation takes place, the bus is monitored by each stream for a time interval being equal to the maximum period of all streams. A list, from now on called *busy_ idle_ list*, is created. An example of it is shown in the lower part of Figure 3. It contains for each time slot during the *monitoring interval* an idle element if the bus is idle and a busy element if the bus is busy. The length of a time slot is the transmission time of one bit. From the busy_idle_list, we can find the *longest idle time* and *longest busy time* (LBT), which are the maximum continuous intervals when the bus was idle or busy, respectively. The variable *time_ since_ last* denotes the amount of time passed between the current time and the time the last message was released for this particular stream. This value is needed to calculate when the next message would be released. The *next_ position* is the time that indicates when in the next cycle a message should be scheduled. It is chosen in the middle of the longest idle time interval. The next message of the stream is then delayed, i.e., the offset is adjusted, so that a message is released at the time specified by next_position.

In distributed systems, all streams are considered independent from each other. If more than one stream starts to execute the adaptation simultaneously, there is a high probability that the value of *next_ position* at more than one stream will be identical. Instead of spreading, the message release times would in that case be clustered around the same time instance. Therefore, we need to ensure only one stream is adapting its offsets at the same time. Ensuring that only one node is adapting is achieved if all nodes make the decision whether to adapt or not based on a unique criterion based on the same information, the traffic on the bus in this case. The criterion we use is to select the stream

belonging to the first busy slot of the LBT. The idea is that this stream causes the biggest delay, because it potentially could have delayed all subsequent messages in the busy period and therefore should be moved first. If there are more than one LBT of equal length, the first one is chosen. If the monitoring phases of all nodes are synchronized, this mechanism ensures that all nodes elect the same stream for adaptation.

The algorithm is best explained on the example from Figure 3. A stream with a period of 12 time slots (ts) is considered. The transmission of one message is assumed to take one ts. The algorithm passes through two phases. In the first phase, monitoring, the busy-idle list in the lower part of Figure 3 is created. The first message of the LBT belongs to the stream under consideration. Therefore, in the second phase, it will adapt, i.e., the next release will be delayed. In order to calculate the needed delay, the next position is determined first by choosing the middle of the longest idle time, which is in this case 16 ts. The delay is then calculated as:

$$\begin{aligned} delay &= (next\_position + time\_since\_last) \, mod \, period \\ &= (16\,ts + 12\,ts) \, mod \, 12\,ts \\ &= 4\,ts \end{aligned}$$

The delay causes the move of the releases of this stream in the next hyper period by additional 4 ts. For the messages at positions 1 to 4 this means, if they were delayed, then their response time is reduced.

3.2 Evaluation

In order to evaluate the quality of our approach, we developed and used our own CAN bus simulator. The reason for this was that no available simulator can describe scenarios we needed and extract the required properties. Our simulator is event-driven with the simulation step size being equal to the transmission time of one CAN bit. The full CAN protocol is reproduced by assuming worst-case bit-stuffing. We can only simulate the synchronous case, where all nodes have the same time base. This means if the offsets are fixed, the schedule repeats after time equal to the hyper-period. The asynchronous case is simulated by using different random initial offsets. The simulation assumes the error-free case. Our simulation engine is comparable to RTaW-Sim[11]. Although RTaW-Sim offers more functionality, such as error insertion and clock drift, and runs much faster, it does not provide provisions to change offsets at run-time that is needed to test DynOAA.

The scenarios used for our experiments consist of synthetic scenarios automatically generated by Netcarbench [3] and are typical for the automotive domain with a bus load that can be freely adjusted. A bus speed of 125 kbit/s is assumed. Typical for practical implementations is that there are not to many different periods (e.g., 5 - 10) that are mostly multiples of each other. This results in a small hyper period. In the example scenarios, it is always 2 seconds, which is also the largest period.

Figure 6 shows the rating function of Eq. 1 over time for different scenarios. The experiments were always run for 10 different random offset initializations. The continuous lines in the plots represent the average of these 10 runs, while the vertical error bars indicate the maximum and minimum value of the rating function at that instance of time. We can see that it converges very fast to a stable value. The plot also shows that we are always improving significantly compared to the non-adaptive case which is represented by the rating values at time zero.

3.3 Asynchronous Monitoring

In Section 3.2, we assumed the monitoring intervals were synchronized. However, in an distributed system, the monitoring intervals of the streams are asynchronous, so choosing the adapting stream gets more difficult. We cannot guarantee anymore that exactly one stream is adapting, as is shown in the example in Figure 4. In the example, the monitoring phase of the two streams overlap in such a way that the longest idle time is the same, but the longest busy period is different. Therefore, streams 1 and 2 would schedule their messages simultaneously. Nevertheless, given the monitoring phases have the same length, the maximal number of streams scheduled to the same idle time is two. This means the effect is not severe as the experiments in the following text will show.

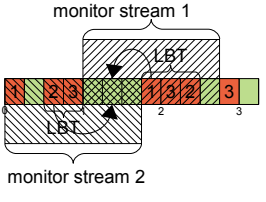

Fig. 4. Example where asynchronous adaptation leads to scheduling of two stream to the same position

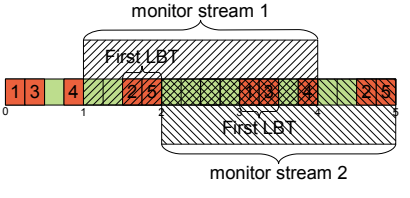

Fig. 5. Example where the adaptation stops because no stream is first of the first longest busy period

Another problem with asynchronous monitoring occurs when there are several longest busy periods with the same length. If the streams always choose the first LBT, it can lead to a stop of adaptation, because no stream will adapt. How this can happen is shown in the example in Figure 5. On the one hand, this problem can be simply avoided by choosing randomly the LBT that is considered. On the other hand, if the monitoring periods are synchronous, the random selection can cause that several streams adapt simultaneously to the same position.

In Figure 7, the rating over time for a scenario with 50% load is depicted. The meaning of the error bars is similar to the one described for Figure 6. Table 1 shows the average rating values for cases from Figure 7. The results for

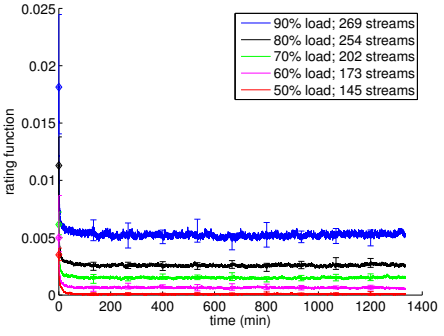

Fig. 6. Rating function as a function of time for different application and load scenarios

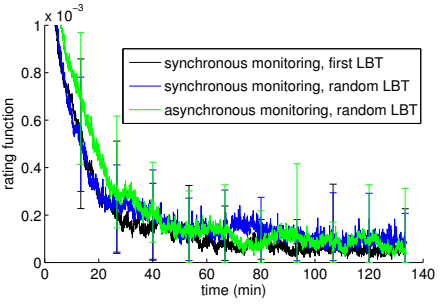

Fig. 7. Rating function as a function of time, comparing synchronous to asynchronous monitoring and choosing the first LBT to choosing a random LBT for deciding which stream adapts

asynchronous monitoring, and choosing which stream to adapt based on a random LBT are not displayed, because, as described above, the adaptation in the simulation gets stuck. As expected, the setup with synchronous monitoring and choosing which stream to adapt based on the first LBT performs best. However, the difference in performance is negligible considering the variation within the results. Results with other scenarios, not shown here, reveal a similar behavior, leading to the conclusion that DynOAA also works well for asynchronous monitoring, which would be the case in a real distributed system. The experiments in the rest of the paper are always run with asynchronous monitoring and random LBT.

Table 1. Average rating value over the whole simulation time for the scenario in Figure 7

synchronous monitoring, first LBT	$2.0011 \cdot 10^{-4}$
synchronous monitoring, random LBT	$2.3788 \cdot 10^{-4}$
asynchronous monitoring, random LBT	$2.5792 \cdot 10^{-4}$

3.4 Adaptation in Dynamically Changing Systems

In this work, we model the first time the dynamic changes of the system configuration by allowing addition and removal of streams during system operation. At this stage, only preliminary simulation runs are performed which indicate behavior of the system when the configuration of the system changes. We start

with the system operation with a fixed number of streams. When a stream is removed from the system, the response times of the remaining streams will either decrease or stay the same, depending on whether the removed stream delayed the transmission of another stream or not. Similarly, when a stream is added, the response times of the other streams increases or stays the same depending on whether the added stream delays another stream or not. The goal of the adaptation, keeping the response times as short as possible, is achieved by adapting the offsets dynamically and finding a new set of offsets for the current scenario that does not depend on the set of offsets for the case before the change of system configuration. We demonstrate this behavior by running DynOAA for different initial offset assignments and comparing the rating value after the system has converged. The results of experiments in Figure 6 (see Section 3.2) show exactly that, because the system converges to a certain rating value depending on the used scenario and not on the initial offset assignment.

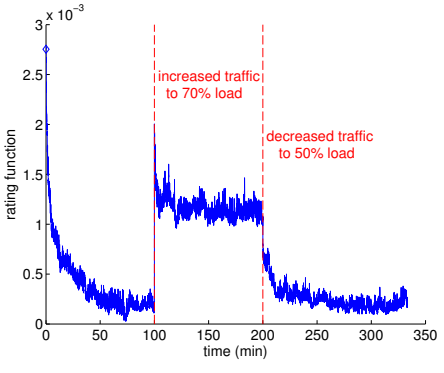

Fig. 8. Rating function as a function of time for a scenario in which at time 100 min, 64 streams are added such that the load increased from 50% to 70%. At time 200 min, these streams are removed again.

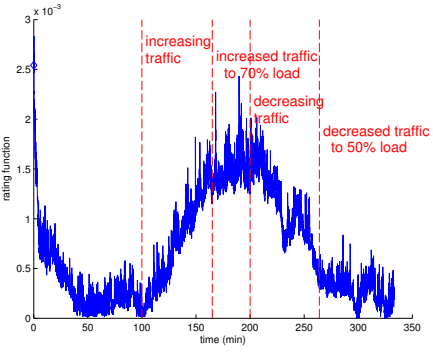

Fig. 9. Rating function as a function of time for a scenario in which at time 100 min, 64 streams are added one every minute such that at time 164 min, the load is increased from 50% to 70%. At time 200 min, these streams are removed again one every minute.

Because the adaptation is constantly performed, the rating value fluctuates even when the system is not changing configuration. Therefore, the reaction on adding or removing one stream will not be obvious. However, we can show the system reaction when several streams are added or removed. In Figure 8, we show the scenario in which 64 streams are abruptly added to the system at time 100 min, and then removed at time 200 min. Corresponding system load increased from 50% to 70% and back to 50%, respectively. Figure 9 illustrates the system behavior when the change was not abrupt, but rather gradual. From time 100 min, 64 streams are added to the system at the rate of one stream per minute,

and then after time 200 min the streams are removed from the system, one stream per minute. It is obvious that the DynOAA adapts offsets and stabilizes the value of the rating function in each case for each new configuration. Further analysis of dynamic system configuration changes is part of our future work.

4 Multi-segment Scheduling

Data rate limitations of CAN can be overcome in some applications by using multi-segment systems. In this paper, we present the case of the system where the segments are connected through a single gateway. Firstly, multiple segments can help to increase the capacity of priority-based communicating embedded systems. Secondly, even though DynOAA reduces the message response times to a minimum for a given load, the multi-segment approach can potentially reduce the load of the individual segments and thus reduce the message response times further. We consider a simple case where a single segment is split into up to five segments. Gains can be expected if the inter-segment communication is not too intensive. Finally, the reason for a multi-segment approach can be purely practical. For example, in the development of large distributed systems, which consist of multiple application domains, the systems are designed by multiple teams whose efforts cluster around individual domains implemented on single-segment system. The integration of the overall system becomes easier if it only requires connection of multiple segments via a gateway.

4.1 Partial Adaptation

An obstacle for applying DynOAA to multi-segment systems is that the current traffic information for the whole system is no longer available to each node, because a certain fraction of messages is routed by the gateway. This means that the nodes change their offsets based only on information on local traffic of the segment they are connected to. Section 4.2 will demonstrate the influence of this on the system performance.

Another problem in multi-segment case is that it is not possible to influence the release of all messages on a segment directly. For example, messages received from the streams of the other bus segments are not aware of the traffic situation on the receiving bus segment and cannot perform correct adaptation. The knowledge of the full traffic situation would be possible by using the gateway to collect the traffic information and transmit (broadcast) it to all nodes on all segments, but it would result in additional overhead and require additional bandwidth. Therefore, this option is left out in our work.

Based on the above analysis, our decision was to first analyze the case in which only a part of the streams, belonging to the messages on one segment, perform DynOAA and adapt their offsets, while the others are not doing that, and we call this partial adaptation. This type of behavior can then be extrapolated to the multi-segment systems directly. The problem with the partial adaptation is that the DynOAA can select a stream that does not adapt as one that should adapt,

and the intended goals would not be achieved. In the case of our synchronous simulation, it even can lead to complete stop of the adaptation. However, a slight modification of the DynOAA corrects the algorithm operation and makes it suitable for the partial adaptation case. The solution is outlined in the following paragraphs.

During traffic monitoring, in addition to the information whether a slot is busy or idle, we record the information whether the busy slot is allocated by an adapting or non-adapting stream in the *annotated busy_idle_list*. This information can be provided by using an indicator (flag) associated with the current message with two possible values, adapting or non-adapting. This indicator could be statically assigned to each message stream at design time, but the flexibility of dynamic adaptation and self-organization would be lost in that case. Therefore, we propose to use the last (least significant) bit of the CAN identifier to assign a priority to the stream to indicate whether it is adapting or not. This has the additional advantage that there is no need to keep the list of all streams and at the same time allows flexible insertion and removal of nodes from the system. The penalty for the adopted approach is that the number of available identifiers is halved. In our opinion, this is not a big disadvantage because the number of still available identifiers is large enough to cover the needs of real distributed systems. The number of available identifiers becomes $2032/2 = 1016$, which is large enough to represent real-life systems. For example, in the current VW Golf VI (2010 model), 49 nodes with an average of 140 streams are used [10]. In addition, due to the limited capacity of the CAN bus, the number of needed identifiers will most probably not grow.

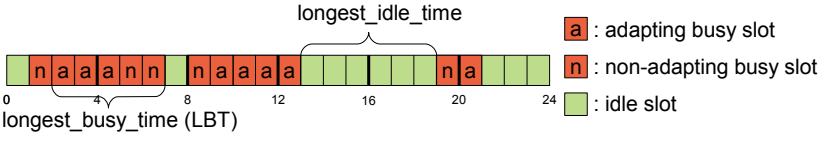

Fig. 10. Example of an annotated busy_idle_list, storing for each time slot during monitoring whether it is idle or busy by an adapting or non-adapting stream

From the annotated busy_idle_list, each node again calculates the longest idle time and longest busy time (LBT). The longest idle time is determined similar to the description given in Section 3.1. As for the LBT, we choose the maximum continuous interval in which the bus was busy starting with an adapting busy slot. This way we ensure that the first slot of the LBT belongs to an adapting stream and, therefore, the adaptation is done. This stream also represents the adapting stream that potentially delays most other streams. In the example in Figure 10, even though the second busy time has four adapting busy slots, the first slot of the first busy time, i.e., here LBT, potentially delays four time slots, while the first one of the second busy time only delays three slots. Therefore, it is better to adapt the first, as it is chosen by the algorithm. In

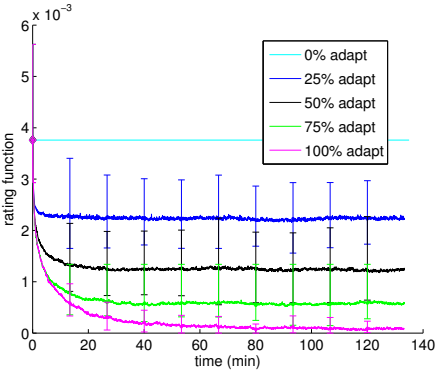

Fig. 11. Rating function as a function of time for different fractions of streams adapting for a scenario with 50% load

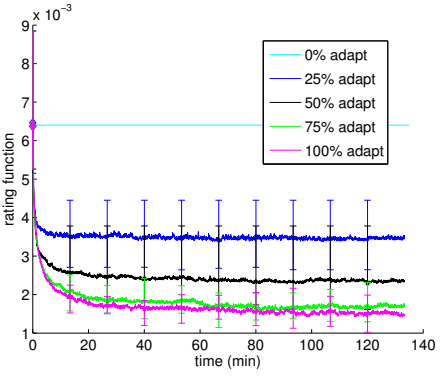

Fig. 12. Rating function as a function of time for different fractions of streams adapting for a scenario with 70% load

Figure 11 and 12, it is shown how the enhanced DynOAA performs for different numbers of adapting streams (and nodes) expressed as a fraction of total number of streams. The experiments carried out for this case are similar to the experiments from Figure 4. For the 50% load scenario, if none of the streams are adapting, the rating value in average is constant at 0.00376. If 50% of the streams are adapting, the rating value converges to an average of 0.00124. This is less than a half of the value of for the non-adapting ones:

$$0.00124 < \frac{0.00376}{2} = 0.00188.$$

For the 70% load scenario, this observation is strengthened further. The performance difference between all (100%) adapting and 75% adapting streams is very small. In addition, the experiments show that the time needed to converge to a stable state depends on the number of streams that are adapting. The converged state is reached faster if fewer streams are adapting.

4.2 Multi-segment Systems

The conditions to run DynOAA on multi-segment systems, by allowing some messages no to adapt, have been outlined and established in Section 4.1. In the multi-segment system case, the routed messages are simply considered as non-adapting messages. In order to evaluate the performance of DynOAA for multi-segment networks, we first generated single-segment scenarios by Netcarbench [2] and then provided additional information on the source and destination segments for each message stream, given a number of segments in the system.

We performed a number of experiments to analyze the performance of multi-segment systems using our rating function in Eq. 1. All the results should be considered as preliminary and a work in progress. These experiments were performed under certain conditions and assumptions which are outlined below. We

first assumed that the source segment of each stream is assigned by uniformly distributing all the streams to the available segments. The destination segment is characterized by two parameters, *percentage of routed* and *percentage of received segments*. The first parameter specifies the percentage of streams that are routed at all, i.e., whether the destination bus is different than the source bus. The second parameter specifies the percentage of segments, from all available segments, which are included in the destination segment set. Which stream is routed and which segment is chosen as the destination is determined randomly, excluding the case when the source and destination segment are the same. For example, for the case of eight streams in the system with four segments with 50% routed and 25% received messages, two streams are assigned to each segment. One of these two streams has one other segment as its destination.

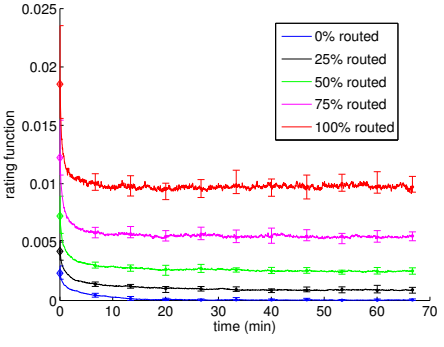

Fig. 13. Rating function as a function of time for a scenario with two segments with different fractions of streams routed to the other segments

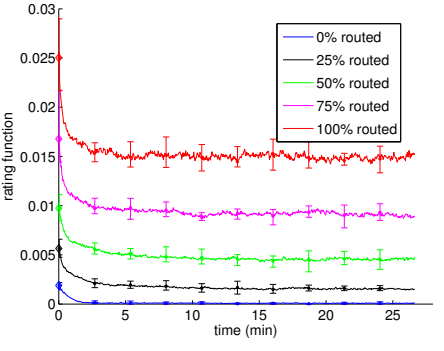

Fig. 14. Rating function as a function of time for a scenario with 4 segments with different fractions of streams routed to the other segments

For the experiments presented in this section, the streams of the scenario with 80% load from Section 3.2 are used. The received parameter is set to 50%. In Figure 13 and 14, it is shown how the enhanced DynOAA performs for different multi-segment scenarios. The experiments carried out for this case are similar to the experiments presented in Figure 4. These experiments show that when using DynOAA for the multi-segment systems, the rating value decreases,i.e., performance improves compared to random chosen values, as denoted by the value at time zero.

The experiments shown in Figure 15 are done under assumption of the same workload on each segment, because all streams from the initial set at every segment are either directly scheduled or are routed streams. First, it has to be noted that the rating value is increasing with the number of segments for DynOAA at the converged state, as well as for randomly set offsets at time zero. Also, it can be noted that the decrease of the rating value is most likely due to the increased routing and not the result of degradation of performance of

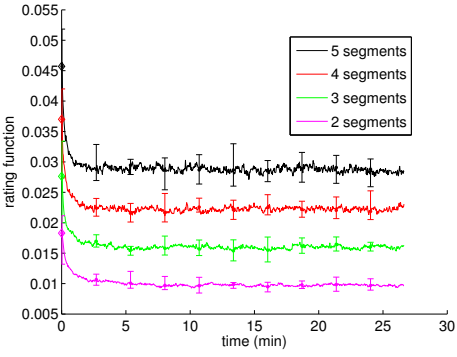

Fig. 15. Rating function as a function of time for scenarios with 2 to 5 segments with 100% routed and 100% received

DynOAA, which uses information local to the segment on which it performs. However, more experiments are needed in future to validate these assumptions.

5 Conclusions and Future Work

Self-organization through dynamic adaptation of message scheduling is a promising way of giving a new life to the existing communication buses used for distributed embedded systems in applications such as automotive or industrial control. We specifically target communication buses, which use priority-based scheduling schemes. The main idea of adapting message offsets in bus-based systems and avoidance of conflicts by simultaneous release of multiple messages has been refined into the algorithm for dynamic offset adaptation, DynOAA. The algorithm is first developed and analyzed for single-bus systems in which all streams adapt the offsets dynamically, and then further enhanced for the case when only certain nodes are adapting. It is shown that by allowing partial adaptation it is possible to use DynOAA also for multi-segment systems in which all segments, consisting of one bus, are homogeneous (the same bus type) and use a single central gateway for their interconnection. The results of experiments run by using a newly developed bit-accurate simulator that incorporates DynOAA and its variations show significant improvement of performance over existing, static scheduling approaches. In particular, the proposed approach results in significantly shorter response times of messages and allows therefore in consequence a much higher utilization of the bus (in our case CAN bus), thus extending applicability of the existing bus technology. The price paid for adaptation is relatively small additional work that has to be performed on the side of system nodes. Our future work includes a closer analysis of the stability of the adaptation process, implementation and its analysis of DynOAA on real microprocessor platforms and further extension of the multi-segment approach with a more realistic model of the gateway.

References

1. CAN Specification 2.0 B. Robert Bosch GmbH, Stuttgart, Germany (1991)
2. Bai, T., Hu, L., Wu, Z., Yang, G.: Flexible fuzzy priority scheduling of the CAN bus. Asian Journal of Control 7(4), 401–413 (2005)
3. Braun, C., Havet, L., Navet, N.: NETCARBENCH: A benchmark for techniques and tools used in the design of automotive communication systems. In: 7th IFAC International Conference on Fieldbuses and Networks in Industrial and Embedded Systems. Citeseer (2007)
4. Di Natale, M.: Scheduling the can bus with earliest deadline techniques. In: Proceedings of the 21st IEEE Real-Time Systems Symposium, pp. 259–268 (2000)
5. Felser, M.: Real-time ethernet - industry prospective. Proceedings of the IEEE 93(6), 1118–1129 (2005)
6. FlexRay Consortium. FlexRay Communications Systems - Protocol Specification v3.0 (2009), `http://www.flexray.com`
7. Goossens, J.: Scheduling of offset free systems. Real-Time Systems 24(2), 239–258 (2003)
8. Grenier, M., Havet, L., Navet, N.: Pushing the limits of CAN-scheduling frames with offsets provides a major performance boost. In: Proc. of the 4th European Congress Embedded Real Time Software (ERTS 2008). Citeseer, Toulouse (2008)
9. Pfeiffer, O., Ayre, A., Keydel, C.: Embedded networking with CAN and CANopen. Copperhill Media (2008)
10. Racu, R.: The role of timing analysis in automotive network design. In: Talk, 4th Symtavision News Conference on Timing Analysis, Braunschweig, Germany (2010)
11. RTaW-Sim. Real-time at Work CAN Simulator, `http://www.realtimeatwork.com/`
12. Ziermann, T., Salcic, Z., Teich, J.: DynOAA - Dynamic Offset Adaptation Algorithm for Improving Response Times of CAN Systems. In: Proceedings of Design, Automation and Test in Europe (DATE 2011), March 14-18. IEEE Computer Society, Grenoble (2011)
13. Zuberi, K.M., Shin, K.G.: Non-preemptive scheduling of messages on controller area network for real-time control applications. In: Rtas, p. 240. IEEE Computer Society, Los Alamitos (1995)