Scheduling of the FlexRAY communication protocol respecting AUTOSAR FlexRay COM stack

Zdeněk Hanzálek, David Beneš
Department of Control Engineering
FEE, Czech Technical University in Prague
Prague, Czech Republic
Email: {hanzalek,benesda1}@fel.cvut.cz

Abstract

The objective of the paper is the formulation and solution of the frame packing and scheduling problem for the static segment with respect to AUTOSAR FlexRay COM stack. There are several other works on the static segment scheduling. The problem becomes more constrained since the release dates and deadlines of the signals are taken into account. While assuming release dates and deadlines to be multiples of the cycle length, a solution of the scheduling problem is based on decomposition to separate nodes. The article shows simple and efficient scheduling algorithm, its evaluation on benchmarks with up to 3000 signals per node and simple experiment on real FlexRay HW.

Index Terms

scheduling, FlexRay, time-triggered communication protocols, heuristic, ILP

I. Introduction

The FlexRay [5] protocol seems to establish a new standard in automotive and together with recent automotive protocols such a CAN, MOST and LIN will form a robust platform for modern cars with x-by-wire systems. BMW, the pioneer in innovative technology, uses the new protocol in their latest model for transmission of large quantities of data within the active chassis system.

Communication protocol can offer different number of the static slots based on its configuration. Nowadays, luxury cars contains up to 70 electronic control units (ECU), which require transmission up to 2500 different signals [1]. Hence, it is essential to develop a powerful algorithm for a frame packaging and scheduling with user defined constrains (signal period, release date, and deadline). In this paper, time window is we considered for each signal given by release date and deadline related to the start of the hyperperiod.

II. FlexRay overview

The FlexRay is a time-triggered communication protocol where the data transmission is based on recurring communication cycle which is composed of static segment, dynamic segment, symbol window and network idle time. The example of communication cycle is depicted in Figure 1.

The static segment comprises a fixed number of equally sized time division multiple access (TDMA) static slots. The segment is designed for frames where predictable frame delay is defined. In each static slot a frame (with a unique identifier frameID) can be transmitted by a node. Once a static slot with a unique identifier is allocated to a specific node, only this node is allowed to transmit during this slot and so protocol ensures collision avoidance. In order to ensure proper slot timing, each single node provides synchronization by synchronization frames - frame where synchronization bit is set. Synchronization is embedded in the standard and is based on differences between real and predicted frame receptions.

The dynamic segment employs the flexible TDMA approach and is intended for the non-critical and sporadic messages with varying length. The symbol window is designed for transmission of protocol symbols used for network wake-up, start-up and testing. The part of timebase correction is performed during the network idle time. The dynamic segment and symbol window are optional part of the communication cycle. The frame scheduling on the dynamic segment (see [11], [10]) is out of the scope of this study.

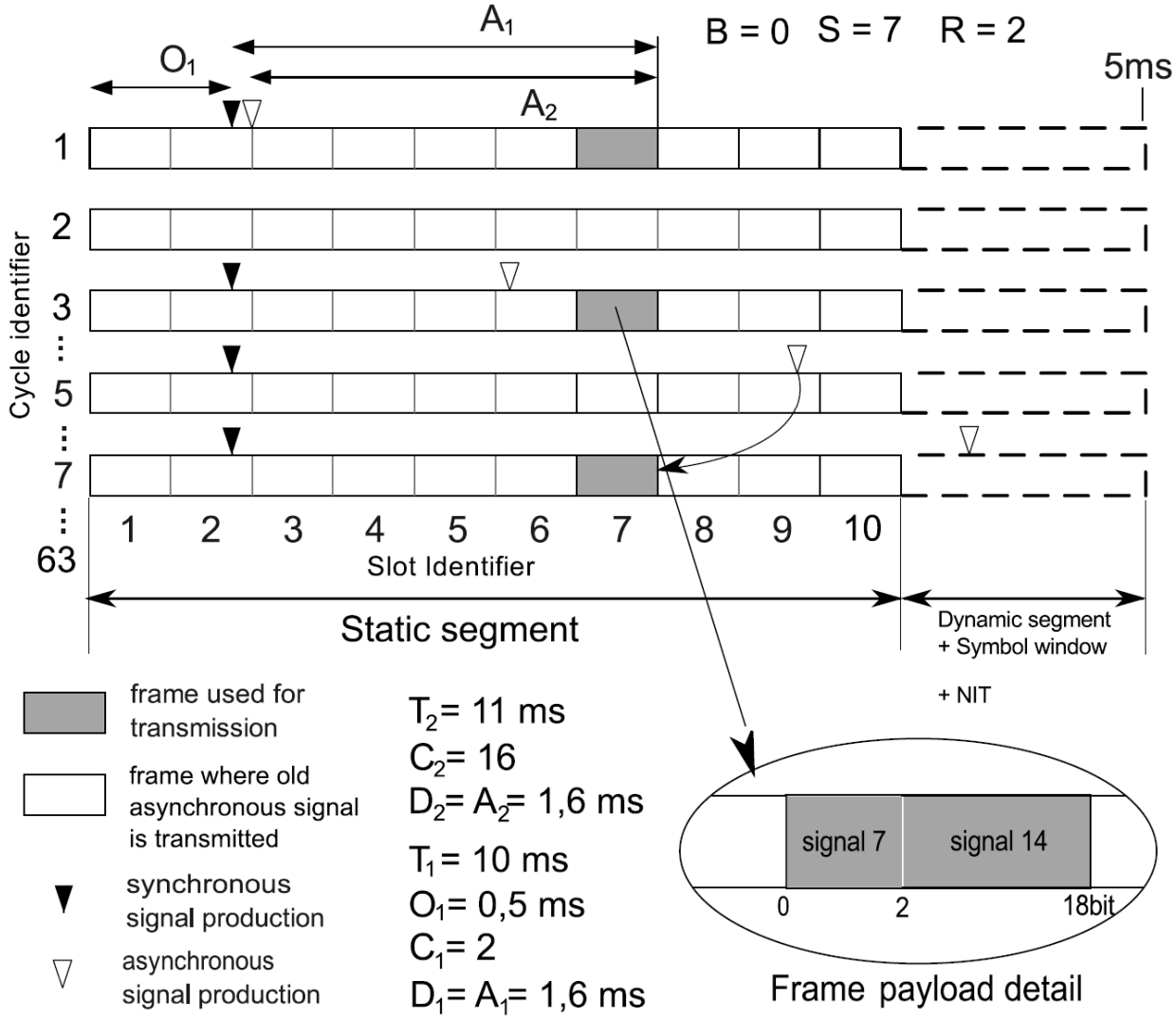

Figure 1. Illustration of generation of synchronous and asynchronous signals

The frame includes previously mentioned frameID which matches to slot identifier in the static segment and the cycle number identification. There are 64 different cycles and up to 2047 static slots. The FlexRay network (cluster) can be arranged in different physical topologies such as a star, a bus, a point-to-point or a hybrid topology. Two independent physical channels should be preferably used for high bitrate or enhanced safety. Every single channel can form a different physical topology. In this paper, it is assumed, that the same data are transmitted on both channels.

III. Related Work

The goal of the paper is the formulation and solution of the frame packing and scheduling problem for the static segment with respect to AUTOSAR FlexRay COM stack. There are several other works on the static segment scheduling. Techniques for determining the timing properties of messages transmitted in both the static and the dynamic segments of a FlexRay communication cycle are proposed in [10]. The analysis techniques for messages are integrated in the context of a holistic schedulability analysis that computes the worst-case response times of all the tasks and messages in the system. Authors present and evaluate three optimization algorithms that can be used to improve the schedulability of a system that uses FlexRay. Unfortunately, authors assume knowledge of all signals in system for bandwidth optimization by modification of protocol parameters such as number of static slots, static and dynamic segment duration. The assumption is quite limiting for automotive industry applications where an incremental design is commonly used.

A mathematical model determining the optimal length of static messages that can achieve more efficient use of a FlexRay is presented in [13]. However, knowledge of all signals in a system is once again required. Moreover, the longest high priority frames are reallocated to the dynamic segment and thus system have to be redesigned with each extension; otherwise it cannot be guaranteed that frames in dynamic segment meet deadlines.

The message frame packing that maximizes network efficiency by reducing unused slots is investigated in [12]. Signals are packed into the same size messages to obey the restrictions of the FlexRay protocol, while narrow bandwidth is used. Then

authors formulate a nonlinear integer programming problem to compute an optimal message set from given set of signals. They also proposed a scheduling method with low jitter and minimum number of static slots to improve the network efficiency. In contrast to [12], the release dates and deadlines of the signals are taken into consideration in this paper, so the problem becomes more constrained.

The authors of [6] assume usage of AUTOSAR COM stack and use heuristic analysis to schedule relatively large message set. The authors assume that frame packing is done by the application level.

An optimization framework that includes not only signal to frame packing, but also frame to slot assignment, task schedule and the synchronization of signal and task scheduling considering end-to-end delays and the precedence constraints induced by information passing between tasks and signals is presented in [18]. Presented mixed-integer linear programming formulation includes the system-level schedule optimization with the definition of an optimal relative phase in the activation of tasks and signals which accounts for deadlines and precedence constraints. Moreover, usage of AUTOSAR standard is assumed, but framework is evaluated only on typical X-by-wire application.

To be able to schedule large CAN message database [14] propose to schedule signals separately for each node. [15] highlight the significance of frame packing. They also schedule signals for switched FlexRay networks, where can be multiple senders simultaneously. Achieving required reliability by retransmission in cases with possible failed transmission is considered in schedules in [17], [16].

IV. Problem statement

Two sets of tasks executed in ECUs are to be considered: the first set contains tasks that are running without synchronization with communication cycle. Signals (i.e. communication of one state variable from one ECU) generated by these tasks are called asynchronous signals. Signals provided by the second set of tasks which are synchronized with communication cycle are called synchronous signals. The objective is to find packing of signals into frames and allocation of each frame to specific slot and cycle while minimizing the length of the schedule (i.e. the length of the static segment).

FlexRay network configuration consists of large parameter set including a cycle length, number of static slots in the static segment, duration of static slot, duration of NIT segment etc. These parameters are usually predefined by system designer and further followed by ECU suppliers. The following setting based on the BMW network design [2] is used in this paper. The communication cycle duration F is equal to 5 ms where static segment takes 3 ms and the rest of the communication cycle is filled by the dynamic segment, symbol window and network idle time. There are M=75 static slots with static slot duration L of 0.04 ms. The frame payload W is 128 bits.

Each signal has defined the following parameters (the same denotation as [6] is used):

- N_i unique identifier of ECU, which transmits signal i,
- T_i the signal period, the signal is assumed to be transmitted only once in the FlexRay cycle,
- O_i the signal release date, it is the latest time after which the first occurrence of the signal is produced in relation to the start of the period,
- C_i signal size in bits,
- A_i the signal age is the duration between the signal production at the sender side and the moment, when the first frame carrying the signal is received.
- D_i the deadline associated to the signal i, it represents the maximum age acceptable at the consumer end.

Next assumption is usage of AUTOSAR communication stack [9] which is de facto standard for automotive applications and it is considered also for other industry applications such as aviation. In this context, the AUTOSAR frame is defined by

- Q_j unique ECU identifier, which transmits the frame j,
- S_j static slot identifier used for frame j,
- R_j the AUTOSAR frame repetition, which is the transmission period of frame expressed as a multiple of the communication cycle with the constrains that the repetition takes its value in $\{2^n | n \in N, n \leq 6\}$,
- B_j the base cycle identifier - the first occurrence of the frame j. $B_j \leq R_j$.

Each single frame j is thus characterized by a tuplet $\{Q_j, S_j, B_j, R_j\}$. The frame contains processing data units (PDU) where each consists of one or more signals. An update bit can be defined for each PDU. It informs recipients whether at least one signal in the PDU was updated or not. The signal can be contained in more than one PDU. In AUTOSAR context is defined that scheduling is static and cannot be changed during runtime.

It is further assumed, that maximal signal size cannot exceed data payload of the frame. It means that signal cannot be split into two or more frames. Message fragmentation is ensured by AUTOSAR FlexRay transport protocol layer [8].

By oversampling, the asynchronous signal is transformed to synchronous one with the following parameters

$$\begin{aligned} D_i = T_i = 2 \cdot n \cdot F + P + U \\ O_i = 0 \end{aligned} \tag{1}$$

where P is packing time - time from task request on signal i transmission to the start of frame transmission and U is unpacking time.

Period	# signals
7	26
8	51
10	143
20	205
40	7
50	51

Period	# signals
100	272
200	117
250	22
400	2
500	265
1000	77

Table I
MESSAGE SET USED IN PRODUCTION CAR

Synchronous signals can keep deadline shorter than a period. In this case deadline is limited by the following values

$$D_i^S \geq \begin{cases} P + L + U & \text{for} \\ & 0 \leq O_i – n \cdot F \leq P + M \cdot L + U \\ P + L + G + U & \text{otherwise} \end{cases} \tag{2}$$

where $n \in 1, 2, 3, ..., 64$, M is number of static slots, L is duration of the static slot, and G is duration of dynamic segment, network idle time and symbolic window.

The Figure 1 depicts a situation where two signals are generated – synchronous one and asynchronous one. The period of synchronous is set to 10 ms and 11 ms for asynchronous. Let us assume that, both signals have the deadline equal to the period. In order to fulfill the deadline constraint, it is necessary to transmit both signals every second cycle, it means $R_j = 2^1$. In the fourth cycle ECU transmits the same data twice due to oversampling of the asynchronous signal. However, this behavior is covered by previously mentioned *update* bit.

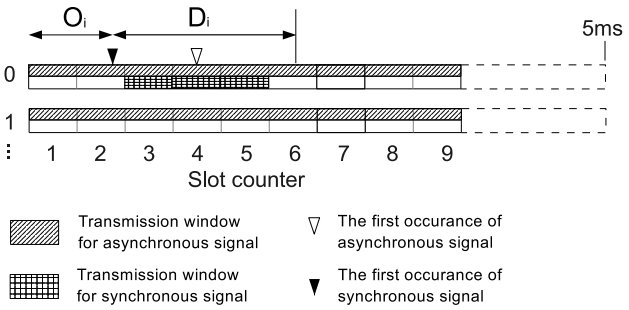

Figure 2. Illustration of the transmission window

A. Signal set

The scheduling approaches are applied to the modified Society of Automotive Engineers (SAE) benchmark signal set.

The SAE report describes a set of signals sent between seven different subsystems in an electric car prototype. The Basic SAE signal set defines 53 sporadic and periodic signals. A periodic message has a fixed period 5, 10, 100, 1000 ms, and implicitly requires the latency to be less than or equal to this period. Sporadic messages have latency requirements imposed by the application: for example, all messages sent as a result of a driver action have a latency requirement of 20 ms. The reader is referred to the work of Kopetz [7] for more detailed benchmark description. Due to the high FlexRay bandwidth (10Mb/s) and electronic equipment's requirements in today's cars we extended the SEA benchmark using NETCARBENCH [3] to:

- increase the number of signals while using the same probability distribution function of parameters - up to 3000 signals per node are considered in this article,
- add parameters (such as release date).

Signal set of fully equipped lower class car is shown in Table I. Signals are exchanged among 33 nodes connected into two network with gateway. Several signals are sent only on change event where maximum deadline is defined. These signals are transformed to periodic signals with periods equal to deadlines.

B. Time constraints

In the first step, it is needed to find each signal pair $\{E_i, V_i\}$ where E is earliest and V latest slot which can be used for transmission of the first occurrence of the signal i. The pair is given by

$$E_i = floor\left(\frac{P+O_i}{F}\right)\cdot M + min\left(ceil\left(\frac{\left(P+O_i-floor\left(\frac{P+O_i}{F}\right)\cdot F\right)}{L}\right), M\right) \tag{3}$$

$$V_i = floor\left(\frac{P+O_i}{F}\right)\cdot M + min\left(floor\left(\frac{\left(P+O_i+D_i-floor\left(\frac{P+O_i}{F}\right)\cdot F\right)}{L}\right), M\right) \tag{4}$$

The Figure 2 shows graphical representation of pair $\{E_i, V_i\}$ for our two signals where we defined stricter deadline for synchronous signal.

In order to obtain an optimal solution, we used the Constraint Programming to formulate the scheduling problem given by C, T, E and V. Due to the combinatorial complexity of the problem we have been able to solve only very small problems (like 15 signals on 4 nodes) even though we have developed several versions using advanced scheduling features of ILOG OPL Studio. In order to simplify the problem, we consider $\{\tilde{O}_i, \tilde{D}_i\}$, earliest and latest cycle to transmit signal i, instead of $\{E_i, V_i\}$. This simplification is adequate since the precise specification of $\{E_i, V_i\}$ has an influence only, when these values fall in the static segment. But in the dynamic segment, they are rounded to the length of the cycle anyway.

For this reason, we express $\{\tilde{O}_i, \tilde{D}_i\}$ as follows:

$$\tilde{O}_i = ceil\left(\frac{E_i}{M}\right) \tag{5}$$

$$\tilde{D}_i = floor\left(\frac{V_i}{M}\right) \tag{6}$$

This simplifies our scenario, since release dates and deadlines could be only multiples of the cycle length, because a position of a signal within the static segment of particular cycle is insignificant with respect to the position of the signal within hyperperiod. This effectively means, that we don't care about assignment of particular slot ID to the node, therefore it is not important, whether the node uses the first slot ID, or the last one. Consequently, we can decompose our scheduling problem as follows: signals of a node can be scheduled separately from the signals of other nodes, since each node has separate slot IDs and there is no competition for slot ID with respect to the release date and deadline, since these values are rounded to the length of the cycle. On the other hand, due to the rounding, we cannot influence a position of the signal in the static slot and we may loose some feasible solution due to the $ceil$ and $floor$ operators.

V. Algorithm

The Time Constrained FlexRay static segment Scheduling (TCFS) algorithm (shown in Algorithm 1) consists of the three principal phases: frame packing, message aggregation and schedule creation. These phases are executed separately for each node. Result of the algorithm is given given by structure $schedule_k$, which contains the following fields for each node $k \in \{1, ..., n\}$: $slot\_ID$, $cycle\_ID$, $position\_in\_frame$ for each signal send by node k and number of slots ($\#slots$) occupied by this node.

Algorithm 1 Pseudo code of the TCFS algorithm

Input: C, T, E, V, N
Output: $schedule_1, ..., schedule_n$

$[signals_1, ..., signals_n]$ = divide_signals(C, T, E, V, N);
for (k = 1 ... $\#nodes$)
    $msgs_k$ = frame_packing($signals_k$);
    $agrmsgs_k$ = aggregate_messages($msgs_k$);
    $schedule_k$ = create_schedule($agrmsgs_k$);
if ($\sum_{k=1..n} schedule_k.\#slots > M$)
    error('Schedule infeasible');

First of all, the signals are divided to nodes, which is performed in linear time complexity. Structure $signals_k$ contains C, T, E, V parameters of signals send by node k (see Table II with $signals_7$). Since the release dates and deadlines are multiples of the cycle length, it is not needed to decide which node should obtain lover or higher slot IDs and it is possible to compute the schedule of each node separately (this may be done conveniently in parallel while using several CPUs). Final schedule is obtained just by putting individual schedules consequently one after the other. Schedule is infeasible, when the number of required slots is higher than M, the number of static slots.

A. *Frame packing*

Frame packing is done separately for each period of signals (following the idea of [12], which solves the problem without release dates by an Integer Linear Programming). Frame packing groups signals into messages that are not larger than the slot and have all properties of signals, it has the same period T_i as signals packed into this message, message size C_i is a sum of the sizes of signals, the message release date $\tilde{O}_i$ equal to the latest release date of signals and the message deadline $\tilde{D}_i$ equal to the earliest deadline of signals.

	C_i	T_i	$\tilde{O}_i$	$\tilde{D}_i$
s_1	26	2	0	2
s_2	2	1	0	1
s_3	2	4	0	11
s_4	6	4	0	13
s_5	6	8	1	8
s_6	8	1	0	1
s_7	2	2	0	2
s_8	4	1	0	1
s_9	32	8	5	8
s_{10}	16	2	1	2

	C_i	T_i	$\tilde{O}_i$	$\tilde{D}_i$
s_{11}	4	4	0	9
s_{12}	14	1	0	1
s_{13}	4	1	0	1
s_{14}	16	2	0	1
s_{15}	10	16	1	6
s_{16}	8	2	0	2
s_{17}	4	2	0	2
s_{18}	2	8	3	7
s_{19}	14	16	1	16
s_{20}	20	1	0	1

Table II
SET OF SIGNALS ON THE FRAME PACKING INPUT FOR ONE NODE (I.E. VARIABLE $signals_7$)

The message simply groups the signals to simplify scheduling while reducing the number of elements that need to be scheduled. When considering, whether given *signal fits* in the message, only the messages with the same period and enough of the free space are taken into account (i.e. actual size of the message plus the size of the signal could not be larger than the size of the slot). Furthermore, when the signal is *inserted* in the message, the time constraints of the message are *adjusted* (i.e. the release date of the message must be equal or larger than the release date of the signal and the message deadline must be equal or shorter than the signal deadline). Two different strategies have been adopted to choose appropriate message for given signal.

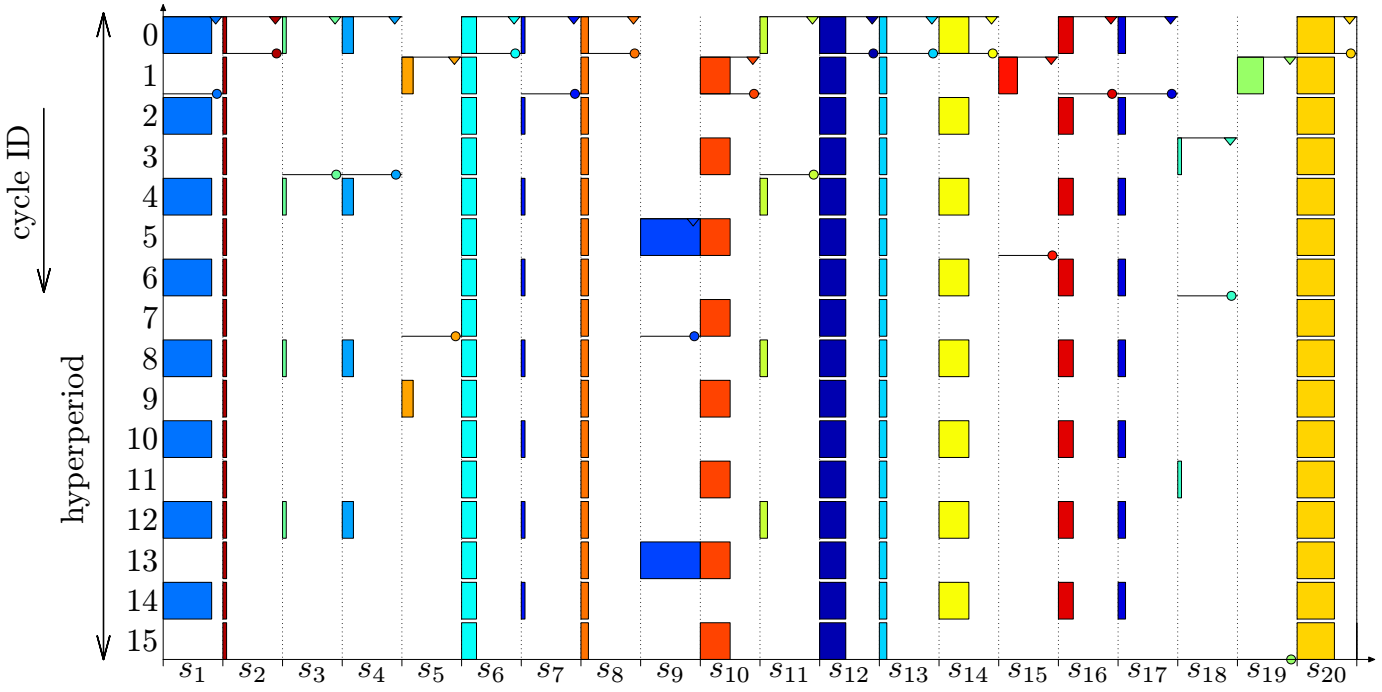

Figure 3. Visual representation of input data in Table II (one signal per one slot)

1) Time constrained frame packing with First Fit strategy: Algorithm 2 uses the First Fit (**FF**) strategy. It was inspired by [4] using this strategy for bin packing problem without time constraints (this strategy attempts to place the item in the first bin that can accommodate the item; if no bin is found, it opens a new bin and puts the item within the new bin). This algorithm inserts the signal in the first message, which has the same period and enough of the free space. If no such message exists, the new message with properties (size C_i, period T_i, release date $\tilde{O}_i$ and deadline $\tilde{D}_i$) of given signal is created.

Algorithm 2 Frame packing using FF strategy

Input: $signals_k$ = [s_1, ..., $s_{\#signals_k}$]
Output: $msgs_k$ = [m_1, ..., $m_{\#msgs_k}$]

$\#msgs_k$ = 0;
for (p=1 ... $\#signals_k$) {
    for (q=1 ... $\#msgs_k$) {
        if (s_p fits into m_q) {
            insert s_p into m_q;
            adjust constrains of m_q;
            break;
        }
    }
    if(s_p did not fit in any message) {
        create new message and insert s_p;
        $\#msgs_k$ += 1;
    }
}
$msgs_k$ = m;

As an example of frame packing process, see Table II and Figure 3 which show input signals for one node. During this part of the algorithm, the signals in Table II, were packed into messages depicted in Table III and on Figure 4.

Figure 3 shows all signals with their size C_i, periodicity T_i, release date $\tilde{O}_i$ (a horizontal line with little triangle) and deadline $\tilde{D}_i$ (a horizontal line with little circle). For example signals s_3, $s_4$and s_{11} can be grouped together into message m_6.

	C_i	T_i	$\tilde{O}_i$	$\tilde{D}_i$	signals
m_1	32	1	0	1	s_2, s_6, s_8, s_{12}, s_{13}
m_2	20	1	0	1	s_{20}
m_3	30	2	0	1	s_7, s_{14}, s_{16}, s_{17}
m_4	26	2	0	2	s_1
m_5	16	2	1	2	s_{10}
m_6	12	4	0	4	s_3, s_4, s_{11}
m_7	8	8	3	7	s_5, s_{18}
m_8	32	8	5	8	s_9
m_9	32	16	1	6	s_{15}, s_{19}

Table III
MESSAGES CREATED DURING FRAME PACKING FOR ONE NODE ($msgs_{node_7}$)

2) Best fit time window increasing strategy : Best Fit time window Increasing (**BFI**) strategy operates by first sorting the signals to be inserted in increasing order by their time window sizes, and then choosing from several messages to which particular signal can fit. The time window size is time between release date and deadline of the signal. Sorting signals in this way ensures that the most constrained signals will be packed together if it is possible and the least constrained signals will preserve the size of their window as much as possible.

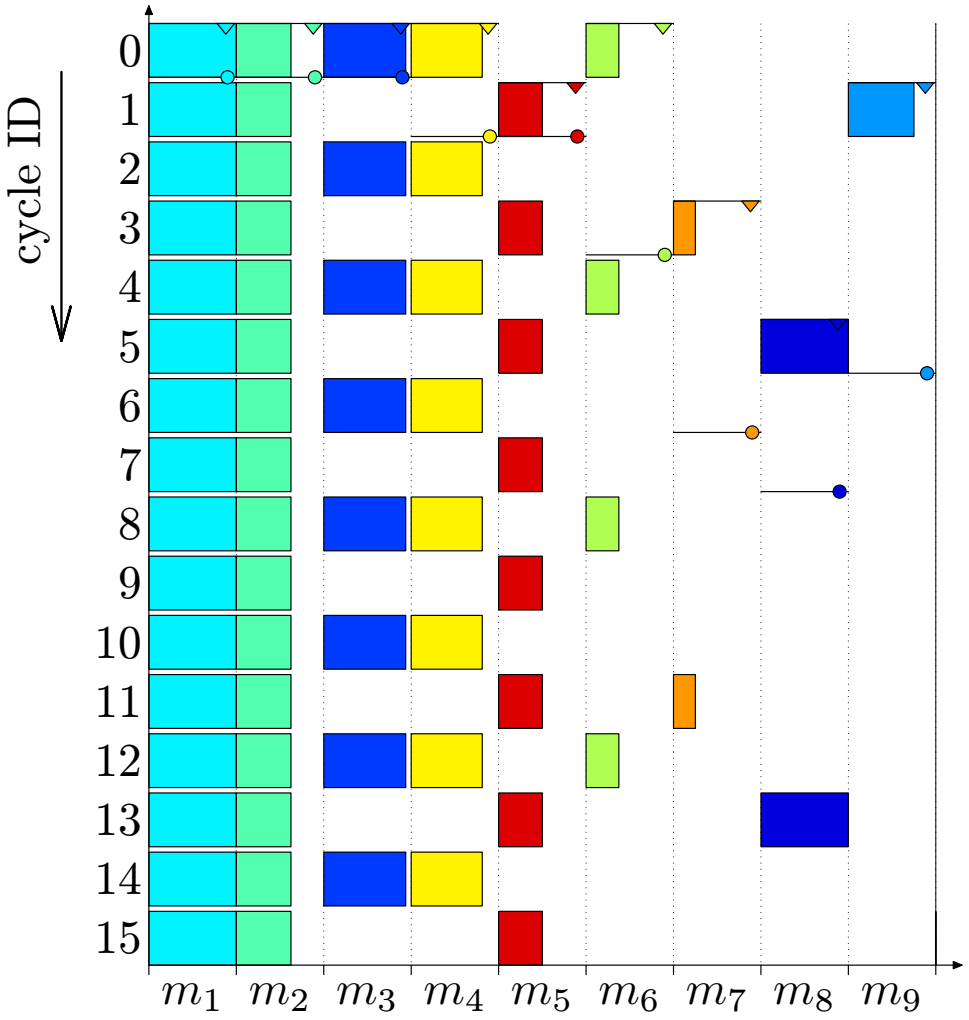

Figure 4. Visualisation of messages created during frame packing for one node

Algorithm 3 Frame packing using BFI

Input: $signals_k$ = [s_1, ..., $s_{\#signals_k}$], $bound$
Output: $msgs_k$ = [m_1, ..., $m_{\#msgs_k}$]

sort signals in increasing order of their time window sizes
for (p=1 ... $\#signals_k$) {
    $score$ = zeros($\#msgs_k$);
    for (q=1 ... $\#msgs_k$) {
        $score_q$ = score of assigning s_p into m_q;
    }
    if (max_{score} > $bound$) {
        [max_score, $index$] = max_{score};
        assign s_p into m_{index};
        adjust constrains of m_{index};
    } else {
        create new message and insert s_p;
        $\#msgs_k$ += 1;
    }
}
$msgs_k$ = m;

In order to find the Best Fit, all messages are evaluated with respect to their suitability to accommodate the signal. The score is linear combination of two factors:

- overlap of the message time window and signal time window
- the message size, in order to prefer completion of the message and to prevent fragmentation of the free space (this factor has lower priority)

If the signal does not fit in the message (different period or not enough of the free space), the score of the message is set to the $bound$ value.

If no message obtains the score higher than the $bound$ value, current signal will not be inserted in any message, but new message will be created from this signal. This avoids insertion of the signal into message with very different constrains and appearance of the messages, that could not be further aggregated in the following step of the algorithm.

Finally the massage with the best score is chosen for insertion of the signal. Both strategies of the frame packing have quadratic time complexity with respect to the number of the signals.

B. Message Aggregation

Most likely, not all of the messages would have exactly the size of the slot, so we decided to include another step, which tries to use the empty space in messages for messages, that can fit into the free space and have lower periodicity. Therefore we reduce the number of messages that needs to be scheduled, so it can cause reduction of the number of needed slot IDs allocated to the node.

Algorithm 4 The idea of the message optimization part of the algorithm

Input: $msgs_k$ = [m_1, ..., $m_{\#msgs_k}$]
Output: $agrmsgs_k$ = [$m^{'}_1$, ..., $m^{'}_{\#agrmsgs_k}$]

```
for (p=1 ... #msgs_k) {
    m'_p = m_p;
    for (q=(p+1) ... #msgs_k)
        if (m'_p can be merged with m_q) {
            m'_p = merge m'_p and m_q);
            break;
        };
    };
end;
agrmsgs_k = m';
```

This is obtained by iterating over all messages (notice, that previous part of the algorithm was performed separately by periods, but this part works on all messages of the node) sorted from the least occurring message to the most occurring one. Each message is checked, if it couldn't be merged with some other message for current node with preserving it's constrains. Sorting messages from the least occurring to the messages with highest periodicity reduces the amount of old unneeded data (data sent more frequently, than is it's periodicity) sent over the network, in the other words algorithm tries to reach the best possible effectivity. Pseudo code of the described method is depicted in Algorithm 4. This method could be briefly described as First-Fit (**FF**) strategy[4].

On Figure 5 it is depicted how release time and deadline was derived for example for the message m'_5. To meet all constrains of original signals (s_3, s_4, s_{10}, s_{11}), latest release date and earliest deadline must have been chosen.

	C_i	T_i	$\tilde{O}_i$	$\tilde{D}_i$	messages	signals
m'_1	32	1	0	1	m_1	s_2, s_6, s_8, s_{12}, s_{13}
m'_2	20	1	0	1	m_2	s_{20}
m'_3	30	2	0	1	m_3	s_7, s_{14}, s_{16}, s_{17}
m'_4	26	2	0	2	m_4	s_1
m'_5	28	2	1	2	m_5, m_6	s_3, s_4, s_{10}, s_{11}
m'_6	32	8	3	6	m_7, m_9	s_5, s_{15}, s_{18}, s_{19}
m'_7	32	8	5	8	m_8	s_9

Table IV
OPTIMIZED MESSAGES CREATED DURING THE MESSAGE OPTIMIZATION FOR ONE NODE ($agrmsgs_{node_7}$)

Optimization of the messages is done in asymptotic time complexity $\mathcal{O}(m^2)$ where m is the number of the messages from the previous step of the given node.

See Table IV and Figure 6 for the output of the message optimization part of the TCFS algorithm.

Algorithm 5 Enhancement of the optimization part of the algorithm

Input: $msgs(k)$ = [m_1, ..., $m_{\#msgs_k}$)], $bound$
Output: $agrmsgs_k$ = [$m_1^{'}$, ..., $m_{\#agrmsgs_k}^{'}$]

```
for (p=1 ... #msgs_k) {
    m'_p = m_p;
    score = inf(1,#msgs_k);
    while (max(score) > bound) {
        for (q=1...#msgs_k) {
            score_q = score of merging m'_p and m_q;
        }
        if (max(score) > bound) {
            m'_p = merge m'_p and m_index of max(score));
            score = inf(1,#msgs_k);
        }
    }
}
agrmsgs_k = m';
```

1) Enhancements: Currently analyzed message is now not merged with the first message that fulfils constrains. For every message, all other messages are evaluated. This evaluation takes into account the difference between release date and deadline of currently analyzed message and the other message, final size of the message in case of merge and the difference of the periods. There is the new bound, that say - if there is no message their's evaluation exceeds the bound, don't merge the current message with any other even if there is some message that fulfils the constrains. While introducing the bound, optimize messages will primarily merge messages with similar constrains and other parameters.

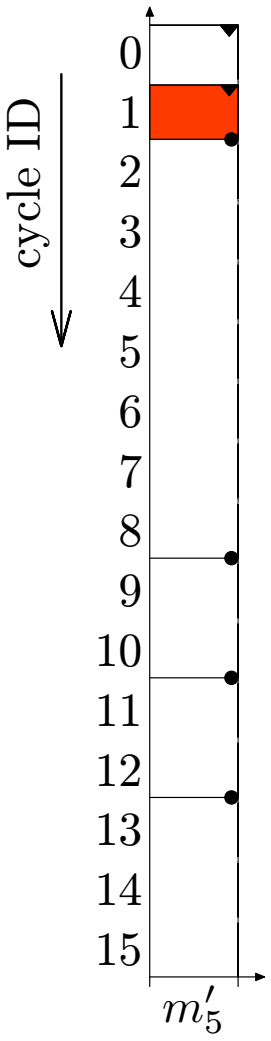

Figure 5. Demonstration how was created release date and deadline for message m'_5

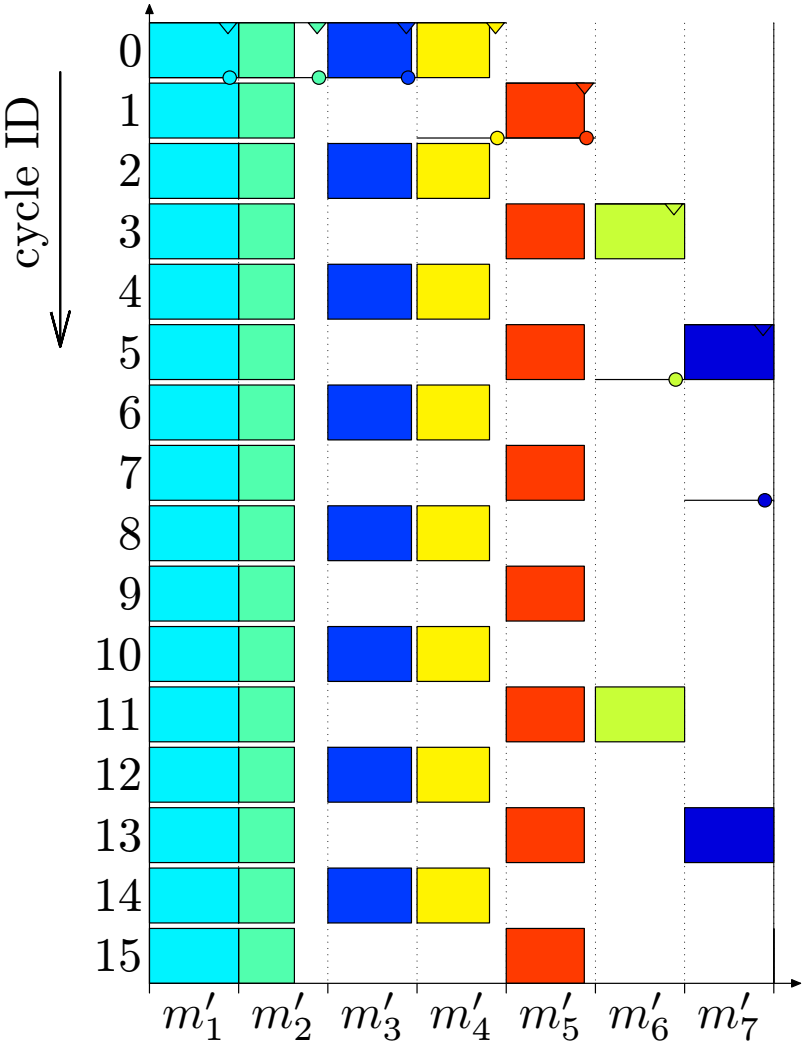

Figure 6. Visualisation of optimized messaged created during the message optimization for one node

Enhanced message optimization could be also described as Best Fit (**BF**)[4] strategy.

Asymptotic time complexity of the enhanced optimize messages is increased over the original version to $\mathcal{O}(m^3)$ where m is the number of the messages from frame packing step of the given node.

C. Create schedule

At this stage our problem is to nonpreemptively schedule unit time harmonic optimized messages with integer release date and deadline without jitter to minimum number of slot IDs (in scheduling terms, slot IDs act as identical processors).

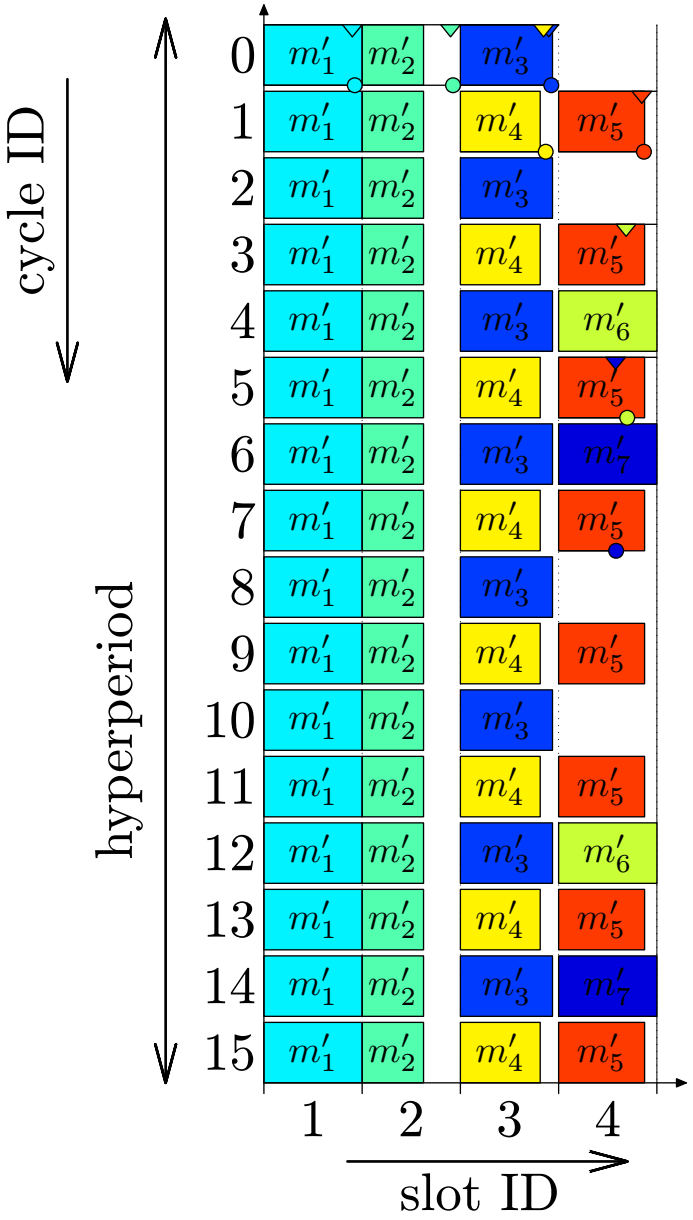

Figure 7. Final computed schedule for one node ($schedule_{node_7}$)

Optimized messages are being scheduled from the most frequent messages to the messages with the longest period. For each message, slot IDs are searched for the first slot, that has enough of the free space (empty frames in the cycles of the hyperperiod) and meets all requirements (release date and deadline of the message). If proper position in the slot ID is found, it's place is recorded to the message and corresponding cycles are marked (because message generally occupies more frames thanks to it's periodicity) as occupied. If no suitable slot ID could be found, new slot ID is reserved, and message is placed into this newly created one and occupied cycles are marked in the same way as for the existing slot ID.

Algorithm 6 Pseudoalgorithm of schedule creation

Input: $agrmsgs_k$ = [$m^{'}_1$, ..., $m^{'}_{\#agrmsgs_k}$]
Output: $schedule_k$

```
sort agrmsgs_k by period;
#slots_k = 0;
for (p=1 ... #agrmsgs_k) {
    for (q=1 ... #slots_k) {
        for (c=m'_p.release ... m'_p.deadline) {
            if(m'_p fits into slot q starting in cycle c) {
                assign m'_p into slot q starting in cycle c;
                break;
            }
        }
    }
    if (m_p didn't fit into any slot) {
        #slots_k += 1;
        assign m'_p into slot #slots_q
                starting in cycle m'_p.release;
    }
}
```

Example final computed schedule is shown on the Figure 7. It shows, that 20 signals on the input of the algorithm could be scheduled into 4 slots. At the beginning, signals requested to transfer 1560 bits of information during the hyperperiod. Thanks to the optimization part, in the final schedule, we need to transfer 1632 bits of the information, so our optimization overhead is in this case 4.6%. In those 4 slots it is possible to transfer at most 2048 bits, that means, that this example use nearly 80% of the available bandwidth.

Creation of schedule is done in asymptotic time complexity $\mathcal{O}\left(max\left(m'_n \cdot i_n \cdot c,\, m'_n \cdot log(m'_n)\right)\right)$ where m' is the number of the optimized messages for the given node, i is the number of the slot IDs associated to the given node and c is the number of cycles in the hyperperiod. $m' \cdot log(m')$ stands for the sort complexity, because optimized messages must be sorted on the input of this part of the algorithm.

1) ILP formulation: Since the number of optimized messages is very significantly lower than the number of signals, optimal solution for the create schedule part of the TCFS algorithm could be employed.

x_i^j — jth possible placement of the optimized message i from it's release date (jth cycle beginning with the release time), $x \in \{0, 1\}$

${}_t y_i^j$ — tth repetition of the optimized message i when first occurrence of this optimized message i is placed in jth cycle after it's release date, $y \in \{0, 1\}$

z_c — number of placed messages in the cycle c, $z \in \{1, ..., max(slot\,ID_{node_k})\}$

$\sum_{j=1}^{window\,size_{message_i}} x_i^j = 1$: there is allowed only one first occurrence of the optimized message i during the window, for each optimized message i

$\sum_{t=1\,t}^{|t_i|} y_i^j - |t_i| x_i^j = 0$: for each optimized message i and it's possible placement j, where $|t_i|$ is the number of repetition of the optimized message i during hyperperiod

$\sum {}_t y_i^j - z_c \leq 0$: all instances of messages placed in cycle c, for each cycle in the hyperperiod

$\sum_{c=1}^{cycles\,in\,hyperperiod} z_c = \sum_{i=1}^{\#optmessages} |t_i|$: guarantee, that there are allocated places exactly for the number of unique instances of messages

It is not needed to cover necessity to send messages always in the same slot ID, because we can achieve that later by allocating messages to slots from the most frequent messages to the least frequent ones.

Algorithm 7 Pseudoalgorithm of create schedule part when using ILP

Input: $agrmsgs_k$
Output: $schedule_k$

$schedule\_heuristic$ = create_schedule($agrmsgs_k$);
UB = $schedule\_heuristic$.slot_IDs;
create matrixes($agrmsgs_k$);
set upper bounds to UB;
while (ILP yields feasible schedule) {
    $xmin$ = execute ILP on matrixes;
    UB = UB - 1;
    set upper bounds to UB; }
$schedule_k$ = extract schedule from $xmin$;

Because ILP is a lot slower than our heuristic, executing heuristic algorithm first gives the upper bound (UB) of the slot IDs to user for ILP. Than ILP is started in loop while lowering available number of slot IDs. When ILP return that this is not feasible, we state that previous feasible schedule is optimal in meaning of the minimal number of slot IDs for optimized messages on the input. Pseudoalgorithm of the create schedule part of the algorithm with employed ILP is depicted in Algorithm 7.

2) Better sorted optimized messages: Due to the not sorting optimized messages according to their periods, it is not ensured, that if there is the free space in the slot for the first occurrence of the optimized message during scheduling, there have to be also free space for every other occurrence of the optimized message during the hyperperiod. Because of that it is required to check each occurrence of the optimized message during the hyperperiod during assigning that optimized message to the slot. Inspection of every occurrence of optimized messages in comparison to verification only the first occurrence for each optimized message increase the algorithm complexity.

The most constrained optimized messages (with the smallest window) should be evidently scheduled first. Also it seems, that it could be favorable to allocate position for optimized messages that has lower release dates before those with higher release dates with conserving the priority of the most constrained optimized messages. That denoted required sorting of optimized messages at the very beginning of the create schedule part of the TCFS algorithm (Algorithm 6). In case of input without release dates and deadlines, this method is equivalent to the sort by period in the original version of the create schedule. As note earlier there have to be few modifications to the heuristic to check all optimized messages' occurrences during the hyperperiod for available space during searching for the slot ID, into which optimized message could be placed.

VI. Results

Newly introduced bounds for the frame packing and message optimizations (described in subsection V-A2 and V-B1) were tested with nearly all possible combinations for each set of signals, but it was discovered that one carefully selected bound for the frame packing and another for message optimization gave the best results for all signal sets.

Comparing just parts of the algorithm doesn't make sense because it is not possible to find the evaluation criteria without finalizing the schedule. For example for frame packing it is not possible to tell that lower number of messages on the output is better, because if those messages has very small window, then they may require more slot IDs to be schedulable.

During the evaluation of the enhanced version of the TCFS algorithm it was found, that ILP always confirm that it is not possible to find schedule with the lower number of slot IDs even if it is known that there are few theoretical cases (for example see Figure 8), where ILP compute schedule with lower number of slot IDs than used heuristic.

The main goal, to minimize the number of required slot IDs for each signal set, is achieved better than in the previous simpler version. But the improvement in the number of required slot IDs is not so significant, because we hit the theoretical minimum required number of slot IDs after the optimization message part of the algorithm for nearly all benchmarked signal sets. Enhanced version of the TCFS algorithm is still pretty fast and is able to schedule thousands of signals for one node in few seconds.

A. Benchmarks

Comparison results of several versions of TCFS algorithm are depicted in Table V, where each column represents one signal set (average values from 10 instances of the same signal set) and rows representing parameters of signals sets and algorithm results. Those signal sets are created according to the Subsection IV-A. Time constraints of signal sets are described in following list:

Set1	release dates nor deadlines are defined
Set2	deadlines are not set, release dates are spread across the first 5 FlexRay cycles

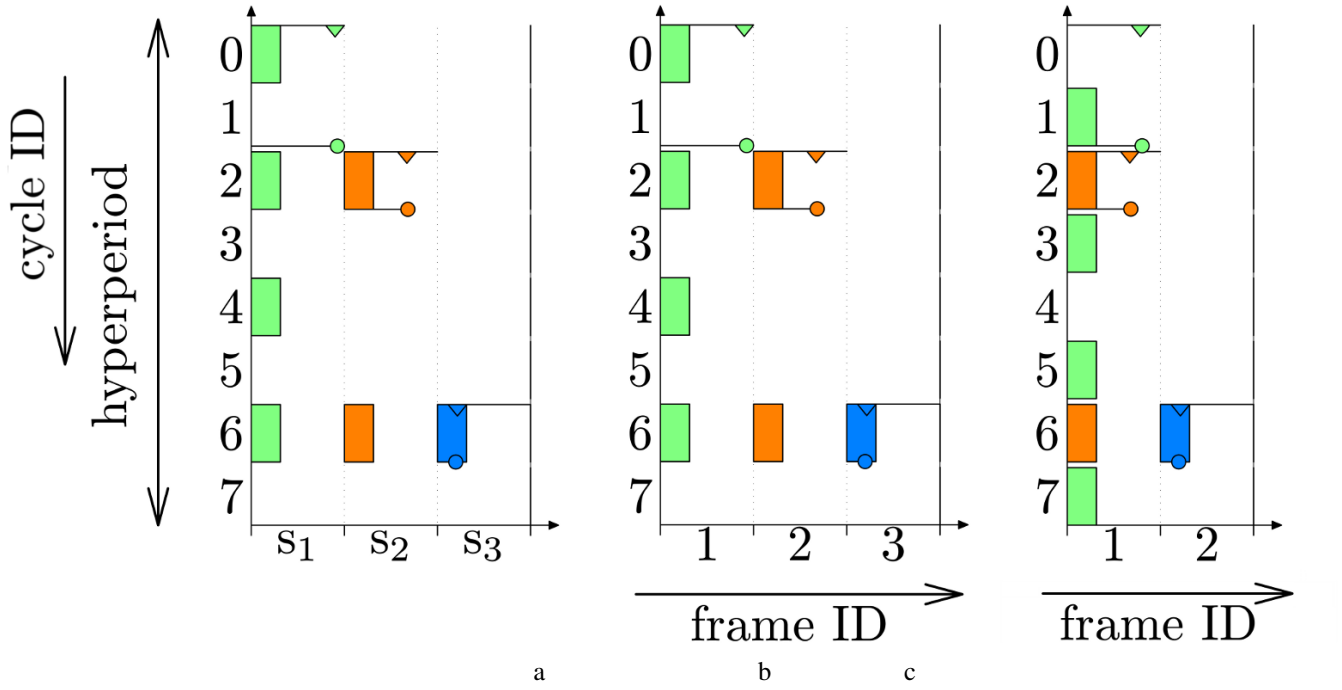

Figure 8. a - Input signal set; b - Schedule generated just using heuristic; c - Schedule with employed ILP

Set3	deadlines are spred across the last third of the signals' period, release dates are spread across the first 5 FlexRay cycles
Set4	similar to set2
Set5	deadlines are not set, release dates are spread across the first 5 FlexRay cycles
Set6	similar constraints as set5, but with higher network usage
Set7	release dates nor deadlines are defined, but there are just few signals per node
1ECU500	few deadlines are randomly defined, release dates are spread across the first 5 FlexRay cycles
1ECU1000	same as 1ECU500
1ECU3000	same as 1ECU500, slot size larger than the rest of the signal sets

To describe signal set, there are 3 rows: *#signals* represent the overall number of signals in the set for all nodes; *Nodes* represent number of nodes among signals are slit up; and *min #ID* is the number, that represent theoretical absolute minimum number of required slot IDs without reflecting release dates and deadlines and in ideal case. For each version of TCFS algorithm results there are two rows - *#ID* representing the number of slot IDs required by the schedule computed by TCFS algorithm for current signal set; and *t* representing computation time of the algorithm in seconds.

Specification of usage of each algorithm part across executed versions is described in Table VI. Create schedule part is not present in the table, because the same is used in all versions. Versions of FP and MO for Window and Message versions are slightly different compared to the Enhanced version in computation of score for assigning signal into message or merging messages respectively. "spec" means carefully selected bound, that works best for all signal sets, "all" means that all possible bounds are tested. Bounds for FP and MO in Window version are set to values that secure the messages' window size and for Message version those bounds ensure creation of the least possible number of elements in each step.

Table V confirms, that Enhanced version of the TCFS performs better than the original version on all benchmarks without significantly increase of the processing time. As expected, Enhanced version of the TCFS algorithm performs at least similar to the other versions. Trying all possible frame packing and message optimization bounds for each signal set individually doesn't find better schedule and is much more time consuming. Experiments with setting frame packing bound and optimize message bound to preserving largest possible window for the messages or pack signals into the minimal number of slot IDs possible also doesn't result into better schedule over the Enhanced case.

TCFS algorithm with employed ILP is not presented in the table for comparison, because of the fact that employing ILP never find better schedule as stated earlier, but notably increase the processing time of the algorithm. Create schedule part of the algorithm with different sorting at the beginning as described in Subsection V-C2 is also not presented in the comparison table since it delivers exactly the same results as the original version but a bit slower.

Overall time complexity of the Enhanced TCFS algorithm without employing ILP is $\mathcal{O}\left(S + n \cdot \left(s_n^2 + m_n^3 + max\left(m'_n \cdot i_n \cdot c,\, m'_n \cdot log(m'_n)\right)\right)\right)$ in the worst case scenario, where S is the overall number of the

		Set1	Set2	Set3	Set4	Set5	Set6	Set7	1ECU_500	1ECU_1000	1ECU_3000
#signals		622	622	622	622	566,7	784,4	850,7	467,3	973	2704,1
nodes		3	3	3	3	6	6	23	1	1	1
min #ID		21,3	21,3	21,3	21,3	12,1	28,3	27,8	20,1	41,5	28,9
Original	#ID	24,2	24,2	24,2	24,2	17,7	32,5	42,2	20,8	42,1	29,7
	t [s]	0,52	0,53	0,53	0,53	0,47	0,68	0,71	0,42	0,91	2,60
Enhance	#ID	23,8	23,8	23,8	23,8	17,6	31,9	41,3	20,4	41,9	29,2
	t [s]	0,62	0,62	0,60	0,66	0,64	1,04	0,95	0,47	0,99	2,96
Window	#ID	23,8	23,8	23,8	23,8	17,7	32	43,4	20,7	42,2	29,6
	t [s]	0,61	0,62	0,62	0,62	0,57	0,78	0,83	0,46	0,96	2,87
Message	#ID	24	24	24	24	17,7	31,9	43,4	20,4	41,9	29,2
	t [s]	1,39	1,41	1,52	1,36	1,07	1,65	1,53	1,17	3,30	9,47
Bounds	#ID	23,8	23,8	23,8	23,8	17,6	31,9	41,3	20,4	41,9	29,2
	t [s]	15,79	15,88	15,84	15,93	14,77	20,46	24,97	11,01	24,51	71,75

Table V
COMPARISON OF DIFFERENT ALGORITHM VERSIONS

	Original	Enhance	Window	Message	Bounds
FP	FF	BFI	BFI-w	BFI-m	BFI
FP bound	-	spec	FP-WP	FP-MP	all
MO	FF	BF	BF-w	BF-m	BF
MO bound	-	spec	MO-WP	MO-MP	all

Table VI
ALGORITHM VERSIONS DESCRIPTION

signals on the input, n is the number of the nodes, s_n is the number if the signals of the given node, m_n is the number of the messages of the given node, m'_n is the number of the optimized messages of the given node, i_n is the number of slot IDs assigned to the given node and c is the number of the cycles in hyperperiod.

All benchmarks were executed in Matlab R2009b on single-core Intel Core 2 Solo processor powered computer with 1GB RAM and Windows d operating system.

VII. CONCLUSION

In this paper schedule signals into FlexRay static segment was described. Presented algorithm is divided into three consequent parts. This paper starts with the simple version of each part and propose enhancements for each part. Designated algorithm is able to schedule very large signal sets in several seconds. The more signals per node is in the signal set, the better results algorithm return. With lower number of signals per node, algorithm may return notably higher required number of slot IDs, than the theoretically minimum number of required IDs.

In order to verify the schedule shown in this paper, we have conducted simple experiment with real HW based on Freescale MC9S12XF and Tektronix oscilloscope MSO/DPO4000B Series. Figure 9 shows screenshot of the oscilloscope displaying the flow of the data for part of the communication cycle on the FlexRay bus. Signals used during the experiment are the same as signals used to demonstrate the algorithm - Figure 3. Freescale units were configured to fulfil the schedule shown on the Figure 7. Used oscilloscope is able to understand the data on the FlexRay bus, so in case it will be magnified, there would be the number of cycle and slot in each communication. In this screenshot, the peaks identify actual transfer of the signal in which row the peak is. There are not all signals displayed in the Figure. For example it is showed, that signals s_6, s_8, s_{12}, s_{13} (signal s_2 is not showed on this Figure) are transferred in cycle 0 and slot 33.

Presented algorithm could be also used for incremental design in case that there are still available static slots into that could be new communicating units scheduled. For those cases it is possible to use TCFS algorithm as is, it is just required to define valid preprocessing parameters (slot length, etc.) of the signals in consideration of configuration of current FlexRay bus.

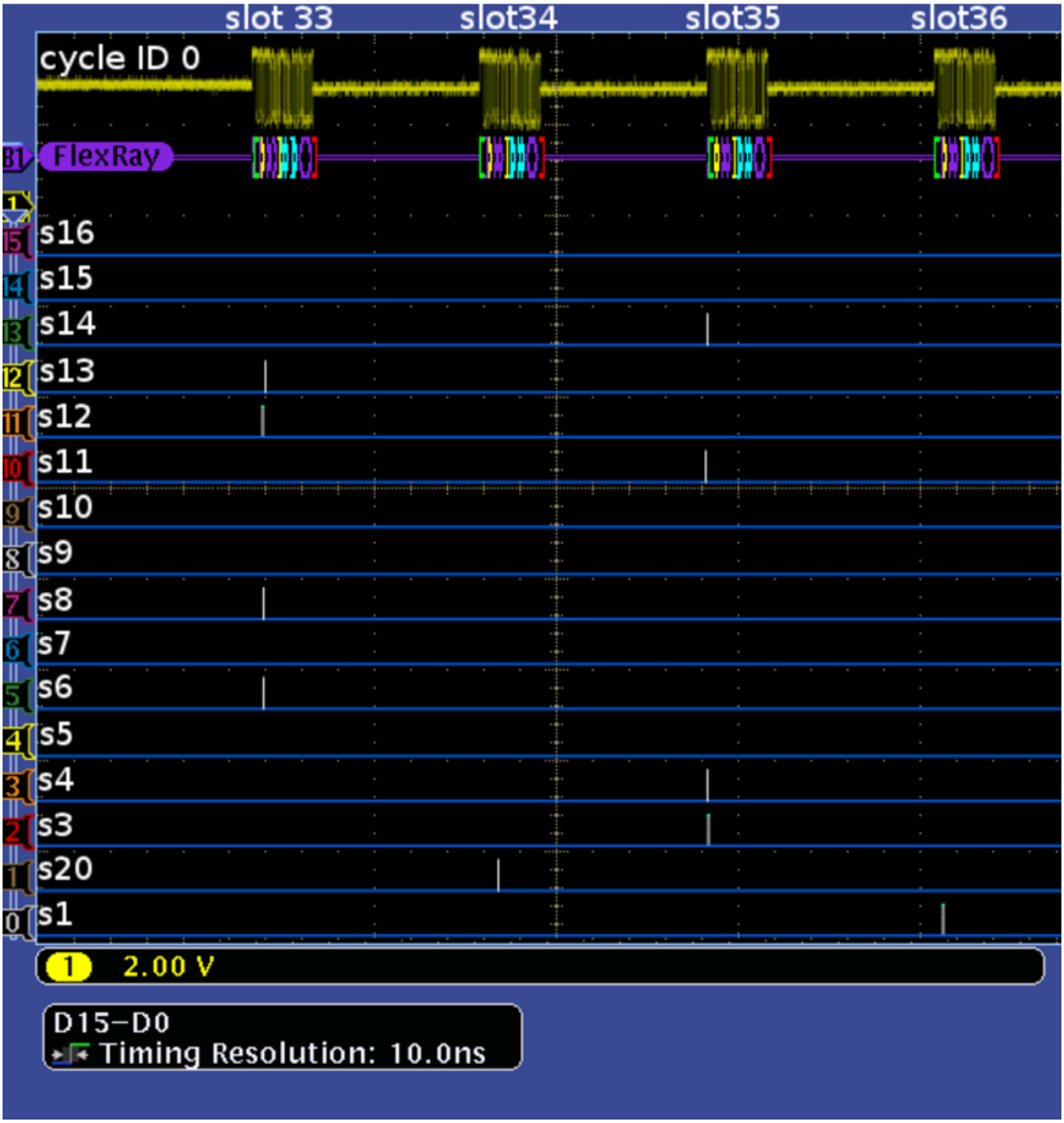

Figure 9. Experiment on real FlexRay hardware

References

[1] Amos Albert. Comparison of Event-Triggered and Time-Triggered Concepts with Regard to Distributed Control Systems. In *Embedded World 2004*, pages 235–252, 17.–19.02.2004 2004.

[2] Josef Berwanger, Martin Peteratzinger, and Anton Schedl. Flexray startet durch – FlexRay-bordnetz får fahrdynamik und fahrerassistenzsysteme. In *Elektronik automotive: Sonderausgabe 7er BMW*, 2008.

[3] Christelle Braun, Lionel Havet, and Nicolas Navet. Netcarbench: A benchmark for techniques and tools used in the design of automotive communication systems. In *The 7th IFAC International Conference on Fieldbuses and Networks in Industrial and Embedded Systems (FeT'2007)*, November 2007.

[4] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation Algorithms for Bin-Packing- An Updated Survey. *Algorithm Design for Computer Systems Design*, G. Ausiello, M. Lucertini, and P. Serafini (eds.), pp. 49–106, Springer-Verlag, 1984.

[5] FlexRay Consortium. Flexray Protocol Specification v2.1 rev. a, 2005.

[6] Mathieu Grenier, Lionel Havet, and Nicolas Navet. Configuring the communication on FlexRay: the case of the static segment. In *Embedded Real Time Software*, France, 2008.

[7] H. Kopetz. A solution to an automotive control system benchmark. In *Real-Time Systems Symposium, 1994., Proceedings.*, pages 154 –158, 7-9 1994.

[8] AUTOSAR Development Partnership. Specification of FlexRay Transport Layer. Version 2.0.1, June 2006.

[9] AUTOSAR Development Partnership. Specification of Module FlexRay Interface. Version 2.0.0, June 2006.

[10] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing analysis of the FlexRay communication protocol. *Real-Time Systems, 2006. 18th Euromicro Conference on*, pages 11 pp.– 216, 2006.

[11] E. G. Schmidt and K. Schmidt. Message scheduling for the FlexRay protocol: The dynamic segment. *Vehicular Technology, IEEE Transactions on*, 58; 58(5):2160–2169, 2009.
[12] K. Schmidt and E. G. Schmidt. Message scheduling for the FlexRay protocol: The static segment. *Vehicular Technology, IEEE Transactions on*, 58; 58(5):2170–2179, 2009.
[13] Byungseok Seo and Dongik Lee. Determining the length of static message for efficient use of FlexRay network. *SICE Annual Conference 2010, Proceedings of*, pages 563 –566, aug. 2010.
[14] Young Hun Song, Man Ho Kim, Suk Lee, Kyung Chang Lee. Node-based FlexRay Scheduling Method for Reducing the Scheduling Complexity, *Proceedings of 2011 International Conference on Mechanical, Industrial, and Manufacturing Engineering*, 2011
[15] Piotr Swedrowski, Raphael Guerra. Optimal scheduling approaches for FlexRay bus, *Proceedings of 5th Real-Time Systems Seminar*, 2011
[16] Bogdan Tanasa, Unmesh Dutta Bordoloi, Petru Eles, Zebo Peng. Scheduling for Fault-Tolerant Communication on the Static Segment of FlexRay, *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, 2011
[17] Bogdan Tanasa, Unmesh Dutta Bordoloi, Petru Eles, Zebo Peng. Reliability-aware frame packing for the static segment of flexray, *Proceedings of the ninth ACM international conference on Embedded software*, pages 175-184, 2011
[18] Haibo Zeng, M. Di Natale, A. Ghosal, and A. Sangiovanni- Vincentelli. Schedule Optimization of Time-Triggered Systems Communicating Over the FlexRay Static Segment. *Industrial Informatics, IEEE Transactions on*, 7(1):1 –17, 2011.