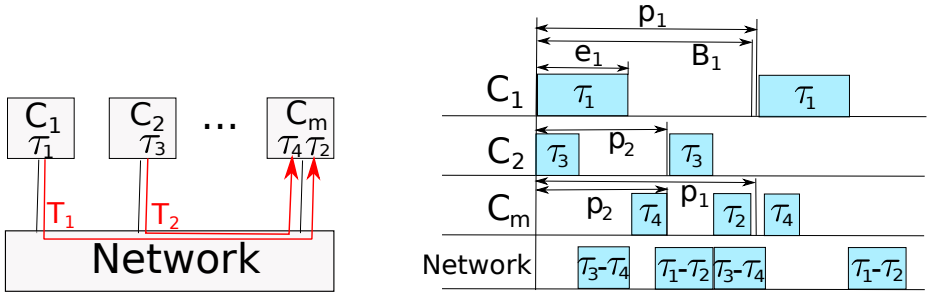
# Multi-Periodic Time-Triggered Scheduling for Safety-Critical Systems

Anna Minaeva Benny Akesson Zdeněk Hanzálek
Czech Technical University in Prague

## I. Motivation and Problem Formulation

The problem of scheduling safety-critical systems with multiple cores that have small local memory and communicate through a network, e.g. FlexRay, can be found in automotive, avionics and other industries. Periodic tasks, such as sensing and actuation, execute on these cores and need to communicate with each other. Tasks have hard real-time requirements that must be always satisfied to guarantee safe operation. Moreover, due to the nature of these systems, tasks have different periods, since control tasks release jobs with different frequencies. The time-triggered approach is commonly used in safety-critical systems due to highly predictable behaviour of the scheduled tasks, resulting in simplicity of both verification if a given solution satisfies all constraints and debugging when something goes wrong.

We consider the problem of *multi-periodic non-preemptive scheduling of tasks with precedence constraints on multiple resources*. The problem is shown in Figure 1a, where there are $m$ cores $C_1, C_2, \cdots, C_m$ that communicate via a network. We suppose that there are several *transactions* $T_k$ in the form $C_i \rightarrow Network \rightarrow C_j$, where first some task must be executed on core $C_i$, then there is communication via the network, followed by execution of a task $\tau_i$ with processing time $e_i$ on core $C_j \neq C_i$. We can formalize each transaction as a chain of tasks $\tau_i \rightarrow$ Network $\rightarrow \tau_{i+1}$, released at the beginning of each period with implicit deadlines $D_i = p_i$, where each task $\tau_j$, $j = 1, 2, \cdots, n$ is assigned to exactly one core where it must be executed. In the example in Figure 1a, there are two transactions $T_1 = \tau_1 \rightarrow$ Network $\rightarrow \tau_2$ and $T_2 = \tau_3 \rightarrow$ Network $\rightarrow \tau_4$. Each transaction $T_k$ may have a different period $p_k$ and a required maximum end-to-end delay, $B_k$, i.e. the maximum time from the beginning of the first task in a transaction to the end of the second and final task in the transaction is constrained by the user. Thus, the problem is to find a time-triggered schedule for the tasks on the cores and communication on the network such that the end-to-end delay of each transaction satisfies the requirement.




(a) Problem description (b) Example schedule

Fig. 1: Multi-periodic scheduling problem description with examples of transactions and solution

A Gantt chart of a possible solution to the problem with transactions $T_1$ and $T_2$ from Figure 1a is shown in Figure 1b. Here, tasks $\tau_1, \tau_2, \tau_3, \tau_4$ and their communication are scheduled on cores $C_1, C_2, C_m$ and on the network. We can see that the periods of tasks and their processing times are different and all tasks and transactions satisfy their deadlines and end-to-end delay requirements. The solution to our problem is defined as a set of start times and durations (equal to the processing time of each job) during a scheduling interval, called the hyperperiod. The hyperperiod is a least common multiple of all the periods of the transactions.

Applying the stated problem on a real system brings additional constraints on the solution. First of all, the small local memories of embedded systems can only store schedules of limited length, i.e. the hyperperiod of the final schedule may not exceed a size $L$ defined by the user. Moreover, the size of an average problem is in the range of *thousands of transactions*, which requires the approach to be scalable, since it has to be possible to find a schedule to the problem in reasonable time. Thus, an ILP approach does not seem to be a good final solution due to the large number of required decision variables, resulting in very long computation time. However, some advanced exact optimization methods or heuristic approaches are good candidates.

Problems similar to the presented problem were considered in different formulations before. In [1], the authors solved similar problem with mono-periodic transactions, i.e. $p_1 = p_2 = \cdots = p_N$. Although it is possible to schedule all transactions with the minimum possible period also in our problem, it results in large over-allocation and restricts the problem instances

schedulable by the approach. Over-allocation here means that more time is reserved in the schedule than the tasks can consume. In [2], [3], [4] and [5] the authors require perfectly periodic scheduling, where the whole schedule for all jobs from a task is completely defined solely by the starting time in the first period. This results in over-allocation and significantly restricts the set of solvable instances as well. The main difference of our problem with [2] is that the athours do not put any constraints on end-to-end delays of transactions. Works [3], [4] and [5] consider the problem of multi-hop networks scheduling, which is a generalization of our problem as it allows complicated network structure instead of a single communication channel in our case and it adds some network constraints. Moreover, the authors in [3] target their solution to the problem with harmonic periods (i.e. for each pair of periods $p_i \geq p_j$ holds $p_i | p_j$). The authors in [6] consider a similar problem without precedence constraints and with periods that are determined by the hardware to be $m \cdot 2^k$, where $m \in \mathbf{N}$ is a constant and $k \in \mathbf{N}$ can take any integer value. Their solution cannot be used for the described problem due to the absence of precedence constraints between tasks.

## II. Open Problems

The problems mentioned in Section I influence both the considered problem and the applied solution: 1) bounded storage capacity, 2) the requirement not to over-allocate, 3) the requirement to be able to solve as many real-life problems as possible, and 4) the scalability issue. The formulation of the problem is equally important as the method chosen to solve it. The reason is that the problem, being formulated in a too constrained manner, can suffer unreasonably high rate of problem instances with no possible solution. For instance, restricting ourselves to perfectly periodic allocation has following advantages: a straight-forward analysis and the fact that the amount of memory required to save the obtained schedule grows linearly with the number of transactions in contrast to the general case, where relatively prime periods can result in a very large hyperperiod. Moreover, the problem of over-allocation is automatically solved as well, since it is not possible to over-allocate with perfectly periodic allocation, considering initial periods. However, it seems that requiring perfectly periodic allocation on the initial problem is too restrictive.

Another question is "is it possible to slightly change the problem in order to have almost all the advantages of perfectly periodic allocation at a lower cost in terms of number of unschedulable problem instances?". By converting the initial problem to a problem with harmonic periods similarly to [7], it is possible to schedule the tasks in a simple way with significantly reduced length of the schedule. For instance, having a problem instance with three transactions with periods $p_1 = 2$, $p_2 = 5$ and $p_3 = 16$ it is possible to convert it to the problem, where $p_2 = 4$ with the rest of the periods unchanged. Then, instead of having the schedule length of $lcm(2, 5, 16) = 80$ we have $lcm(2, 4, 16) = 16$, significantly saving the local memory space, but resulting in over-allocation since now we schedule the tasks in $T_2$ every 4 time units instead. Furthermore, scheduling becomes easier, since there is less collisions in the schedule when allocating tasks perfectly periodically. However, this solution can cause significant over-allocation, resulting in impossibility to solve instances with high load.

Ideally, it is required not just to solve the satisfiability problem, but also care about the utilization of the final schedule. Typically, there are not only periodic tasks in the system, but also sporadic tasks that are required to be scheduled in an event-triggered manner. It means that once in some relatively short interval there should be a gap in the schedule that we want to maximise for sporadic tasks to be efficiently scheduled.

## Acknowledgement

This work was supported by the Grant Agency of the Czech Republic under the Project GACR P103/12/1994, the Ministry of Education of the Czech Republic under project number CZ.1.07/2.3.00/30.0034, and Eaton European Innovation Centre.

## References

[1] Z. Hanzalek, P. Burget, and P. Sucha, "Profinet io irt message scheduling with temporal constraints," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 3, pp. 369–380, Aug 2010.

[2] L. Cucu and Y. Sorel, "Non-preemptive multiprocessor scheduling for strict periodic systems with precedence constraints," *Proceedings of 23rd Annual Workshop of the UK Planning and Scheduling Special Interest Group, PLANSIG'04*, 2004.

[3] W. Steiner, "An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks," *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, vol. 31, pp. 375–384, November 2010.

[4] S. S. Craciunas and R. S. Oliver, "Smt-based task- and network-level static schedule generation for time-triggered networked systems," in *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, ser. RTNS '14. New York, NY, USA: ACM, 2014, pp. 45:45–45:54. [Online]. Available: http://doi.acm.org/10.1145/2659787.2659812

[5] A. Biewer, B. Andres, J. Gladigau, T. Schaub, and C. Haubelt, "A symbolic system synthesis approach for hard real-time systems based on coordinated smt-solving," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15. San Jose, CA, USA: EDA Consortium, 2015, pp. 357–362. [Online]. Available: http://dl.acm.org/citation.cfm?id=2755753.2755834

[6] J. Dvorak and Z. Hanzalek, "Multi-variant time constrained flexray static segment scheduling," *Factory Communication Systems (WFCS), 2014 10th IEEE Workshop on*, pp. 1–8, 2014.

[7] K.-J. Lin and A. Herkert, "Jitter control in time-triggered systems," *System Sciences, 1996., Proceedings of the Twenty-Ninth Hawaii International Conference on*, vol. 1, pp. 451–459, 1996.