Design of TDMA-based In-Car Networks: Applying Multiprocessor Scheduling Strategies on Time-triggered Switched Ethernet Communication

Jan Kamieth, Till Steinbach, Franz Korf, Thomas C. Schmidt
Department of Computer Science
Hamburg University of Applied Sciences, Germany
{jan.kamieth, till.steinbach, korf, schmidt}@informatik.haw-hamburg.de

Abstract

Real-time Ethernet variants gain importance for communication infrastructure of various time-critical domains, such as in-car networks. Synchronous time-triggered traffic guarantees strict timing but requires a detailed schedule for all participants. Designing these schedules by hand is extensive work and with increasing network size almost impossible.

In this paper, we contribute a mapping of the time-triggered network scheduling problem into the domain of multiprocessor scheduling. This set of transformation rules allows us to apply established scheduling algorithms as well as new strategies to organise time-triggered switched networks. Experimental results from a prototype implementation of a scheduling framework based on this mapping show the feasibility of our concept. The framework demonstrates a multiple solver approach that uses algorithms with different optimality criteria in parallel.

1 Introduction

Modern automobiles are distributed and highly interconnected systems with up to 100 electronic control units (ECU) and more than 2000 signals [1]. Due to new *Advanced Driver Assistance Systems* (ADAS), upcoming X-by-Wire systems, enter- and infotainment applications, the amount of communication is growing. Today, the ECUs exchange data via several fieldbus technologies, which were designed according to specific requirements of different application domains like infotainment or engine control. Main technologies are *Controller Area Network* (CAN) [16], *Media Oriented Systems Transport* (MOST) [20] and *FlexRay* [10]. Gateways between different domain specific busses allow messages to traverse the borders of these sub-networks for domain overlapping functionality.

New applications that require camera or laser scanner based sensors introduce bandwidth demands that exceed the capacity of todays fieldbus technologies used in cars. Currently, expensive wiring like Low Voltage Differential Signaling (LVDS) is used for bandwidth demanding applications. To satisfy the bandwidth requirements, to reduce costs, and to simplify the complex communication structures at the same time, the industry shows a growing interest towards Ethernet as an in-vehicle network infrastructure [5]. Recently, research focused on how Ethernet technology can provide benefits to the different functional domains. Investigated technologies were Ethernet/IP [17], Ethernet AVB [22] and time-triggered Ethernet [26].

Time-triggered Ethernet Protocols are promising candidates to fulfil the strict temporal requirements of automotive and other application areas. Therefore, the IEEE *Time-Sensitive Networking* (TSN) Task Group recently started the standardization of scheduled (time-triggered) traffic (IEEE 802.1Qbv [15]) as an extension to the *Audio/Video Bridging* (AVB) protocol suite.

The major challenge when using time-triggered Ethernet protocols is the correct and efficient scheduling of applications on ECUs and time-triggered messages in the network. Since time-triggered fieldbusses such as FlexRay operate on a single shared medium, only one message schedule for the whole topology is required. In contrast, switched Ethernet is a point-to-point technology, limiting the congestion domain to the link between two participants. For each of these links a schedule for time-triggered messages must be defined. This set of schedules must be coordinated to satisfy the specified end-to-end real-time requirements of all applications.

Due to the nature of time-triggered traffic, a time window, which is defined by the schedule to transfer a specific time-triggered message over a link, is allocated exclusively for this message. Since this limits the transmission of non time-triggered traffic, gaps must be placed between the time-triggered windows, to allow for non time-triggered traffic to be sent. To handle this requirement during the design process, a framework supporting different scheduling algorithms and heuristics within an incremental work flow is required.

With this paper, we introduce a system model that maps the network scheduling problem to the well known problem of real-time multi processor scheduling. This mapping allows us to apply existing scheduling strategies like

Earliest Deadline First (EDF), new algorithms, and manual modifications in an incremental design flow in order to achieve fast and reliable results. We demonstrate a framework based on the provided mapping rules, for the scheduling of time-triggered traffic in switched networks. It enables us to implement different scheduling algorithms with little effort and to consider a multiple solver approach. A network can be scheduled by several algorithms in parallel while the best result can be selected. Our experimental evaluation uses an example network taken from the communication patterns of a real-world series car.

The paper is organised as follows. Section 2 elaborates the details of the time-triggered Ethernet protocol, multiprocessor scheduling and scheduling algorithms. Moreover previous and related work are shown. Section 3 describes the network model, the system transformation and the scheduling specifics. In section 4 the details of the implementation of the framework are explained. Section 5 presents scheduling examples and results of a real-world use-case. In section 6 the conclusion and outlook is given.

2 Background & Related Work

2.1 Time-triggered Ethernet

TTEthernet (AS6802) [24] is an extension of the IEEE 802.3 standard and one of a variety of time-triggered Ethernet protocols such as Profinet, or the upcoming IEEE 802.1Qbv [15] standard. TTEthernet adds real-time capabilities to the standard switched Ethernet protocol. It was standardised as AS6802 by the Society of Automotive Engineers (SAE) for the utilisation in control systems of airplanes, automobiles and industrial applications. TTEthernet supports three traffic classes and adds a clock synchronisation protocol to enable time-triggered transmission.

Time-Triggered (TT) traffic class: Time-triggered frames operate according to a coordinated *Time Division Multiple Access* (TDMA). A TT frame has a cyclic behaviour and is transmitted during its statically defined transmission window in its period. A TT frame can be sent as multicast message over several links on a fixed route through the network. Hence, an exclusive transmission window will be defined for each TT frame and each link. To define transmission windows for different frames with different periods a schedule as well as a synchronisation among all participants is required. This ensures a fully deterministic transmission with low latency and minimum jitter. Due to the reserved transmission windows for TT messages, the TT traffic has the highest priority.

Rate-Constrained (RC) traffic class: Similar to TT traffic, rate-constrained traffic is sent as multicast through the network. But instead of defining a timed transmission window for every RC message through a schedule, a certain amount of bandwidth is defined, which can be used for transmission. The traffic shaping is organised using so called Bandwidth Allocation Gaps (BAGs), that define a minimum distance between two consecutive frames of the same stream. This allows to guarantee the bandwidth for an application, which does not require strict timing but highly reliable transmission. Rate-constrained messages are compatible with the ARINC 664 (AFDX) standard [2].

Best-Effort (BE) traffic class: Best-Effort messages have the lowest priority and use the remaining bandwidth. They can be used for tasks without temporal requirements.

This paper focuses on the scheduling of time-triggered messages. Nevertheless, the bandwidth required by RC and BE traffic will be reserved by scheduling additional time slots for these traffic classes. Although the scheduling concepts are currently implemented for TTEthernet the framework and the results presented in this paper are easy transferable to other time-triggered protocols.

2.2 Multiprocessor Scheduling Problem

The scheduling problem which occurs in a multiprocessor environment belongs to the class of combinatorial optimisation problems. It is known as a NP-hard problem [12]. Multiprocessor scheduling has to solve two questions: "On which processor has a task to be executed ?" (allocation problem) and "In which order multiple tasks have to be executed?" (scheduling problem). These problems can be solved by different approaches such as annealing [18], particle swarm optimisation [8], neural networks [3] or heuristic algorithms, e.g. Earliest Deadline First or Rate Monotonic scheduling policies [30].

Such a scheduling problem will be described as a directed acyclic graph (DAG) $G = (V, E)$ – also known as a task graph. It depicts the task dependencies of the set of parallel tasks / processes. The vertices V represent the tasks and the directed edges E the order of execution. The task graph has two dummy vertices, the entry and exit vertices. The entry vertex represents the start point of the program, whereas the exit vertex represents the end of the program. Further parameters like deadlines or execution times can be attached to the DAG.

2.3 Scheduling Heuristics

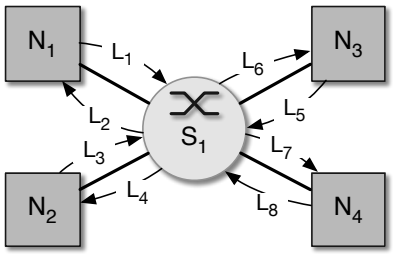
To reduce complexity, we start by using heuristics out of the domain of uniprocessor scheduling. It has been shown, that these heuristics provide mixed results when used in a multiprocessor environment. The quality of the results depend on the given task graph and the used priority parameters [19]. Nevertheless, these heuristics produce feasible schedules with little computation time and can be used as starting point for future improvements. The compared heuristics are:

First Come First Served The processes are executed in the order of their arrival time.

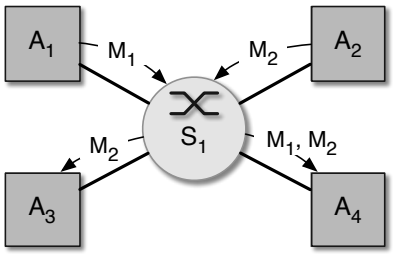
Earliest Deadline First The processes are executed in ascending order of their deadline.

Shortest Job First The processes are executed in ascending order of their execution time.

Longest Job First The processes are executed in descending order of their execution time.



(a) Network with 4 nodes, 1 switch and 4 connections (8 links)



(b) Applications and messages mapped to nodes and links

Figure 1: Model of simple network example

2.4 Related Work

In the specific field of scheduling problems in time-triggered Ethernet, only few research results were presented in the past. Steiner proposes [28] the use of a Satisfiability Modulo Theory (SMT)-solver to create the TT traffic schedule. One step further go Tamas-Selicean et al. [29] by solving the problem with a Tabu Search to schedule TT messages and also optimise the RC traffic.

In the past, there were many proposed solutions [23] [14] to schedule periodic tasks in distributed real-time systems, but they do not regard aperiodic tasks, which is necessary to handle event-triggered messages like the RC traffic in the TTE networks. Furthermore, the approaches are often optimising their schedules towards only one timing metric like the response time [21], which is not applicable to our problem. Instead, we have to optimise towards multiple timing objectives to regard event-triggered traffic. To fulfil these objectives, a new metric is needed.

Fohler [11] describes a solution to schedule preemptive periodic and aperiodic tasks in statically scheduled distributed real-time systems, which uses TDMA based processing and communication nodes to model the system. The static schedule is used to calculate spare capacities. These capacities are used by an online scheduler to determine if a periodic or aperiodic task should be executed next. After every execution, the spare capacities are calculated again. Although the problem is similar to ours, the proposed solution is not applicable. In a TTE network the schedule has to be completly static. Once a TT message is scheduled, it can not be altered. To achieve a fair execution of the aperiodic tasks, all periodic tasks have to be scheduled in such manner that every interval has enough spare capacities in advance.

Our approach transforms the problem of scheduling TTEthernet networks into the domain of multiprocessor scheduling. It reduces the complexity of the system to only one type of processing nodes and removes the communication between the nodes. This allows us to use a wide range of scheduling algorithms to optimise the schedule towards different timing metrics, network topologies and task graph complexities. Furthermore, we are analysing the impact of different gap sizes between two TT messages onto the other traffic classes.

3 System Description

3.1 System & Network Model

The system consists of a set of nodes N, which are interconnected using switched Ethernet. It consists of a set of switches S and a set of directed links L allowing simultaneous transmission and reception via a connection. Figure 1a depicts a simple network with 4 nodes, 1 switch and 4 connections resulting in 8 (directed) links.

The nodes are executing a set of applications A, which communicate using a set of messages M. In the example of figure 1 the application A_1 runs on node N_1 and generates a message M_1. M_1 is transmitted via link L_1, switch S_1 and link L_7 to N_4. Finally, application A_2 running on N_4 consumes message M_1.

Furthermore, a message can be sent to multiple receivers. In figure 1, the message M_2 is generated by an application A_3 and sent to A_4 and A_5. This functionality corresponds to Ethernet multicast and enables the composition of complex control systems. The applications run periodically and feature real-time characteristics. In particular, the consuming application has to finish its execution before a specified deadline. To meet this deadline, the preceding applications and messages have to fulfil corresponding timing constraints.

3.2 System Transformation

For applying scheduling algorithms to the previously defined systems, a system will be mapped to a task graph and a set of processors as follows: The applications and messages of the system are represented by tasks, which are executed by processors. Each node and each directed link are mapped to a separate processor. A processor executes one task at a time. Like an application, which blocks a node for a certain amount of time, a message blocks a link for the time of its transmission. Given that a message has to be transmitted over more than one link, the message has to be split in multiple tasks – one task for each link on the path. In this model a switch acts like a special type of processor. It forwards an incoming message to the corresponding outgoing links. This delays the message for a constant amount of time to regard forwarding delays. A switch can forward multiple messages from different ports at the same time, assuming enough hardware resources in

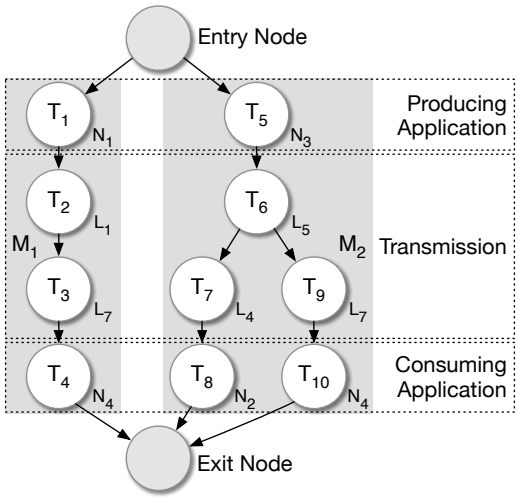


Figure 2: Task graph of two control loops

the switch for the forwarding decision. Usually switches are able to process forwarding decisions at linespeed.

After this transformation, two different sets are left in the model. A set of processors P and a set of tasks T.

$$P = N \cup L \cup S \quad (1)$$
$$T = A \cup M \quad (2)$$

3.3 Task Model

A subset of interacting applications and messages define a *logical control loop* of the system, e.g.: An application produces a message, which is transmitted to one or more receiver nodes and gets consumed by other applications. Considering the example of figure 1, the first control loop consists of 4 tasks: T_1, which represents the producing application A_1, T_2, which represents the transmission of the message M_1 over the first link, T_3, which represents the transmission of M_1 over the second link and T_4, which is the consuming application A_2. The left side of figure 2 represents this control loop. As given on the right side of this figure, the second control loop generates 6 tasks respectively. The subgraphs of all control loops of a system form the task graph.

Following timing properties exist for a task T_i:

Release Time r_i: is the time at which T_i becomes ready for execution

Start Time s_i: is the time at which T_i starts its execution

Computation Time c_i: is the duration of execution of T_i

Finishing Time f_i: is the time at which T_i finishes its execution

$$f_i = s_i + c_i \quad (3)$$

Deadline d_i: is the time before T_i has to finish its execution. A feasible schedule must guarantee

$$f_i \leq d_i \quad (4)$$

Lateness l_i: is the delay of completion of T_i with respect to its deadline

$$l_i = f_i - d_i \quad (5)$$

l_i is negative if a task completes before its deadline

Response Time R_i: is the amount of time, which elapses between release - and start time of a T_i

$$R_i = s_i - r_i \quad (6)$$

Typical deadlines refer to the arrival time of messages at applications and not to the arrival time at components within the network. Hence only tasks belonging to consuming applications feature a deadline. A deadline must be added to the other tasks to apply a scheduling on the task set. These deadlines can be derived from the consuming task: If T_j is the sole direct successor of task T_i ($T_i \rightarrow T_j$), then the deadline d_i can be calculated by:

$$d_i = d_j - c_j \quad (7)$$

If a task T_i has a set of direct successors T_{succ}, then d_i will be calculated as follows

$$d_i = min(\{d_k - c_k \mid T_k \in T_{succ}\}) \quad (8)$$

This step is repeated until every task has a deadline.

3.4 Task To Processor Mapping (Allocation Problem)

The allocation of the applications to the processors is system specific and depends on additional resource constraints. In the majority of cases of automotive designs, an application will be assigned to a specific ECU due to external resources like sensors and actors. Thus, we determine that an application will be mapped to one specific processor within our system transformation.

The tasks of the links are mapped by the used routing algorithm. In the example we use a minimum hop algorithm to define the path from one ECU to another. Since this is a static routing algorithm, the path of a message does not change and the processor for every message task can be defined in advance. This strategy can even be applied to real-time networks that support redundancy. In this case the scheduling must be applied for each path between sender and receiver.

Not all applications depend on external resources. According to the AUTOSAR architecture it would be possible to assign the application to different nodes. To reduce the complexity, we start by assuming that every application has a designated node. Due to this task to processor mapping, the allocation problem of the multiprocessor scheduling approach is solved.

3.5 Scheduling Model

We can describe our system as a static multiprocessor scheduling of periodic non-preemptive tasks with precedence and timing constraints. The tasks of the control loops correspond to the job model in scheduling theory.

Since we have a fixed task to processor mapping, the scheduling of jobs complies with the flow job sequencing problem [6]. Most of the algorithms, which solve the flow job problem try to minimise the overall makespan of the jobs [9]. This optimisation criterion and the corresponding scheduling algorithms do not fit to our model due to the fact that our makespan is defined by the length of the periods of the jobs. But the length of the period is determined by the application and should not be changed. Regarding feasibility and optimality, it does not make a difference if a task is scheduled at the beginning or the end of its period as long as the deadline is met.

With respect to automotive applications the scheduling has to consider the impact on the other traffic classes sent over the same physical layer. RC traffic is event-based and can occur at any time in the period. To minimise the delay of this traffic, transmission gaps must be placed between successive time-triggered messages. For example, if all TT messages would be scheduled at the beginning of a period, an incoming RC message at the beginning of the period would be substantially delayed. Another problem occurs if all gaps between TT messages are too small to transmit a full size RC message. In this scenario a full size RC message can never be transmitted.

To implement these gaps dummy tasks are added to the scheduling, which simulate possible RC frames and their bandwidth. These tasks can be added by different strategies. A strict strategy, which adds a dummy task after every real task and a range based strategy, which adds a minimal number of dummy tasks in a defined timespan.

We propose an optimality criterion which is based on the lateness of a task to respect the deadline constraints. The lateness criterion can be reflected by the maximum lateness

$$L_{max} = max_i(l_i) \tag{9}$$

An additional criterion is the maximum flow response time. The Flow Response Time is the sum of all delays on the path F between sending and receiving node, that are caused by the scheduling. If $F = \{F_1, F_2, \ldots, F_j\}$ is the set of paths defined by $F_j = \{T_1, T_2, \ldots, T_k\}$, the maximum flow response time of these tasks holds to

$$R_{max} = max_i \left(\sum_{n=0}^{|F_i|} R_n(F_i) \right) \tag{10}$$

The maximum lateness and flow response time should be minimised.

In scheduling theory, the problems are noted and classified with the $\alpha|\beta|\gamma$ schema [13]. The characteristics of our scheduling problem holds to the following class

$$F|prec, r_j, d_j|L_{max}, R_{max} \tag{11}$$

4 Framework & Implementation

The presented framework consists of three layers: input layer, processing layer and output layer. The input layer supports two options to provide a system description to the framework. The first is a graphical user interface (GUI), implemented as browser application. The second is an import for Field Bus Data Exchange Format (FIBEX) files. FIBEX [4] is a XML-based format to exchange network data and has been extended to support time-triggered Ethernet protocols in previous work. The processing layer can use a FIBEX file as direct input to schedule the network.

The core unit is the processing layer, which is implemented in Java and consists of four subcomponents. These are the data, routing, algorithm and scheduling components. The data component, reads from the GUI or a FIBEX file to an internal data structure and serves as base for the other components. The routing component models the network topology as a graph. This provides the opportunity to apply graph algorithms to the network for generating the routes of messages. Currently the Dijkstra algorithm is used to find the shortest path between two nodes. Since the network is modelled as a graph, other routing algorithms can be plugged in.

The algorithm component provides an interface to implement the underlying scheduling algorithms. With this consistent interface it is possible to add a new algorithm to the framework with low effort and to realise a multiple solver approach. Further, an interface to implement different strategies to add the transmission gaps between time-triggered frames is available. The algorithm component implements a search tree [7], which captures every scheduling decision. This search tree is used to add backtracking strategies to the algorithms.

The scheduling component uses the data, routing and algorithm components to perform the calculation of the schedule. It uses the data component to model the nodes, links and switches into processors and the applications and messages into tasks. The routing component is used to generate the path of every message, and the mapping of the message tasks to the link processors. After the task set is determined and the paths are set, the task graph is built to resolve the dependencies. Finally, the algorithm component is used to calculate all start times of the tasks.

The output layer transforms the scheduling result into a machine or human readable result. Either it is sent back to the browser and rendered as a scheduling table or transformed back into the FIBEX format. The input and output layer, especially the FIBEX interface, allows to integrate the scheduling framework into other toolchains. E.g., to integrate it into a simulation framework [25] to schedule and simulate a network in one automated step. Further it allows to read and write partly scheduled configurations to enable an incremental scheduling approach.

	Gap	Scheduling Metrics excluding Tasks				Scheduling Metrics including Tasks			
	min	Lateness		Flow Response Time		Lateness		Flow Response Time	
Algorithm	**[Byte]**	**avg [%]**	**max [µs]**	**avg [µs]**	**max [µs]**	**avg [%]**	**max [µs]**	**avg [µs]**	**max [µs]**
First Come First Served	0	−4.11	−4113	307	930	−5.08	−3873	475	1160
	1538	−9.87	−2903	1187	4029	−10.18	−2813	1196	4029
Shortest Job First	0	−4.43	−4183	230	860	−6.01	−3793	392	1340
	1538	−11.06	−1938	1135	2785	−11.63	−1838	1144	2795
Longest Job First	0	−3.33	−4143	317	930	−4.39	−3833	737	1856
	1538	−7.98	−3163	1362	4299	−7.86	−3123	1361	3124
Earliest Deadline First	0	−3.08	−4593	305	860	−3.76	−4460	623	1703
	1538	−6.20	−4460	1305	3459	−6.11	−4370	1369	3619

Table 1: Scheduling Results for the Example In-Car Network with the different Algorithms. Scheduling Metrics are given including and excluding the reception tasks.

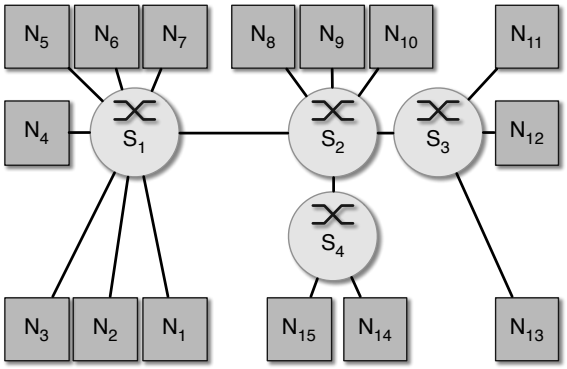


Figure 3: Evaluated In-Car Network

5 Experimental Evaluation

In this section the scheduling framework is evaluated using a realistic network example. Different scheduling algorithms and gap strategies are applied to the network. Afterwards the results are compared and evaluated.

5.1 Example Network & Use-Case

The example network is based on traffic-patterns obtained from a current series car applied to a star based network topology in accordance with todays domain based partitioning [27] (see Figure 3). The network size is reduced to only contain nodes and messages from the network core. Private communication on dedicated networks is neglected. The example consists of 4 switches, 15 nodes and 18 links. The nodes execute 148 time-triggered applications with execution time between 10 µs and 100 µs from which 41 are traffic sources and 107 are traffic sinks. There are 41 time-triggered minimum size messages (64 B) which are transmitted from the producing to the consuming applications. This number corresponds to approx. 5% of signals with cyclic time-critical data and is observed from the traffic model of the chosen car. Applications which rely on rate-constrained or best-effort traffic are not included in the scheduling process and therefore neglected. Their metrics can be obtained from analytical or simulative evaluations after the schedule was generated.

5.2 Parameters

All applications are executed periodically. The cluster period has a length of 1 second, meaning that every application has been executed at least once in this period. In detail every set of producing and consuming applications has its own period. These periods differ from 500 ms to 5 ms, meaning that the applications are executed 2 times to 200 times per cluster period (1 s).

After the transformation there are 55 processors (1 for each switch and node and 2 for each link) and 379 tasks. 148 tasks represent applications, the remaining 231 tasks represent the transmission of messages over the links.

5.3 Results

The schedules were generated on a common desktop computer and took only a couple of seconds. Table 1 shows results of the schedule of the given network. The metrics are exemplarily given for a schedule without gaps for event-triggered traffic (0 B) and a schedule with gaps for at least one fullsize message with all overheads (1538 B) after each TT frame. On the left side of the table the schedule metrics are given without the receiving application, on the right, these tasks are included. The data without the receiving tasks offer a better look on the timing characteristics of the transmission tasks, because the execution times of the network tasks are much shorter than those of the application tasks. This results in long queues on the receiving nodes, which induces a greater average lateness for all tasks.

The maximum lateness, representing the difference between a task scheduled on the receiving node and its deadline, is negative for all cases. Hence the shown configuration always produces a feasible schedule, all deadlines are met. The best result in terms of maximum lateness is achieved with Earliest Deadline First Scheduling (−4593 µs). This complies with the expectations as EDF privileges the task with the first deadline. The maximum lateness for the other algorithms varies insignificantly. The schedule with the latest task in proportion to the deadline is given for Shortest Job First (−1938 µs). When calculating the ratio between runtime and lateness, in average the tasks utilise not more than 3.08 % to 11.06 % of

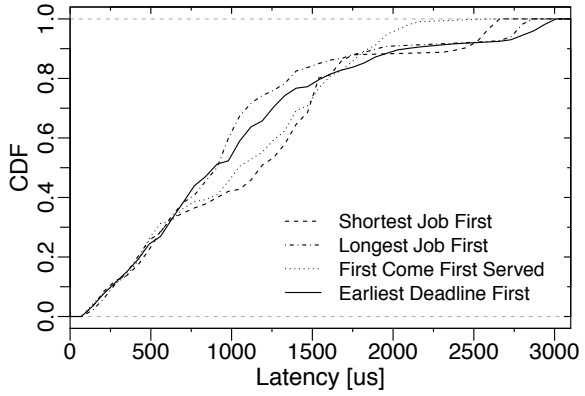


Figure 4: Cumulative Distribution Function (CDF) of End-to-end Latency scheduled with different scheduling algorithms

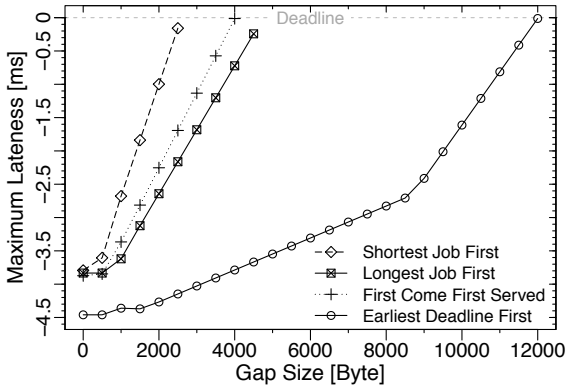


Figure 5: Influence of different Gap Sizes on the Maximum Lateness

the available time.

The Flow Response Time is the sum of all delays on the path between sending and receiving node that are caused by the scheduling. The best results are achieved with the Shortest Job First algorithm that schedules messages with a short distance to its receiving node first (average $230\,\mu s$, maximum $860\,\mu s$). Consequently, the worst average flow response time is scheduled with Longest Job First (average $1362\,\mu s$, maximum $4299\,\mu s$).

When comparing the results without the reception tasks (left) with the metrics for the scheduling including the tasks, the algorithms perform almost equally. The worst-case maximum lateness for schedules without gaps is now observed for the Shortest Job First algorithm ($-3793\,\mu s$) instead of First Come First Served ($-3873\,\mu s$). The worst-case Flow Response Time is now given for the Earliest Deadline First scheduling ($1369\,\mu s$). Due to the computation time at the receiving node the influence of the scheduling gaps decreases. The proportion between the schedule without gaps and the schedule with $1538\,B$ gap is smaller for the scheduling including the tasks.

Figure 4 shows the (empirical) cumulative distribution function (CDF) over the end-to-end latencies. The CDF is a fingerprint of the scheduling algorithm, showing the schedules performance in terms of communication delay. For all algorithms approximately $35\,\%$ of the messages have an end-to-end latency below $700\,\mu s$. Up to this point all algorithms perform equally. For messages between $700\,\mu s$ and $1800\,\mu s$ the algorithms differ in their latency distribution. While Shortest Job First and Earliest Deadline First have more messages with a lower latency, Longest Job First and First Come First Served generate messages with slightly higher delay in this range. Over all messages, First Come First Served produces the smallest, while Earliest Deadline First has the highest maximum end-to-end latency. This corresponds to the expectation as First Come First Served always picks the message that was queued the longest time, neglecting other parameters such as the tasks deadline.

Figure 5 shows the influence of different gap sizes on the maximum lateness. The gaps are included in the schedule to allow for event-triggered messages between scheduled frames. The larger these gaps are, the more event-triggered messages can be placed between time-triggered traffic, resulting in lower delays.

In general there is a linear dependency between the gap size and the maximum lateness. The special characteristics of the network define the gradient. The Earliest Deadline First algorithm performs best when scheduling with large gaps. With gaps of more than $8500\,B$ it becomes more challenging to schedule the network resulting in a higher slope. The example can be scheduled using EDF with gaps of more than $12\,000\,B$. For the other algorithms the schedules become infeasible with smaller gap sizes. The second-best algorithm is Longest Job First, that privileges messages with a long path. Consequently, the worst result is for Shortest Job First using the opposite strategy.

An interesting aspect are the outliers, e.g. for EDF at $1000\,B$. These are caused by the artificially scheduled gaps. When these gaps become larger, the algorithm queues up more tasks resulting in a larger selection and therefore potentially better scheduling decisions.

5.4 Discussion

From the presented real-world example we can conclude, that a time-triggered switched network can easily cope with the amount of synchronous communication in current in-car network designs. The number of messages allows to generate schedules even with simple heuristics. Typically, the computation time of applications is multiple orders of magnitude larger than the transmission. Though, it is important to include all tasks of the system when generating the schedules.

As only part of the in-car network relies on synchronous communication, it is not enough to just find a feasible schedule. It is of great importance to in-

clude mechanisms in the scheduling algorithm that regard bandwidth and transmission gaps for event-triggered messages. These gaps must be chosen in conformance with the message size of concurrent event-triggered real-time streams to minimise wasted bandwidth while achieving low forwarding delays. With increasing size of these gaps, scheduling becomes more challenging. Simple heuristics may not be able to generate feasible schedules anymore. In this case, more sophisticated algorithms, e.g. flow shop scheduling strategies can be used.

The presented results underline the strengths and weaknesses of different scheduling algorithms. As each algorithm typically optimises for a special scheduling metric, there is not an optimum scheduling algorithm. With the presented incremental multiple solver approach, the results of different algorithms can be merged to increase the quality of the schedule.

6 Conclusion & Outlook

Future in-car networks must satisfy the increasing demands of the upcoming advanced driver assistance applications – with high bandwidth sensors such as cameras or laser scanners and the requirement for ultra low latency and jitter – to enable (semi-) autonomous driving. At the same time the demand for high-end multimedia and infotainment, with enormous amounts of data to be transferred, enters the automobile domain. In conjunction with other traffic classes like AVB or RC, time-triggered real-time Ethernet is a good candidate to satisfy all these requirements, offering high bandwidth and deterministic communication. The major challenge in time-triggered Ethernet variants is the design of a suitable schedule. For large networks a schedule cannot be designed by hand.

With this work we contribute an incremental multi solver scheduling approach to automatically generate schedules for time-triggered networks. By mapping the network scheduling problem to the well known domain of multiprocessor scheduling, algorithms and strategies can be transferred and applied on real-time networks. Our real-world example derived from a series car, shows how different algorithms optimise for different scheduling metrics. As a conclusion a multi solver approach that uses different algorithms in parallel can significantly improve the results. Further, the impact of event-triggered traffic on the scheduling is shown. To prevent event-triggered frames from starvation, scheduling must consider transmission gaps for unscheduled traffic. This additional requirement significantly influences the scheduling performance and can lead to infeasible results.

In future work we will attach the scheduling framework to event-based network simulation [25] to analyse the behaviour of event-triggered streams in different schedules under realistic traffic patterns. This will allow to improve the gapping strategies and reduce potential transmission delays for asynchronous applications. Further, we will apply new algorithms for the routing, scheduling and the event-triggered gaps. This will especially include algorithms optimised for the flow shop problem. In relation to the AUTOSAR architecture – that allows at design time to relocate the execution of tasks that do not rely on specific hardware from one processor to another – we dismiss the fixed assignment between tasks and processors and add the allocation problem to the scheduling framework. This significantly increases the scheduling complexity but offers the potential to optimise the network and ECU load.

Acknowledgements

This work is funded by the Federal Ministry of Education and Research of Germany (BMBF) within the RECBAR project.

References

[1] U. Abelein, H. Lochner, D. Hahn, and S. Straube. Complexity, quality and robustness - the challenges of tomorrow's automotive electronics. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 870–871, 2012.

[2] Aeronautical Radio Incorporated. Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network. Standard ARINC Report 664P7-1, ARINC, 2009.

[3] A. H. Ali. Non-preemptive multi-constrain scheduling for multiprocessor with hopfield neural network. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, Aug. 2013.

[4] ASAM - Association for Standardisation of Automation and Measuring Systems. Data Model for ECU Network Systems (Field Bus Data Exchange Format). Specification 4.0.0, ASAM e.V., Sept. 2011.

[5] L. L. Bello. The case for ethernet in automotive communications. *SIGBED Rev.*, 8(4):715, Dec. 2011.

[6] Blazewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., and Weglarz, J. Flow shop scheduling. In *Handbook on Scheduling*, International Handbook on Information Systems, pages 271–320. Springer Berlin Heidelberg, Jan. 2007.

[7] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 2011.

[8] T.-C. Chiang, P.-Y. Chang, and Y.-M. Huang. Multiprocessor tasks with resource and timing constraints using particle swarm optimization. *IJCSNS International Journal of Computer Science and Network Security*, 6(4):7177, 2006.

[9] H. Emmons and G. Vairaktarakis. *Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications*. Springer, Sept. 2012.

[10] FlexRay Consortium. Protocol Specification. Specification 3.0.1, FlexRay Consortium, Stuttgart, Oct. 2010.

[11] G. Fohler. Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. In *, 16th IEEE Real-Time Systems Symposium, 1995. Proceedings*, pages 152–161, Dec. 1995.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[13] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In P. Hammer, E. L. Johnson, and B. H. Korte, editors, *Annals of Discrete Mathematics*, volume Volume 5 of *Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium*, pages 287–326. Elsevier, 1979.

[14] C.-J. Hou and K. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. *IEEE Transactions on Computers*, 46(12):1338–1356, Dec. 1997.

[15] Institute of Electrical and Electronics Engineers. 802.1Qbv - Bridges and Bridged Networks - Amendment: Enhancements for Scheduled Traffic. Draft Standard P802.1Qbv/D1.0, IEEE, Dec. 2013.

[16] International Organization for Standardization. Road vehicles – Controller Area Network (CAN). ISO 11898, ISO, Genf, 2003.

[17] A. Kern, H. Zhang, T. Streichert, and J. Teich. Testing switched ethernet networks in automotive embedded systems. In *2011 6th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 150–155, June 2011.

[18] N. Lotfi and A. Acan. Solving multiprocessor scheduling problem using multi-objective mean field annealing. In *2013 IEEE 14th International Symposium on Computational Intelligence and Informatics (CINTI)*, pages 113–118, 2013.

[19] C. McCreary, A. A. Khan, J. J. Thompson, and M. E. McArdle. A comparison of heuristics for scheduling DAGs on multiprocessors. In *8th Int. Parallel Processing Symposium*, pages 446–451, Apr. 1994.

[20] MOST Cooperation. Media Oriented Systems Transport.

[21] D.-T. Peng, K. Shin, and T. Abdelzaher. Assignment and scheduling communicating periodic tasks in distributed real-time systems. *IEEE Transactions on Software Engineering*, 23(12):745–758, Dec. 1997.

[22] R. Queck. Analysis of ethernet AVB for automotive networks using network calculus. In *2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 61–67, July 2012.

[23] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *IEEE Transactions on Parallel and Distributed Systems*, 6(4):412–420, Apr. 1995.

[24] Society of Automotive Engineers - AS-2D Time Triggered Systems and Architecture Committee. Time-Triggered Ethernet AS6802. SAE Aerospace, Nov. 2011.

[25] T. Steinbach, H. Dieumo Kenfack, F. Korf, and T. C. Schmidt. An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy. In *SIMUTools 2011 – 4th International OMNeT++ Workshop*, pages 375–382, New York, USA, March 21-25 2011. ACM DL.

[26] T. Steinbach, F. Korf, and T. Schmidt. Real-time ethernet for automotive applications: A solution for future in-car networks. In *2011 IEEE International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, pages 216–220, Sept. 2011.

[27] T. Steinbach, H.-T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz. Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802). In *2012 IEEE Vehicular Technology Conference (VTC Fall)*, Piscataway, New Jersey, Sept. 2012. IEEE Press.

[28] W. Steiner. An evaluation of SMT-Based schedule synthesis for time-triggered multi-hop networks. In *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, pages 375–384, Nov. 2010.

[29] D. Tamas-Selicean, P. Pop, and W. Steiner. Synthesis of communication schedules for TTEthernet-based mixed-criticality systems. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '12, page 473482, New York, NY, USA, 2012. ACM.

[30] O. U. P. Zapata and P. M. Alvarez. Edf and rm multiprocessor scheduling algorithms: Survey and performance evaluation. *Seccion de Computacion Av. IPN*, 2508, 2005.