Offloading Safety- and Mission-Critical Tasks via Unreliable Connections

Lea Schönberger
Design Automation for Embedded Systems Group, Faculty of Computer Science, TU Dortmund University, Germany
lea.schoenberger@tu-dortmund.de

Georg von der Brüggen
Design Automation for Embedded Systems Group, Faculty of Computer Science, TU Dortmund University, Germany
georg.von-der-brueggen@tu-dortmund.de

Kuan-Hsun Chen
Design Automation for Embedded Systems Group, Faculty of Computer Science, TU Dortmund University, Germany
kuan-hsun.chen@tu-dortmund.de

Benjamin Sliwa
Communication Networks Institute, Faculty of Electrical Engineering, TU Dortmund University, Germany
benjamin.sliwa@tu-dortmund.de

Hazem Youssef
Chair of Material Handling and Warehousing, Faculty of Mechanical Engineering, TU Dortmund University, Germany
hazem.youssef@tu-dortmund.de

Aswin Karthik Ramachandran Venkatapathy
Chair of Material Handling and Warehousing, Faculty of Mechanical Engineering, TU Dortmund University, Germany
aswin.ramachandran@tu-dortmund.de

Christian Wietfeld
Communication Networks Institute, Faculty of Electrical Engineering, TU Dortmund University, Germany
christian.wietfeld@tu-dortmund.de

Michael ten Hompel
Chair of Material Handling and Warehousing, Faculty of Mechanical Engineering, TU Dortmund University, Germany
michael.tenHompel@tu-dortmund.de

Jian-Jia Chen
Design Automation for Embedded Systems Group, Faculty of Computer Science, TU Dortmund University, Germany
jian-jia.chen@tu-dortmund.de

Abstract

For many cyber-physical systems, e.g., IoT systems and autonomous vehicles, offloading workload to auxiliary processing units has become crucial. However, since this approach highly depends on network connectivity and responsiveness, typically only non-critical tasks are offloaded, which have less strict timing requirements than critical tasks. In this work, we provide two protocols allowing to offload critical and non-critical tasks likewise, while providing different service levels for non-critical tasks in the event of an unsuccessful offloading operation, depending on the respective system requirements. We analyze the worst-case timing behavior of the local cyber-physical system and, based on these analyses, we provide a sufficient schedulability test for each of the proposed protocols. In the course of comprehensive experiments, we show that our protocols have reasonable acceptance ratios under the provided schedulability tests. Moreover, we demonstrate that the system behavior under our proposed protocols is strongly dependent on probability of unsuccessful offloading operations, the percentage of critical tasks in the system, and the amount of offloaded workload.

2012 ACM Subject Classification Computer systems organization → Real-time systems

Keywords and phrases internet of things, cyber-physical systems, real-time, mixed-criticality, self-suspension, computation offloading, scheduling, communication

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2020.18

Funding This work is supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876, projects A1, A3, A4, and B4.

Acknowledgements The authors thank Jui-Lin Liang for his support and Niklas Ueter for his valuable feedback.

1 Introduction

Processing large amounts of sensor data within short, pre-defined intervals of time is crucial for many cyber-physical systems as, e.g., IoT systems, robots, drones, and autonomous vehicles, in order to accomplish their mission or even to maintain their operability. However, it may be the case that a system does not have sufficient resources at its disposal to always perform the necessary operations fast enough and to deliver the required results in time if these computations are performed merely locally. Since the system hardware can only be enhanced up to a certain level due to constraints in terms of cost, energy efficiency, size, and other noteworthy factors, such insufficiencies can be overcome by offloading a share of the system workload via, e.g., 4G/5G or IEEE802.11p-based [18] wireless connections, while providing a local fallback mechanism. Nevertheless, wireless connections exhibit a certain level of unreliability, which must be factored in when deciding which tasks to offload. As an example, the resulting end-to-end performance within cellular vehicular communication systems is severely impacted by highly dynamic channel conditions related to shadowing effects, multipath fading, handover situations, and even technology switches [24]. In addition, as the available cell resources are shared by the different network participants, the achievable network performance significantly depends on the traffic patterns of the other active cell users and the implemented resource scheduling policy of the cell. Different network quality indicators can be utilized to estimate the end-to-end behavior of data transmissions in terms of delay and data rate. However, as pointed out in recent analyses [25], the interdependencies of these factors can significantly differ among multiple mobile network operators due to varying strategies for network configuration and infrastructure deployment.

Against this background, we aim to allow resource-constrained systems to offload computation shares not only of non-criticial, but also of safety- and mission-critical tasks, while ensuring that the timing requirements of safety- and mission-critical tasks are not violated even in the case of connectivity issues and providing as much service for non-critical tasks as possible. Accordingly, we strive to specify the system's offloading behavior in such a way that it can be verified at design time for all potential scenarios.

Self-Suspension. From a modeling perspective, the considered cyber-physical system can be reduced to the local system, on which the performed offloading operations are perceived as tasks being executed, paused, and (in the successful case) resumed after an upper-bounded interval of time. One concept allowing to model this particular local system view is *self-suspension* [7], which characterizes tasks temporarily interrupting their execution and proceeding as soon as a certain operation is finished, i.e., as soon as a response from the auxiliary processing unit is received. In fact, the actual time elapsing between the moment a message is sent to an auxiliary processing unit and the latest safe moment in which a response may be received could be simply modeled as additional computation time rather than as so-called *suspension time*, but this would lead to a pessimistic under-utilization of computation resources [23, 28]. Instead, for the sake of accuracy, one of the state-of-the-art models can be applied such as the dynamic self-suspension model (cf. e.g. [12], [15]), the segmented self-suspension model (cf. e.g. [22]), or a hybrid model, e.g. [27], (for a detailed overview refer to [6, 7]). In this work, we make use of the *segmented self-suspension model*, which allows to precisely depict a specific suspension pattern, i.e., to specify the exact point in time in which an offloading operation starts as well as a legal upper bound on its duration (formal descriptions will be given in Sec. 2).

Mixed-Criticality. Modeling a local system as well as successful offloading operations, however, does not suffice as a basis for verifying the behavior of a cyber-physical system as considered in this work. De facto, the question remains, how to model and how to handle unsuccessful offloading operations. To this effect, we benefit from the notion of so-called *mixed-criticality systems*, which were formally introduced for the first time by Vestal [26] and received much attention thenceforward (a comprehensive survey can be found in [5]). This concept describes systems integrating tasks with different *criticality levels* on the same platform and providing distinct system modes, usually one per criticality level. In case of special events such as fault-occurrence, mixed-criticality systems perform a *mode change*, i.e., switch to another system mode, which permits to maintain the system safety by ensuring that the timing requirements of all tasks corresponding to the current or a higher system mode can still be met. For all lower-criticality tasks, however, no more timing guarantees are provided. Analogously to mode changes in mixed-criticality systems, it is necessary to carefully anticipate all events that may occur during an offloading operation, e.g., a missing response due to an unreliable wireless connection, and to specify mechanisms to handle these deterministically. However, we do *not* model the contemplated type of cyber-physical systems as a mixed-criticality system, but rather exploit the characteristics exhibited by the latter. Namely, we classify the overall set of tasks in the system into a set of *critical tasks* (comprising safety- as well as mission-critical tasks) and a set of *non-critical tasks*, but we do not specify explicit system modes. Instead, we consider the system to exhibit different *execution behaviors* under different circumstances: either a *normal* execution behavior, under which timing constraints are satisfied for all tasks, or a *local* one, under which timeliness is only guaranteed for critical tasks. The actual system behavior in each possible scenario, however, needs to be clearly defined and to follow a pre-specified, analyzable, and verifiable protocol, which will be developed hereinafter.

Related Work. The idea of cloud-based control for automotive systems is not new, but has already been addressed in 2012 by Kumar et al. [14], who proposed a cloud-assisted system for autonomous driving, which, however, is not used to offload control applications, but rather to provide additional information to the vehicle. In 2015, Esen et al. [9] presented a software architecture named *Control as a Service* according to which all control functions are completely moved to the cloud, while only sensors, actuators, and communication infrastructure remain in the vehicle. Network latencies have been pointed as a challenge, but no concrete solution or mechanism to handle suchlike has been proposed, and, moreover, connection losses have not been addressed at all. In a proof of concept, the authors have modeled network latencies using discrete stochastic models. In 2018, Adiththan et al. [1] proposed an adaptive offloading technique for control applications that makes all offloading decisions online based on a network performance monitor. However, due to the heuristic nature of the approach, the timing behavior of the system cannot be verified. Beyond that, the authors mentioned the necessity to handle connectivity losses and large communication delays and stated that in such cases task executions must be redirected to the local system. Nevertheless, no concrete mechanism or protocol has been suggested for this purpose. Moreover, the potential consequences have not been further discussed. Recently, Al Maruf and Azim [17] proposed a strategy for task offloading in multiprocessor mixed-criticality systems with dynamic scheduling policies under overload conditions. More precisely, they suggested to select low-criticality tasks based on a machine learning approach that are offloaded to the cloud and executed in parallel aiming to reduce the number of deadline misses. Potential connectivity issues have not been taken into consideration.

Contributions. In a nutshell, we contribute the following:

- We provide two protocols for cyber-physical systems allowing to safely offload critical and non-critical tasks, which address different system requirements: i) the *service protocol* provides as much service for non-critical tasks as possible in any point in time, and ii) the *return protocol* allows a fast return to the normal system behavior in the case of an unsuccessful offloading operation.
- In Sec. 4 and Sec. 5, we analyze the worst-case timing behavior of the local system and examine our proposed protocols by considering all possible transient and ready states regarding the normal and the local execution behavior. Based on these analyses, we provide a sufficient schedulability test for each of the proposed protocols.
- By means of comprehensive simulations and a case study, we i) show that the acceptance ratios of our proposed protocols under the provided schedulability tests are reasonable, and ii) demonstrate that the system behavior under our proposed protocols is strongly dependent on probability of unsuccessful offloading operations, the percentage of critical tasks in the system, and the amount of offloaded workload.

2 System Model

We consider a cyber-physical system comprising a set of tasks $\mathcal{T}$ that can be divided into two subsets with different requirements, namely, the set of *critical* tasks $\mathcal{T}_{crit}$, and the set of *non-critical* tasks $\mathcal{T}_{non}$, such that $\mathcal{T} = \mathcal{T}_{crit} \cup \mathcal{T}_{non}$ and $\mathcal{T}_{crit} \cap \mathcal{T}_{non} = \emptyset$. While for each $\tau_k \in \mathcal{T}_{crit}$ timing constraints must be satisfied at any point in time, for each $\tau_k \in \mathcal{T}_{non}$ timing violations may be unpleasant but not hazardous. According to the classification of tasks into two subsets, we specify two different system execution behaviors, i.e., *normal* and *local* execution behavior. When the system exhibits normal execution behavior, all timing requirements of all tasks are satisfied at any point in time, whereas, if the system exhibits local execution behavior, timing guarantees can only be given for all critical tasks $\tau_k \in \mathcal{T}_{crit}$. The degree of service provided with respect to the non-critical tasks $\tau_k \in \mathcal{T}_{non}$ depends on the particular recovery protocol implemented in the system (cf. Sec. 3).

Each recurrent real-time task $\tau_k \in \mathcal{T}$ in the considered cyber-physical system is assumed to have a sporadic arrival pattern and is characterized[1] by a tuple $(C_{k,1}, C_{k,s}, C_{k,2}, S_k, p_k, q_k, D_k, T_k)$:

- Each τ_k releases an infinite number of task instances denoted as *jobs*. T_k indicates the minimum inter-arrival time of τ_k, i.e., the arrival times of any two consecutive jobs of τ_k must be separated by at least T_k.
- D_k describes the relative deadline of τ_k. The absolute deadline of a job of task τ_k arriving at time r_k is given by $r_k + D_k$ if *it must be guaranteed that the job meets its deadline*. Throughout this paper, we assume a constrained-deadline task system, in which $D_k \leq T_k$ for each task τ_k.
- $C_{k,1}$ and $C_{k,2}$ denote the worst-case execution times of the first and second *computation segments*, respectively.
- $C_{k,s}$ is the worst-case execution time of the typically offloaded task share if executed on the *local* system.
- p_k and q_k are the worst-case execution times of the pre- and post-processing routines, which are executed before and after the offloading operation of a job of task τ_k, respectively.
- S_k is the offloading or *suspension* time of τ_k.

[1] To provide a better overview, the notation is additionally summarized in Table 1.

Table 1 An overview about the notation.

Notation	Meaning
$C_{k,1}$	worst-case execution time (WCET) of the first computation segment of τ_k
$C_{k,2}$	WCET of the second computation segment of τ_k
$C_{k,s}$	WCET of the computation segment of τ_k that is typically offloaded if executed locally
$C_k^\flat$	$C_{k,1} + p_k + q_k + C_{k,2}$
$C_k^\sharp$	$C_{k,1} + S_k + C_{k,2}$
D_k	relative deadline of τ_k
$hp(\tau_k)$	set of tasks with higher priority than τ_k
p_k	WCET of the offloading pre-processing routine of τ_k
q_k	WCET of the offloading post-processing routine of τ_k
S_k	offloading/suspension time of τ_k
T_k	minimum inter-arrival time of τ_k
$\mathcal{T}$	complete task set
$\mathcal{T}_{crit}$	set of critical tasks
$\mathcal{T}_{non}$	set of non-critical tasks

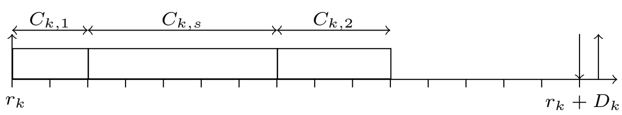


Figure 1 A job of task τ_k is executed locally (local execution behavior).

We assume that $T_k \geq D_k > 0$ and $C_{k,1}, C_{k,s}, C_{k,2}, S_k, p_k, q_k \geq 0$. Moreover, we make the natural assumption that $p_k + q_k \leq C_{k,s}$, since offloading is not meaningful otherwise. Furthermore, the worst-case execution time of a job of task τ_k under any possible execution scenario is greater than 0, i.e., $C_{k,1} + C_{k,s} + C_{k,2} > 0$ and $C_{k,1} + p_k + q_k + C_{k,2} > 0$. For notational brevity, we denote $C_k^\sharp = C_{k,1} + C_{k,s} + C_{k,2}$ and $C_k^\flat = C_{k,1} + p_k + q_k + C_{k,2}$.

Throughout this paper, we assume that the local cyber-physical real-time system, termed *local system*, is a uniprocessor system, in which tasks are scheduled according to a preemptive fixed-priority policy. More precisely, each task is assigned a unique priority, i.e., all jobs of task τ_k have the same priority. If at any point in time multiple jobs are ready, i.e., eligible for being executed on the local system, the job having the highest priority is executed. For each task τ_k, the unique set of the higher-priority tasks is denoted as $hp(\tau_k)$.

For a job of task τ_k arriving at time r_k the following execution scenarios are possible:

- The job is *executed locally* (cf. Fig. 1). In this case, the worst-case execution time of the job released at time r_k is $C_{k,1} + C_{k,s} + C_{k,2}$, i.e., $C_k^\sharp$.
- The job is *offloaded*. In this case, the job is first executed locally for up to $C_{k,1}$ execution time units and thereon enters the pre-processing routine for offloading for up to p_k execution time units. Suppose that the first computation segment as well as the pre-processing routine are finished at time ρ. Then, the considered job is offloaded to the remote system at time ρ. The actual offloading operation can be either *successful* or *unsuccessful*:
  - *Offloading is successful* if the computation result or *offloading response* is returned to the local system until time $\rho + S_k$. In this case, the offloading response is post-processed for up to q_k time units and the second computation segment is executed for up to $C_{k,2}$

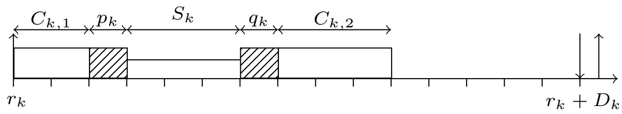


Figure 2 An offloading operation of a job of task τ_k is performed successfully (normal execution behavior).

time units (cf. Fig. 2). Accordingly, the execution time of the job of τ_k on the local system is at most $C_k^{\flat}$.

- *Offloading is unsuccessful* otherwise. In this case, at time $\rho + S_k$, a local re-execution of the offloaded task share is performed for up to $C_{k,s}$ time units followed by the execution of the second computation segment for up to $C_{k,2}$ time units. In this case, the execution time of the job of τ_k on the local system is at most $C_k^{\sharp} + p_k$. This scenario will be discussed more in detail hereinafter.

3 Recovery Protocols

Cyber-physical systems can be encountered throughout a broad range of application areas, each exhibiting individual requirements and thus a need for situationally appropriate system behavior. For safety-critical cyber-physical systems, the timeliness of critical tasks must be guaranteed under any circumstances - even in the event of an unsuccessful offloading operation. Since in this case a larger amount of local resources is required, as explained with respect to the possible execution scenarios in Sec. 2, less resources remain to serve the non-critical tasks. However, depending on the actual system characteristics, timing constraints for non-critical tasks tend to be less strict. It is, for instance, possible that a non-critical task misses its deadline, but the results are still useful up to a certain degree [4, 3]. Nevertheless, it may be desirable to return to the normal execution behavior and to re-establish timing guarantees for both critical and non-critical tasks as soon as possible, especially since a non-critical task is not necessarily unimportant and thus should provide functionally and temporally correct results most of the time (further discussion on the relation between criticality and importance can be found in [10]).

Against this backdrop, we propose two recovery protocols allowing the system to satisfy its requirements under local execution behavior and to return to normal execution behavior:

- The *service protocol* aims to provide as much service as possible for non-critical tasks, even under local execution behavior.
- The *return protocol* aims to minimize the amount of time, in which the system exhibits local execution behavior after an unsuccessful offloading operation.

Independent of the actual protocol, we assume that the local system exhibits normal execution behavior at time 0, such that offloading is enabled for all tasks in $\mathcal{T}$. The schedule considers the execution of all tasks until the first moment $\gamma_{1,\searrow}$, in which the offloading operation of a certain task τ_k is unsuccessful, i.e., a job of task τ_k, which has offloaded its computation at time $\gamma_{1,\searrow} - S_k$, does not receive the offloading response until time $\gamma_{1,\searrow}$ (cf. Fig. 3). Immediately after $\gamma_{1,\searrow}$, the local system exhibits local execution behavior. Until time $\gamma_{1,\searrow}$, three scenarios are possible for each incomplete job of all critical tasks τ_i in $\mathcal{T}_{crit}$:

- *The job of τ_i has not been offloaded*: In this case, no offloading operation will be performed for this job, but it is executed locally instead. Since it is possible that the pre-processing routine for offloading is already active at time $\gamma_{1,\searrow}$, the worst-case execution time of this job is upper-bounded by $C_{i,1} + p_i + C_{i,s} + C_{i,2}$, i.e., $C_i^{\sharp} + p_i$.

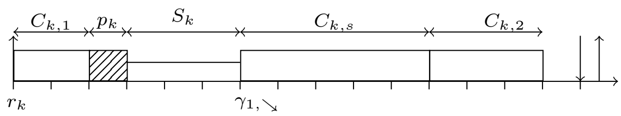


Figure 3 An unsuccessful offloading operation of τ_k resulting in the transition to the local system behavior at time $\gamma_{1,\searrow}$.

- *The job of τ_i is already offloaded, but no offloading response was received until time* $\gamma_{1,\searrow}$: In this case, the offloading process is aborted and the job is executed locally as of time $\gamma_{1,\searrow}$. Therefore, the worst-case execution time of this job is upper-bounded by $C_{i,1} + p_i + C_{i,s} + C_{i,2}$, i.e., $C_i^\sharp + p_i$.
- *The job of τ_i is already offloaded and the offloading response has been received prior to time* $\gamma_{1,\searrow}$: In this case, the job continues its final processing. Therefore, the worst-case execution time of this job is upper-bounded by $C_{i,1} + p_i + q_i + C_{i,2}$, i.e., $C_i^\flat$.

After $\gamma_{1,\searrow}$, timing guarantees are only provided for $\mathcal{T}_{crit}$. Moreover, offloading is inhibited for all critical tasks in the near future of $\gamma_{1,\searrow}$, due to the currently unreliable connection leading to the missing offloading response. The offloading decision for non-critical tasks, however, depends on the applied recovery protocol:

- **Service Protocol**: Under the *service protocol*, offloading is inhibited for all instances of all tasks that are active as long as the system exhibits local execution behavior. The task share of each $\tau_i \in \mathcal{T}$ that is offloaded under normal execution behavior is executed locally within $C_{i,s}$ units of execution time. Since this leads to a higher workload on the local system, timeliness cannot be guaranteed for any non-critical task. Nevertheless, no non-critical task is aborted.
- **Return Protocol**: The *return protocol* does not inhibit offloading for *all* tasks, but only for critical ones under local execution behavior. Non-critical tasks, in contrast, are offloaded regardless, but neither a re-execution nor a re-transmission is performed if an offloading response is not received in time. More precisely, the second subtask of τ_i is only executed if an offloading response is received, and aborted otherwise. Moreover, a job of τ_i in $\mathcal{T}_{non}$ is aborted whenever it misses its deadline.

As of time $\gamma_{1,\searrow}$, the local system exhibits local execution behavior until the point in time $\gamma_{1,\nearrow}$, in which timing guarantees can be given again for all tasks in $\mathcal{T}$. In the proposed protocols, two options are considered for the transit from local to normal execution behavior, which should be chosen depending on the actual system requirements:

- **Abort-Transit**: This option aims to re-establish the normal system execution behavior as quickly as possible. Suppose that $\gamma_{1,\nearrow}$ is the earliest moment (after $\gamma_{1,\searrow}$) in which there is no incomplete job from $\mathcal{T}_{crit}$ at $\gamma_{1,\nearrow}$. All released but not yet finished instances of non-critical tasks are discarded.
- **Idle-Transit**: This option re-establishes the normal system execution behavior at the earliest moment $\gamma_{1,\nearrow}$ (after $\gamma_{1,\searrow}$) in which there is no incomplete job from $\mathcal{T}$ at $\gamma_{1,\nearrow}$.

We note that the above transitions are well-defined and the local system exhibits normal and local execution behavior in an interleaving manner.

4 Existing Analysis and Workload Characteristics

When the system exhibits normal execution behavior, the same task execution patterns are identifiable under both proposed protocols, i.e., an offloading operation is performed for each task. Hence, each $\tau_k \in \mathcal{T}$ is a (segmented) self-suspending task consisting of two computation segments as well as of one *suspension interval* of length S_k, and can therefore be analyzed applying any suitable technique.

▶ **Definition 1.** *Suppose that the system always exhibits normal execution behavior. Then, for each task $\tau_k \in \mathcal{T}$, the worst-case response time R_k^{normal} is the worst-case response time of task τ_k and R_k^1 is the worst-case response time of the first computation segment of task τ_k. By definition, $R_k^1 \leq R_k^{normal}$. This paper assumes that $R_k^{normal} \leq D_k \leq T_k$, $\forall \tau_k \in \mathcal{T}$.*

▶ **Lemma 2.** *If τ_k is in $\mathcal{T}_{non}$, the worst-case response time of τ_k under normal execution behavior is upper-bounded by R_k^{normal} regardless of the adopted protocol.*

Proof. This is based on the definition. ◀

Regarding the analysis of self-suspending tasks, several misconceptions exist in the literature. Detailed and correct treatments can be found in a recent survey paper by Chen et al. [6]. The latest result was developed by Schönberger et al. in [22]. However, instead of going into detail regarding the analysis of uniprocessor segmented self-suspending task systems, we assume that one of the existing analyses has been used and each task in $\mathcal{T}$ has been validated to meet its deadline if the system *always exhibits normal execution behavior* by applying the analysis given in [22].

The following lemma characterizes the maximum workload of a task τ_i that can be executed in a time interval $[t, t+\Delta)$ under the assumption that the local system resumes from idling at time t, i.e., it idles at $t-\varepsilon$ for an infinitesimal ε, under normal execution behavior and it does not switch from the local execution behavior to the normal execution behavior before $t+\Delta$ for $\Delta > 0$.

▶ **Lemma 3.** *Suppose that the local system resumes from idling at time t, i.e., it idles at $t-\varepsilon$ for an infinitesimal ε and executes a certain job at time t, under normal execution behavior and it does not switch from the local execution behavior to the normal execution behavior before $t+\Delta$ for $\Delta > 0$. For a task τ_i, in which*

- *τ_i is in $\mathcal{T}$ under the service protocol or*
- *τ_i is in $\mathcal{T}_{crit}$ under the return protocol,*

the amount of execution time for which task τ_i is executed in time interval $[t, t+\Delta)$ on the local system is upper-bounded by $\max\{f_1(\tau_i,\Delta), f_2(\tau_i,\Delta)\}$, where

$$f_1(\tau_i,\Delta) = p_i + \left\lceil \frac{\Delta}{T_i} \right\rceil C_i^\sharp \tag{1}$$

and

$$f_2(\tau_i,\Delta) = C_{i,s} + C_{i,2} + \left\lceil \frac{\Delta - (T_i - (R_i^1 + S_i))}{T_i} \right\rceil C_i^\sharp \tag{2}$$

Recall that $C_i^\sharp$ is defined as $C_{i,1} + C_{i,s} + C_{i,2}$.

Proof. We first consider the simpler case, in which the local system stays in the normal execution behavior from t to $t+\Delta$. In this case, there are two scenarios:

- Case 1a: if τ_i does not have any unfinished job before t (cf. Fig. 4), then, the workload of task τ_i executed in time interval $[t, t+\Delta)$ is at most $\left\lceil \frac{\Delta}{T_i} \right\rceil C_i^\flat \leq f_1(\tau_i,\Delta)$.

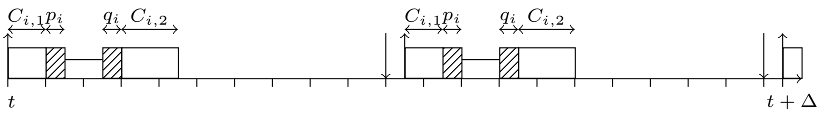


Figure 4 Execution of a task τ_i under analysis in $[t, t+\Delta)$ in case 1a of Lemma 3.

- Case 1b: If τ_i does have an unfinished job before t, then, by the definition that the local system returns from idling at time t, task τ_i has been suspended (cf. Fig. 5). Since the system still exhibits normal execution behavior prior to time t, we know that there is at most one such suspended job of τ_i and its arrival time r_i cannot be earlier than $t - (R_i^1 + S_i)$. Therefore, the first job of task τ_i released after t is released no earlier than $t - (R_i^1 + S_i) + T_i$. Since the system exhibits normal execution behavior from t to $t + \Delta$, the workload of task τ_i executed in time interval $[t, t + \Delta)$ is at most $q_i + C_{i,2} + \left\lceil \frac{\Delta - (T_i - (R_i^1 + S_i))}{T_i} \right\rceil C_i^\flat \leq f_2(\tau_i, \Delta)$.

We then consider another case, in which the local system switches to local execution behavior at time γ, where $t \leq \gamma < t + \Delta$. There are also two scenarios:

- Case 2a: There is no job of τ_i arriving before t that has not been finished yet by time t (cf. Fig. 6). From t to γ, at most $\left\lceil \frac{\gamma - t}{T_i} \right\rceil$ jobs of task τ_i are released. Specifically, among them, the last job of τ_i offloaded prior to γ may be offloaded unsuccessfully. For this particular job, its worst-case execution time is $C_{i,1} + p_i + C_{i,s} + C_{i,2} = C_i^\sharp + p_i$, whilst the worst-case execution time of the other $\left\lceil \frac{\gamma - t}{T_i} \right\rceil - 1$ jobs is at most $C_i^\flat \leq C_i^\sharp$. Moreover, for any job of τ_i released after γ, under the service protocol or the return protocol when $\tau_i \in \mathcal{T}_{crit}$, its worst-case execution time is at most $C_i^\sharp$. Therefore, the workload of τ_i from t to $t + \Delta$ is

$$\left(\left(\left\lceil \frac{\Delta}{T_i} \right\rceil - 1\right) C_i^\sharp\right) + (C_i^\sharp + p_i) \leq \quad p_i + \left\lceil \frac{\Delta}{T_i} \right\rceil C_i^\sharp \quad \stackrel{\text{def}}{\equiv} \quad f_1(\tau_i, \Delta)$$

- Case 2b: A job of τ_i suspended prior to t and its second computation segment is released at or after t (cf. Fig. 7). Identically to the discussion in Case 1b, the next job of task τ_i arrives no earlier than $t - (R_i^1 + S_i) + T_i$. If the job suspended prior to t is offloaded unsuccessfully, i.e., its execution time after t is at most $C_{k,s} + C_{k,2}$, the other subsequent jobs of τ_i will be executed only locally, until the moment in which the local system switches to the normal execution behavior again, under the service protocol or under the return protocol when τ_i is in $\mathcal{T}_{crit}$. Therefore, the workload of τ_i from t to $t + \Delta$ is as defined in $f_2(\tau_i, \Delta)$. If the job suspended prior to t is offloaded successfully, at most of one of the jobs released after $t - (R_i^1 + S_i) + T_i$ is offloaded unsuccessfully. In this case, the workload of τ_i from t to $t + \Delta$ is at most

$$q_i + C_{i,2} + p_i + \left\lceil \frac{\Delta - (T_i - (R_i^1 + S_i))}{T_i} \right\rceil C_i^\sharp \leq f_2(\tau_i, \Delta),$$

since $p_i + q_i \leq C_{i,s}$. ◀

So far, the maximum workload that can be contributed to a time interval $[t, t + \Delta)$ by a task $\tau_i \in \mathcal{T}$ under the service protocol and by a task $\tau_i \in \mathcal{T}_{crit}$ under the return protocol has been analyzed. For the missing case that τ_i is in $\mathcal{T}_{non}$ under the return protocol, the workload in the time interval $[t, t + \Delta)$ can be reduced based on the definition of the protocol (cf. Sec. 3), as given in the following lemma:

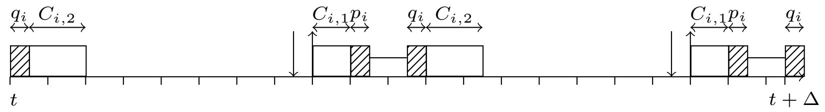


Figure 5 Execution of a task τ_i under analysis in $[t, t+\Delta)$ in case 1b of Lemma 3.

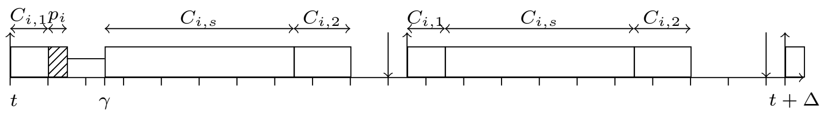


Figure 6 Execution of a task τ_i under analysis in $[t, t+\Delta)$ in case 2a of Lemma 3.

▶ **Lemma 4.** *For a task τ_i in $\mathcal{T}_{non}$ under the return protocol, under the same condition for t and $t+\Delta$ as specified in Lemma 3, the amount of execution time that task τ_i is executed in the time interval $[t, t+\Delta)$ is upper-bounded by $\left(\left\lceil \frac{\Delta}{T_i} \right\rceil + 1\right) C_i^\flat$.*

Proof. Under the return protocol, a job of task τ_i in $\mathcal{T}_{non}$ is aborted whenever it misses its deadline. Therefore, the number of jobs of τ_i, which have not yet missed their deadlines in a time interval $[t, t+\Delta)$, is at most $\left(\left\lceil \frac{\Delta}{T_i} \right\rceil + 1\right)$ since $D_i \leq T_i$. Under the return protocol, a task τ_i in $\mathcal{T}_{non}$ is always offloaded if the first computation segment is completed before the job's deadline. Any execution of such a job on the local system requires up to $C_i^\flat$ execution time units by definition. Therefore, $\left(\left\lceil \frac{\Delta}{T_i} \right\rceil + 1\right) C_i^\flat$ is the upper bound of the workload. ◀

5 Timing Analysis

To analyze the worst-case timing behavior of the local system, the timing behavior of each task τ_k must be analyzed beginning with the highest-priority task. In the course of this, two scenarios need to be analyzed:

- If τ_k is in $\mathcal{T}_{crit}$, the worst-case response time under local and normal execution behavior must be analyzed.
- If τ_k is in $\mathcal{T}_{non}$, only the worst-case response time under normal execution behavior must be analyzed.

We note that the worst-case response time of τ_k in $\mathcal{T}_{non}$ under local execution behavior is not of interest, since its jobs may be aborted (cf. Sec. 3).

In our analysis, we assume a concrete fixed-priority preemptive schedule σ for the task set $\mathcal{T}$ from time 0 onwards. For the concrete schedule σ, let $\gamma_{h,\searrow}$ be the h-th moment in which σ switches from normal to local execution behavior. Moreover, let $\gamma_{h,\nearrow}$ be the h-th moment in which σ switches from the local behavior to the normal behavior. By the definition of our protocols (cf. Sec. 3), $\gamma_{h,\searrow} < \gamma_{h,\nearrow} < \gamma_{h+1,\searrow}$.

We consider the j-th job of task τ_k, denoted as τ_k^j, in schedule σ and assume that at the arrival time of job τ_k^j there exists no incomplete job of task τ_k in the schedule σ. Now, we

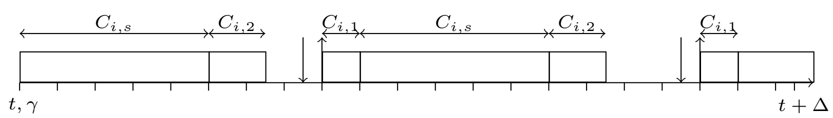


Figure 7 Execution of a task τ_i under analysis in $[t, t+\Delta)$ in case 2b of Lemma 3.

remove all lower-priority jobs from the schedule σ. Since σ is a fixed-priority preemptive schedule and all tasks in $\mathcal{T}$ are independent from each other, the removal of these jobs does not have any impact on the execution of any remaining job in the schedule σ. Thereon, we remove all jobs of task τ_k having arrived before the release of τ_k^j at time r_k^j from the schedule σ. Due to the assumption that the jobs of τ_k released before r_k^j have been finished before r_k^j, the removal of these jobs of τ_k does not have any impact on the execution of any remaining job in the schedule σ.

For simplicity of notation, we remove the index j for the rest of the proof, when the context is clear. Accordingly, r_k denotes the arrival time of the job of τ_k under analysis and f_k its finishing time. By definition, the next job of τ_k cannot arrive before $r_k + T_k$.

For the rest of this section, we will focus on the analysis of the case that τ_k is in $\mathcal{T}_{crit}$.

▶ **Lemma 5.** *Under both service and return protocol as well as both abort- and idle-transit, for any $\tau_k \in \mathcal{T}_{crit}$, in the interval $[r_k, f_k)$, the local system switches at most once from normal to local behavior. That is, at most one $\gamma_{h,\searrow}$ exists in $[r_k, f_k)$.*

Proof. This property results from the definition of the protocols and abort- and idle-transits. That is, the local system only switches from the local to normal execution behavior when there is no unfinished job of $\mathcal{T}_{crit}$. ◀

Based on Lemma 5, only four cases need to be considered:

- σ is executed under local execution behavior at time r_k and under normal execution behavior at time f_k, denoted as *L2N*.
- σ is executed under normal execution behavior at time r_k and under normal execution behavior at time f_k, denoted as *N2N*.
- σ is executed under normal execution behavior at time r_k and under local execution behavior at time f_k, denoted as *N2L*.
- σ is executed under local execution behavior at time r_k and under local execution behavior at time f_k, denoted as *L2L*.

In the following, we examine each of these cases individually, beginning with those that do not depend on the implemented recovery protocol, i.e., L2N and N2N. Thereon, the remaining cases are considered first under the service protocol in Sec. 5.1 and consecutively under the return protocol in Sec. 5.2.

▶ **Lemma 6.** *The case L2N is not possible under Abort-Transit and Idle-Transit.*

Proof. In both transitions from local to normal execution behavior, no incomplete job exists in the local system at time $\gamma_{h,\nearrow}$ for any $h \geq 0$. ◀

▶ **Lemma 7.** *The response time $f_k - r_k$ in the case N2N is at most R_k^{normal}, as defined in Definition 1.*

Proof. This is identical to self-suspension task systems. Suppose that $\gamma_{h,\nearrow} \leq r_k < \gamma_{h,\searrow}$. We can remove all the jobs in the schedule σ before $\gamma_{h,\nearrow}$ without changing any execution in σ after $\gamma_{h,\nearrow}$. Therefore, the jobs arriving in $[\gamma_{h,\nearrow}, f_k)$ are exactly the same as the task system analyzed in Definition 1. ◀

5.1 Analysis of the Service Protocol

Unlike the cases L2N and N2N, the cases N2L and L2L must be analyzed under each protocol separately, since the timing behavior of a task τ_k in $\mathcal{T}_{crit}$ under analysis differs depending on how the actual system execution behavior is specified. For case N2L, the worst case response time of task τ_k can be obtained by means of the following lemma:

▶ **Lemma 8.** *Under the service protocol, the response time $f_k - r_k$ in the case N2L is upper-bounded by the minimum positive value of Δ, for which*

$$\Delta = p_k + C_k^\sharp + \sum_{\tau_i \in hp(\tau_k)} \max\{f_1(\tau_i, \Delta), f_2(\tau_i, \Delta)\} \tag{3}$$

if $\Delta \leq T_k$.

Proof. By definition of case N2L, the execution behavior of the local system changes at time $\gamma_{h,\searrow}$, in which $r_k \leq \gamma_{h,\searrow} < f_k$.

There are two cases to be considered:

- **Case 1**: In the interval $[r_k, \gamma_{h,\searrow})$, the schedule σ does not idle at all.
- **Case 2**: In the interval $[r_k, \gamma_{h,\searrow})$, the schedule σ idles at some time prior to $\gamma_{h,\searrow}$.

We note that the schedule σ is busy from $\gamma_{h,\searrow}$ to f_k, since σ is a work-conserving schedule and there is no suspending behavior between $\gamma_{h,\searrow}$ and f_k under the service protocol.

Proof of Case 1: Let t be the earliest moment such that the schedule σ is busy from t to r_k. We note that such t exists. Under the above construction, the schedule σ is busy from t to f_k and idles right prior to t. If we alter the arrival time of the job τ_k from r_k to t, its response time becomes $f_k - t$, which is no less than $f_k - r_k$.

If the job τ_k^j is not offloaded, its execution time is at most $C_{k,1} + C_{k,s} + C_{k,2}$. If the job τ_k^j is offloaded successfully, its execution time is at most $C_{k,1} + p_k + q_k + C_{k,2}$. If the job τ_k^j is offloaded unsuccessfully, its execution time is at most $C_{k,1} + p_k + C_{k,s} + C_{k,2}$. Under the assumption that $p_k + q_k \leq C_{k,s}$ in Section 2, we know that its execution time is upper-bounded by the maximum of the above three scenarios, which is at most $C_{k,1} + p_k + C_{k,s} + C_{k,2} = p_k + C_k^\sharp$.

Since the local system idles prior to t under normal execution behavior, the interference of the higher-priority tasks can be derived from Lemma 3. Therefore, the worst-case response time of τ_k in this case is the minimum positive value of Δ such that

$$\Delta = p_k + C_k^\sharp + \sum_{\tau_i \in hp(\tau_k)} \max\{f_1(\tau_i, \Delta), f_2(\tau_i, \Delta)\} \tag{4}$$

Proof of Case 2: Let t' be the latest moment such that schedule σ is busy from t' to f_k. We note that such t' exists since the schedule idles at some moment in $[r_k, \gamma_{h,\searrow})$. By definition, $t' - r_k \leq R_k^1 + S_k$.

We now analyze an upper bound of $f_k - t'$. Since the schedule σ idles prior to time t', the job of τ_k must have been offloaded. If the offloading operation is successful, the execution time of task τ_k in the interval $[t', f_k)$ is at most $q_k + C_{k,2}$. If the job τ_k^j is offloaded unsuccessfully, its execution time in the interval $[t', f_k)$ is at most $C_{k,s} + C_{k,2}$. Under our assumption that $p_k + q_k \leq C_{k,s}$ in Section 2, we know that its execution time in the interval $[t', f_k)$ is at most $C_{k,s} + C_{k,2}$.

The interference of the higher-priority jobs in the time interval $[t', f_k)$ is obtained using Lemma 3. Since $t' - r_k \leq R_k^1 + S_k$ and the interference of a higher-priority task τ_i from t' to $t' + \Delta$ is at most $\max\{f_1(\tau_i, \Delta), f_2(\tau_i, \Delta)\}$, the worst-case response time of τ_k in Case 2 is $R_k^1 + S_k + \Delta$, where Δ is the minimum positive value with

$$\Delta = C_{k,s} + C_{k,2} + \sum_{\tau_i \in hp(\tau_k)} \max\{f_1(\tau_i, \Delta), f_2(\tau_i, \Delta)\} \tag{5}$$

Because $C_{k,s} + C_{k,2} \leq C_k^\sharp$, the worst-case response time obtained by Eq. (4) dominates the one from Eq. (5), which concludes this lemma. ◀

Having examined the case N2L under the service protocol, we subsequently consider the case L2L for a task τ_k in $\mathcal{T}_{crit}$ under analysis. The worst-case response time of task τ_k in this case can be determined by the following lemma:

▶ **Lemma 9.** *Under the service protocol, the response time $f_k - r_k$ in the case L2L is upper-bounded by the worst-case response time derived in Lemma 8 if $\Delta \leq T_k$.*

Proof. We note that the schedule σ is busy from r_k to f_k, since σ is a work-conserving schedule and there is no suspending behavior between r_k and f_k under the service protocol. Based on the schedule σ, we examine the following two moments[2]:

- Let t be the earliest moment such that schedule σ is busy from t to r_k.
- Let t' be the latest moment such that local system switches from normal to local execution behavior before or at r_k.

We note that both t and t' exist. There are two scenarios to be analyzed:

- $t \leq t'$: The local system exhibits normal execution behavior prior to time t'. We can change the release time of job τ_k to t' without decreasing its response time. Then, the analysis of Case 1 in Lemma 8 can be applied directly, since the worst-case execution time of τ_k is at most $C_k^\sharp$ in this scenario.
- $t > t'$: The schedule σ idles at $t - \varepsilon$ for an infinitesimal ε and the local system exhibits local execution behavior from t' to t. Therefore, the idle time in schedule σ is due to the removal of the lower-priority tasks from the original schedule. In this case, there exists no unfinished job of any $hp(\tau_k)$ at time t. All jobs released by τ_k and $hp(\tau_k)$ are executed locally. Therefore, the classical critical instant theorem by Liu and Layland [16] can be applied. The worst-case response time in this case is at most the minimum positive value of Δ, for which

$$\Delta = C_k^\sharp + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{\Delta}{T_i} \right\rceil C_i^\sharp \tag{6}$$

  if $\Delta \leq T_k$. This case is dominated by Eq. (4). ◀

Resulting from the above analyses, the schedulability of a task τ_k in $\mathcal{T}_{crit}$ under the service protocol can be verified by the following theorem:

▶ **Theorem 10.** *Consider the service protocol. Suppose that Definition 1 holds, i.e., every task τ_k in $\mathcal{T}$ meets its deadline under normal execution behavior. Every task τ_k in $\mathcal{T}_{crit}$ meets its deadline under local execution behavior if there exists a Δ with $0 < \Delta \leq D_k$ such that the condition in Eq.* (3) *holds.*

Proof. According to Lemma 9, the scenario L2L is dominated by N2L. By Lemma 6, L2N is not possible under both protocols in this paper. By Lemma 7, the worst-case response time due to N2N is at most R_k^{normal}. By Definition 1, $R_k^{normal} \leq D_i$. Therefore, the only condition to check the feasibility is to verify the scenario N2L based on Eq. (3) in Lemma 8. ◀

5.2 Analysis of the Return Protocol

The return protocol is designed to reduce the workload on the local system so that a faster transit from local to normal execution behavior is possible. Under the return protocol, the execution time of a job of τ_i in $\mathcal{T}_{non}$ on the local system is always no more than $C_i^\flat$

[2] We note that the schedule σ is already modified by removing lower-priority jobs. Therefore, it is possible that the reduced schedule idles but the local system exhibits local execution behavior.

independent of the system execution behavior. The analysis of the service protocol in Lemma 8 and Lemma 9 can be slightly changed to accommodate such workload reduction under the return protocol, as stated in the following lemmata:

▶ **Lemma 11.** *Under the return protocol, the response time $f_k - r_k$ in case N2L is upper bounded by the minimum positive value of Δ, for which*

$$\Delta = p_k + C_k^\sharp + \sum_{\tau_i \in hp(\tau_k) \subset \mathcal{T}_{crit}} \max\{f_1(\tau_i, \Delta), f_2(\tau_i, \Delta)\} + \sum_{\tau_i \in hp(\tau_k) \subset \mathcal{T}_{non}} \left(\left\lceil \frac{\Delta}{T_i} \right\rceil + 1\right) C_i^\flat \tag{7}$$

if $\Delta \leq T_k$.

Proof. The proof is almost identical to the proof of Lemma 8 by considering different interferences of the higher-priority task τ_i using Lemma 3 when τ_i is in $\mathcal{T}_{crit}$ or Lemma 4 when τ_i is in $\mathcal{T}_{non}$. We note that the main argument in the proof of Lemma 8 that the local system is busy from $\gamma_{h,\searrow}$ to f_k remains valid, since task τ_k is in $\mathcal{T}_{crit}$ and cannot offload from $\gamma_{h,\searrow}$ to f_k. ◀

▶ **Lemma 12.** *Under the return protocol, the response time $f_k - r_k$ in the case L2L is upper bounded by the worst-case response time derived in Lemma 11 if $\Delta \leq T_k$.*

Proof. The proof is identical as a patch of Lemma 9 by considering different interferences of the higher-priority task τ_i using Lemma 3 when τ_i is in $\mathcal{T}_{crit}$ or Lemma 4 when τ_i is in $\mathcal{T}_{non}$. ◀

Resulting from the above analyses, the schedulability of a task τ_k in $\mathcal{T}_{crit}$ under the return protocol can be verified by the following theorem:

▶ **Theorem 13.** *Consider the return protocol. Suppose that Definition 1 holds, i.e., every task τ_k in $\mathcal{T}$ meets its deadline under normal execution behavior. Every task τ_k in $\mathcal{T}_{crit}$ meets its deadline under local execution behavior if there exists a Δ with $0 < \Delta \leq D_k$ such that the condition in Eq. (7) holds.*

Proof. According to Lemma 12, the case L2L is dominated by N2L. By Lemma 6, L2N is not possible under both protocols in this paper. By Lemma 7, the worst-case response time due to N2N is at most R_k^{normal}. By Definition 1, $R_k^{normal} \leq D_i$. Therefore, the only condition to check the feasibility is to verify the case N2L under the return protocol based on Eq. (7) in Lemma 11. ◀

6 Evaluation

To evaluate our proposed protocols, we perform comprehensive experiments using synthesized data as well as a case study regarding a robot. In the following, we first clarify the setup for both experiments in Sec. 6.1, before we discuss our findings in Sec. 6.2 and Sec. 6.3, respectively.

6.1 Experiment Setup

In our simulations based on synthetic data, we examine the system behavior under each protocol depending on different aspects, namely, 1a) the system utilization under normal execution behavior, 1b) the probability that an offloading operation is performed unsuccessfully, 1c) the percentage of critical tasks in the task set (CT), and 1d) the interval out of which the task periods are generated. More precisely, we measure the amount of time the considered system exhibits local execution behavior in experiments 1a-I) and 1b) - 1d), and the number of synthesized task sets that pass the schedulability tests in Theorem 10 and Theorem 13, respectively, i.e., the so-called acceptance ratio, in experiment 1a-II). For the purpose of analyzing the system execution behavior, we developed an event-based miss rate simulator, which will be released together with the submitted paper.

For each scenario, i.e., 1a) - 1d)[3], we generate sets of 10 non-suspending sporadic tasks, whereat the task utilization values for a given system utilization[4] are generated by means of the UUniFast method [2]. The task periods in experiments 1a) - 1c) are specified according to a log-uniform distribution over the interval $[1ms, 100ms]$ (detailed explanations regarding this approach can be found in [8]) and according to a uniform distribution over the intervals $[1ms, 2ms], [1ms, 10ms], [1ms, 20ms], [1ms, 50ms]$, and $[1ms, 100ms]$ in experiment 1d). The worst-case execution time under normal execution behavior is given as $C_k = T_k \cdot U_k$. Moreover, we set deadlines as implicit, i.e., $D_k = T_k$. Thereon, we transform all non-suspending sporadic tasks into self-suspending tasks consisting of two computation segments as well as one suspension interval, as predefined in the system model in Section 2. For this purpose, we choose the length of each task's suspension interval according to a uniform random distribution out of the interval $S_k \in [0.01 \cdot (T_k - C_k), 0.1 \cdot (T_k - C_k)]$. To generate the corresponding local computation segments $c_{k,s}$, we choose a scaling factor $\alpha = 2$ such that $C_{k,s} = S_k \cdot \alpha$, whereas C_k is divided into $C_{k,1}$ and $C_{k,2}$. Priorities are assigned on the task-level according to the rate-monotonic (RM) approach. We assume the probability that a task τ_k offloads unsuccessfully to follow a Poisson distribution, i.e., $1 - exp(-\lambda \cdot S_k)$, which models that for an offloading operation with longer suspension time the probability of experiencing an unsuccessful offloading operation is higher. We set λ to $0.01 \cdot \frac{1}{ms}, 0.05 \cdot \frac{1}{ms}, 0.1 \cdot \frac{1}{ms}, 0.5 \cdot \frac{1}{ms}$, and $1 \cdot \frac{1}{ms}$ in experiment 1a) and 1b), and to $0.1 \cdot \frac{1}{ms}$ otherwise. The percentage of critical tasks in the system (CT) is set to $10\%, 20\%, 30\%, 40\%, 50\%$, and 60% in experiment 1c) and to 20% otherwise. Each experiment was repeated 100 times, except experiment 1a-I), which was repeated 10 times. For experiments 1a-I) and 1b)-1d), only task sets were considered that passed the schedulability tests in Theorem 10 and Theorem 13.

In experiment 2), we consider a Robotnik RB-1 Base robot platform [19], which is capable of performing loading operations of logistics objects in a highly unstructured environment, using the Robot Operating System (ROS) [20]. We simulated the navigation of the robot in a virtual map in a Gazebo-based environment [11] and measured the timing data of the `move_base` node during a time frame of 60 seconds using the Real-Time Scheduling Framework for ROS (ROSCH) [21] and RESCH [13]. More precisely, the execution times of the topics the `move_base` node, i.e., the main node used for the robot navigation in ROS, subscribed to, were measured, i.e., `/rb1_base/front_laser/scan`, which is the laser scanner data topic, `/rb1_base/robotnik_base_control/odom`, which is the odometry topic containing motor encoder readings and is used for localization, and /tf, which contains the transformations between different ROS 3D coordinate frames.

Resulting from this, three periodic, implicit-deadline tasks have been obtained, as shown in Table 2, which are transformed into self-suspending tasks analogously to the tasks in experiment 1), while considering the cases that $20\%, 40\%$, and 60% of the task workload are

[3] Please note that we also tested other configurations, but since the results were similar, we restrain from discussing them in this section.

[4] Please note that the utilization indicated in all figures always refers to the utilization under normal execution behavior. The system utilization under local execution behavior is in all cases higher and varies depending on the properties of the individual task sets.

offloaded. Moreover, we assume that $\mathcal{T}_{crit} = \{\tau_{odom}\}$ and $\mathcal{T}_{non} = \{\tau_{laser}, \tau_{tf}\}$. We simulate the system behavior using the event-based miss rate simulator from experiments 1) with $\lambda = 0.1 \cdot \frac{1}{ms}$. For each offloading case, the simulation was repeated 100 times.

Table 2 Periodic, implicit-deadline tasks, measurements of a Robotnik RB-1 Base robot platform. Note that the frequency of task τ_{laser} is 15.5 Hz.

Task	Worst-Case Execution Time [ms]	Period [ms]
τ_{laser}	6.732	64.516
τ_{odom}	1.046	60.0
τ_{tf}	0.333	60.0

6.2 Simulation Results

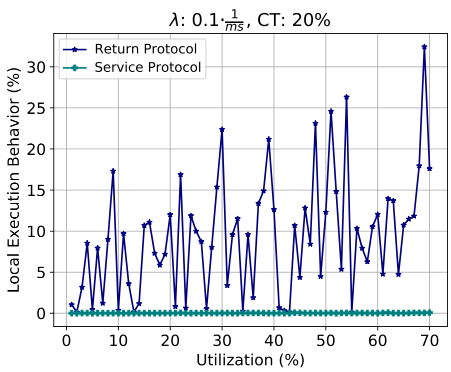


Figure 8 Experiment 1a-I): The percentage of time the system exhibits local execution behavior depending on the system utilization.

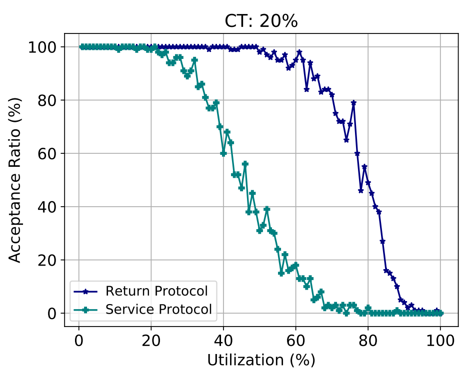
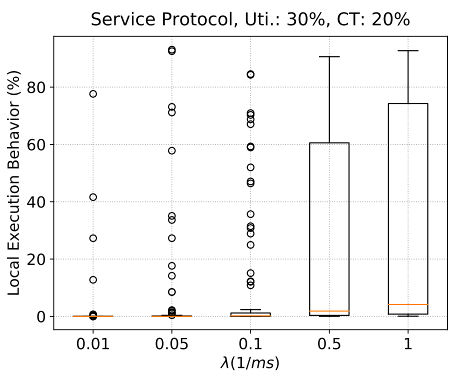
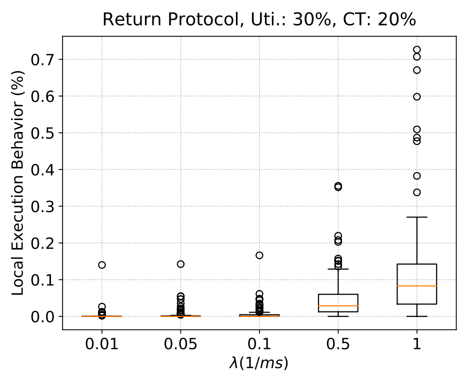


Figure 9 Experiment 1a-II): The acceptance ratios of the schedulability tests of the service and the return protocol.



(a) The percentage of local execution behavior for different probabilities of unsuccessful offloading operations under the service protocol.



(b) The percentage of local execution behavior for different probabilities of unsuccessful offloading operations under the return protocol.

Figure 10 Experiment 1b): The percentage of time the system exhibits local execution behavior during the simulation for different probabilities of unsuccessful offloading operations under the service and the return protocol with a system utilization of 30% and 20% critical tasks.

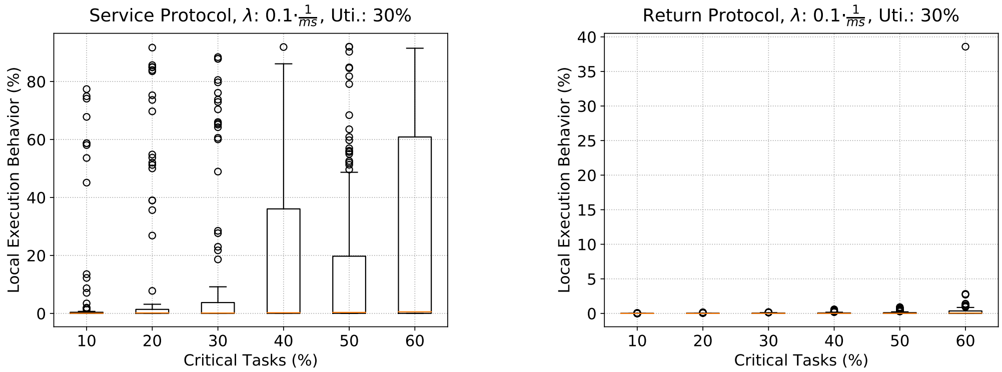
In Fig. 8, the results of experiment 1a-I) are depicted, namely, the mean value over all experiment repetitions of the percentage of simulation time, in which the system exhibits local execution behavior, as a function of the system utilization under normal execution behavior. While the percentage of time the system exhibits local execution behavior under the return protocol is always 0 or close to 0, the values under the service protocol vary largely between 0 and approximately 30%. From these results, we conclude that the percentage of time the system exhibits local execution behavior is not only dependent on the system utilization, but very likely also on other factors, which are discussed hereinafter.

As the outcome of experiment 1a-II), Fig. 9 portrays the acceptance ratios for the schedulability tests in Theorem 10 and Theorem 13, i.e., the percentage of generated task sets passing the schedulability tests for the service and the return protocol, respectively. The service protocol achieves an acceptance ratio of (close to) 100% until approximately 20% system utilization, which approaches 0% at approximately 70% system utilization. The acceptance ratio of the return protocol, in contrast, is (close to) 100% until approximately 40%, before it decreases and finally reaches 0% at approximately 95% system utilization. Please note that similar results were obtained under different configurations.

The following figures, i.e., Fig. 10 - Fig. 13, can be understood as follows: An orange line marks the median, whereas a box comprise three quartiles of the data points, i.e., the lower margin indicates the 25-percent-mark Q_1 and the upper margin indicates the 75-percent-mark Q_3. The distance between the 25- and 75-mark is denoted interquartile range IQ. The lower whisker indicates $Q_1 - 1.5 \cdot IQ$, the upper whisker $Q_3 + 1.5 \cdot IQ$, and circles mark outlier points.

In Fig. 10, the percentage of time the system exhibits local execution behavior under different probabilities of unsuccessful offloading operations, i.e., the outcome of experiment 1b), is presented with respect to the service protocol (cf. Fig. 10a) and the return protocol (cf. Fig. 10b). In general, despite some outliers, the time the system exhibits local execution behavior increases with an increasing probability of unsuccessful offloading operations. If λ is low, i.e., $0.01 \cdot \frac{1}{ms}$ and $0.05 \cdot \frac{1}{ms}$, both protocols lead in the majority of cases to a quite low percentage of time with local execution behavior. If, however, λ is high, i.e., $1 \cdot \frac{1}{ms}$, the service protocol leads to a significantly higher percentage of time under local execution behavior (median approximately 5%, Q_3 approximately 75%, upper whisker approximately 95%) than the return protocol (median approximately 0.08%, Q_3 approximately 0.15%, upper whisker approximately 0.28%). This follows from the different handling of non-critical tasks under the particular protocol if the system exhibits local execution behavior. The outliers can, in general, be explained by the fact that different tasks can suffer from unsuccessful offloading operations, leading to different consequences (consider, e.g., a task with a short period in contrast to a task with a long period).

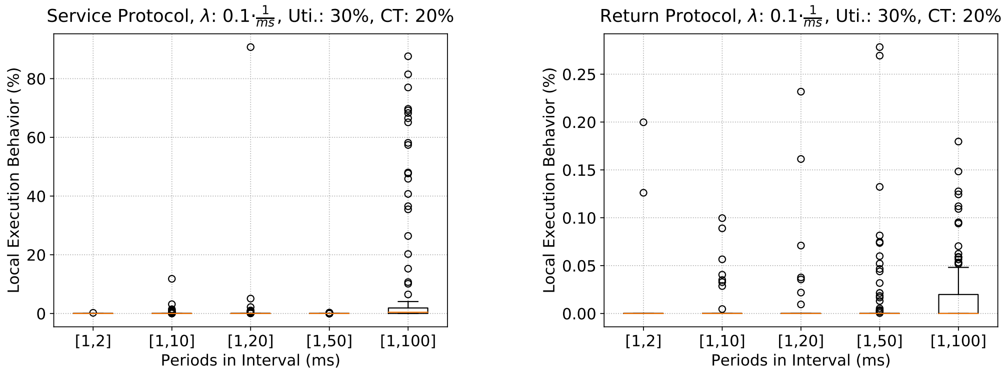
Fig. 11 illustrates the percentage of time the system exhibits local execution behavior under different percentages of critical tasks in the system with respect to the service protocol (cf. Fig. 11a) and the return protocol (cf. Fig. 11b), i.e., the results of experiment 1c). It is evident that the percentage of time the system exhibits local execution behavior increases with an increasing percentage of critical tasks in the system, although the increase is larger under the service protocol than under the return protocol (except one outlier under a percentage of critical tasks of 60% in Fig. 11b). However, comparing the medians in Fig. 11a to those in Fig. 10a, it can be stated that the effect the percentage of critical tasks in the system has on the percentage the system exhibits local execution behavior is less than the impact of the probability of unsuccessful offloading operations.



(a) The percentage of local execution behavior for different percentages of critical tasks in the system under the service protocol.

(b) The percentage of local execution behavior for different percentages of critical tasks in the system under the return protocol.

Figure 11 Experiment 1c): The percentage of time the system exhibits local execution behavior during the simulation for different percentages of critical tasks in the system under the service and the return protocol with a system utilization of 30% and $\lambda = 0.1 \cdot \frac{1}{ms}$.



(a) The percentage of local execution behavior for different period intervals under the service protocol.

(b) The percentage of local execution behavior for different period intervals under the return protocol.

Figure 12 Experiment 1d): The percentage of time the system exhibits local execution behavior during the simulation for different intervals used for the period generation with UUnifast under the service and the return protocol with a system utilization of 30%, 20% critical tasks and $\lambda = 0.1 \cdot \frac{1}{ms}$.

In Fig. 12, the percentage of time the system exhibits local execution behavior under different probabilities of unsuccessful offloading operations, i.e., the outcome of experiment 1d), is depicted with respect to the service protocol (cf. Fig. 12a) and the return protocol (cf. Fig. 12b). Under both protocols, no clear correlation is discernible between the percentage of time the system exhibits local execution behavior and the intervals out of which the task periods are generated except with respect to the interval $[1, 100]$. Although the medians are close to 0% under each protocol in this case, a slight increase of the percentage of time with local execution behavior is visible. As mentioned regarding the results of experiment 1b), this may result from widely differing task periods leading to an increased amount of time under local execution behavior if a task with a long period offloads unsuccessfully.

6.3 Case Study

In Fig. 13, the results of experiment 2), i.e., of our case study considering the task set obtained from a Robotnik RB-1 Base robot (cf. Sec. 6.1), are visualized. From Fig. 13b, Fig. 13d, and Fig. 13f, it is discernible that the amount of offloaded workload per task has no significant impact on the percentage of time the system exhibits local execution behavior. Under the service protocol, in contrast, a clear increase of the time the system exhibits local execution behavior for higher probabilities of unsuccessful offloading operations, i.e. for $\lambda \in \left\{0.5 \cdot \frac{1}{ms}, 1 \cdot \frac{1}{ms}\right\}$, is visible with an increasing amount of offloaded workload per task (cf. Fig. 13a, Fig. 13c, and Fig. 13e). In consequence, it can be concluded that the amount of offloaded workload per task has strong impact on the system execution behavior under the service protocol and thus should be taken into consideration at system design time.

7 Conclusion

In this work, we proposed two protocols for cyber-physical systems by means of which critical and non-critical tasks can be offloaded safely, namely, the service protocol and the return protocol (cf. Sec. 3). We analyzed the worst-case timing behavior of the local cyber-physical system and, based on these analyses, we provided a sufficient schedulability test for each of the proposed protocols (cf. Sec. 4 and Sec. 5). In the course of comprehensive experiments and a case study involving a Robotnik RB-1 Base robot (cf. Sec. 6), we showed that our protocols have reasonable acceptance ratios under the provided schedulability tests. Moreover, we demonstrated that the system behavior under our proposed protocols depends on different factors, namely, on the probability of unsuccessful offloading operations, the percentage of critical tasks in the system, and the amount of offloaded workload. Not least, evidence was found that the percentage of time the system exhibits local execution behavior also depends on the actual task experiencing an unsuccessful offloading operation and, in consequence, also on the interval out of which task periods are chosen.

References

1 A. Adiththan, S. Ramesh, and S. Samii. Cloud-assisted control of ground vehicles using adaptive computation offloading techniques. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 589–592, 2018. doi:10.23919/DATE.2018.8342076.

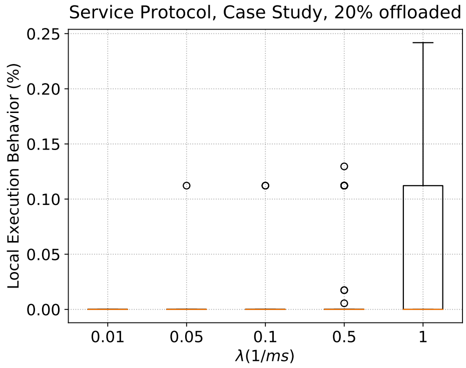
2 Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

3 Georg von der Brüggen, Kuan-Hsun Chen, Wen-Hung Huang, and Jian-Jia Chen. Systems with dynamic real-time guarantees in uncertain and faulty execution environments. In *Real-Time Systems Symposium (RTSS)*, Porto, Portugal, 2016. URL: https://ieeexplore.ieee.org/document/7809865.

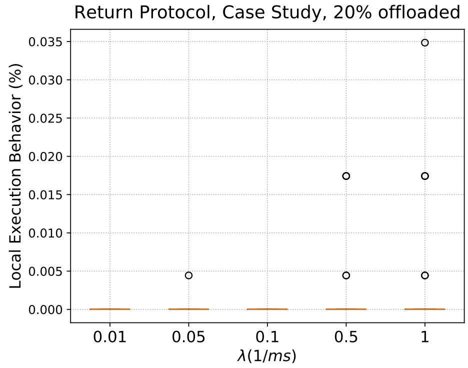
4 Georg von der Brüggen, Lea Schönberger, and Jian-Jia Chen. Do nothing, but carefully: Fault tolerance with timing guarantees for multiprocessor systems devoid of online adaptation. In *The 23rd IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2018)*, Taipei, Taiwan, 2018. URL: https://ieeexplore.ieee.org/document/8639554.

5 Alan Burns and Robert I. Davis. A survey of research into mixed criticality systems. *ACM Comput. Surv.*, 50(6):82:1–82:37, 2018. doi:10.1145/3131347.

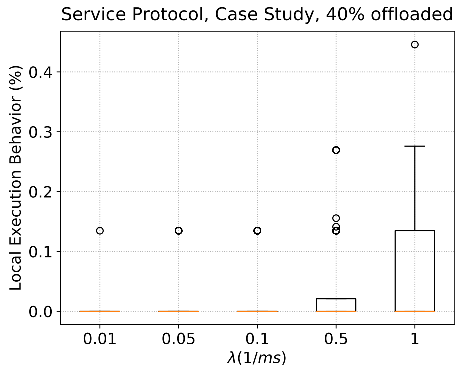
6 Jian-Jia Chen, Geoffrey Nelissen, Wen-Hung Huang, Maolin Yang, Björn Brandenburg, Konstantinos Bletsas, Cong Liu, Pascal Richard, Frédéric Ridouard, Neil Audsley, Raj Rajkumar, Dionisio de Niz, and Georg von der Brüggen. Many suspensions, many problems: A review of self-suspending tasks in real-time systems. *Real-Time Systems*, 2018. doi:10.1007/s11241-018-9316-9.



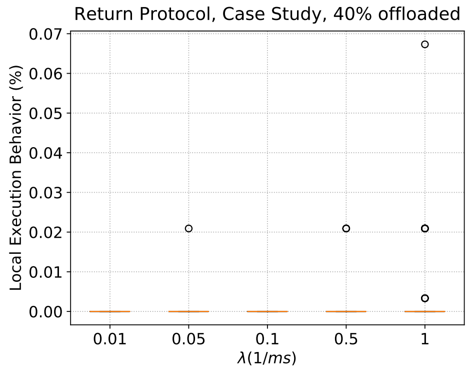
(a) The percentage of local execution behavior for different probabilities of unsuccessful offloading operations under the service protocol and 20% offloaded workload per task.



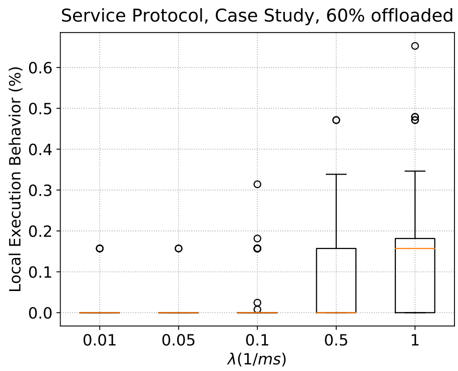
(b) The percentage of local execution behavior for different probabilities of unsuccessful offloading operations under the return protocol and 20% offloaded workload per task.



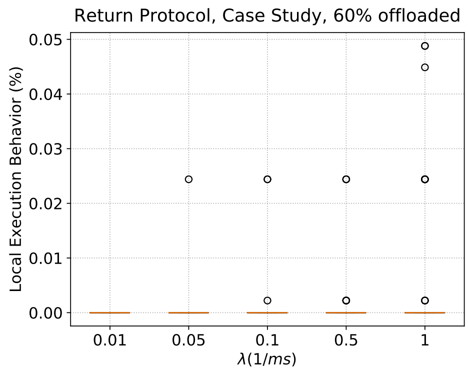
(c) The percentage of local execution behavior for different probabilities of unsuccessful offloading operations under the service protocol and 40% offloaded workload per task.



(d) The percentage of local execution behavior for different probabilities of unsuccessful offloading operations under the return protocol and 40% offloaded workload per task.



(e) The percentage of local execution behavior for different probabilities of unsuccessful offloading operations under the service protocol and 60% offloaded workload per task.



(f) The percentage of local execution behavior for different probabilities of unsuccessful offloading operations under the return protocol and 60% offloaded workload per task.

Figure 13 Experiment 2): The percentage of time the robot exhibits local execution behavior during the simulation for different probabilities of unsuccessful offloading operations and different percentages of offloaded workload under the service and the return protocol.

7 Jian-Jia Chen, Georg von der Brüggen, Wen-Hung Huang, and Cong Liu. State of the art for scheduling and analyzing self-suspending sporadic real-time tasks. In *23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA*, pages 1–10, 2017. `doi:10.1109/RTCSA.2017.8046321`.

8 Robert I. Davis, Attila Zabos, and Alan Burns. Efficient exact schedulability tests for fixed priority real-time systems. *Computers, IEEE Transactions on*, 57(9):1261–1276, 2008.

9 Hasan Esen, Masakazu Adachi, Daniele Bernardini, Alberto Bemporad, Dominik Rost, and Jens Knodel. Control as a service (caas): Cloud-based software architecture for automotive control applications. In *Proceedings of the Second International Workshop on the Swarm at the Edge of the Cloud*, SWEC '15, page 13–18, New York, NY, USA, 2015. Association for Computing Machinery. `doi:10.1145/2756755.2756758`.

10 Alexandre Esper, Geoffrey Nelissen, Vincent Nélis, and Eduardo Tovar. How realistic is the mixed-criticality real-time system model? In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, RTNS '15, page 139–148, New York, NY, USA, 2015. Association for Computing Machinery. `doi:10.1145/2834848.2834869`.

11 Gazebo. Gazebo. robot simulation made easy. `http://gazebosim.org/`.

12 W.-H. Huang, J. Chen, and C. Liu. Pass: Priority assignment of real-time tasks with dynamic suspending behavior under fixed-priority scheduling. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2015. `doi:10.1145/2744769.2744891`.

13 Shinpei Kato, R Rajkumar, and Yutaka Ishikawa. A loadable real-time scheduler suite for multicore platforms. *Technical Report CMU-ECE-TR09-12*, 2009.

14 Swarun Kumar, Shyamnath Gollakota, and Dina Katabi. A cloud-assisted design for autonomous driving. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, page 41–46, New York, NY, USA, 2012. Association for Computing Machinery. `doi:10.1145/2342509.2342519`.

15 C. Liu and J. Chen. Bursty-interference analysis techniques for analyzing complex real-time task models. In *Real-Time Systems Symposium (RTSS)*, pages 173–183, 2014.

16 C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973. `doi:10.1145/321738.321743`.

17 M. A. Maruf and A. Azim. Extending resources for avoiding overloads of mixed-criticality tasks in cyber-physical systems. *IET Cyber-Physical Systems: Theory Applications*, 5(1):60–70, 2020.

18 Institute of Electrical and Electronic Engineers. 802.11p-2010 - IEEE standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 6: Wireless access in vehicular environments. `https://standards.ieee.org/standard/802_11p-2010.html`, 2017.

19 Robotnik. Mobile robot RB-1 base. `https://www.robotnik.eu/mobile-robots/rb-1-base-2/`.

20 ROS. Robot operating system (ROS). `https://www.ros.org/`.

21 Y. Saito, F. Sato, T. Azumi, S. Kato, and N. Nishio. Rosch:real-time scheduling framework for ROS. In *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 52–58, 2018. `doi:10.1109/RTCSA.2018.00015`.

22 Lea Schönberger, Wen-Hung Huang, Georg von der Brüggen, Kuan-Hsun Chen, and Jian-Jia Chen. Schedulability analysis and priority assignment for segmented self-suspending tasks. In *The 24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Hakodate, Japan, 2018.

23 Lea Schönberger, Wen-Hung Huang, Georg von der Brüggen, and Jian-Jia Chen. Schedulability analysis and priority assignment for segmented self-suspending tasks. Technical report, TU Dortmund, 2018.

24 Benjamin Sliwa, Robert Falkenberg, Thomas Liebig, Nico Piatkowski, and Christian Wietfeld. Boosting vehicle-to-cloud communication by machine learning-enabled context prediction. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

25 Benjamin Sliwa and Christian Wietfeld. Empirical analysis of client-based network quality prediction in vehicular multi-MNO networks. In *2019 IEEE 90th Vehicular Technology Conference (VTC-Fall)*, Honolulu, Hawaii, USA, 2019.

26 S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 239–243, 2007. `doi:10.1109/RTSS.2007.47`.

27 Georg von der Brüggen, Wen-Hung Huang, and Jian-Jia Chen. Hybrid self-suspension models in real-time embedded systems. In *23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA*, pages 1–9, 2017. `doi:10.1109/RTCSA.2017.8046328`.

28 Georg von der Brüggen, Wen-Hung Huang, Jian-Jia Chen, and Cong Liu. Uniprocessor scheduling strategies for self-suspending task systems. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS)*, pages 119–128, 2016. URL: `http://dl.acm.org/ft_gateway.cfm?id=2997497&ftid=1804918&dwn=1&CFID=691780547&CFTOKEN=64912419`.