# Environment-driven Communication in Battery-free Smart Buildings

MAURO PIVA, ANDREA COLETTA, and GAIA MASELLI, Sapienza University of Rome, Italy
JOHN A. STANKOVIC, University of Virginia, USA

Recent years have witnessed the design and development of several smart devices that are wireless and battery-less. These devices exploit RFID backscattering-based computation and transmissions. Although singular devices can operate efficiently, their coexistence needs to be controlled, as they have widely varying communication requirements, depending on their interaction with the environment. The design of efficient communication protocols able to dynamically adapt to current device operation is quite a new problem that the existing work cannot solve well. In this article, we propose a new communication protocol, called ReLEDF, that dynamically discovers devices in smart buildings and their active and nonactive status and when active their current communication behavior (through a learning-based mechanism) and schedules transmission slots (through an Earliest Deadline First– (EDF) based mechanism) adapt to different data transmission requirements. Combining learning and scheduling introduces a tag starvation problem, so we also propose a new mode-change scheduling approach. Extensive simulations clearly show the benefits of using ReLEDF, which successfully delivers over 95% of new data samples in a typical smart home scenario with up to 150 heterogeneous smart devices, outperforming related solutions. Real experiments are also conducted to demonstrate the applicability of ReLEDF and to validate the simulations.

CCS Concepts: • **Networks → Link-layer protocols**; **Cyber-physical networks**;

Additional Key Words and Phrases: RFID backscattering, battery-free devices, reinforcement learning, EDF

**ACM Reference format:**
Mauro Piva, Andrea Coletta, Gaia Maselli, and John A. Stankovic. 2021. Environment-driven Communication in Battery-free Smart Buildings. *ACM Trans. Internet Things* 2, 2, Article 14 (April 2021), 30 pages.
https://doi.org/10.1145/3448739

## 1 INTRODUCTION

The recent dramatic rise in production of wireless Internet of Things (IoT) devices, coupled with the difficulty in disposing exhausted batteries, raises a crucial question: Is it possible to develop IoT devices that do not require batteries for power? Researchers in the fields of networks and communication have answered this question with backscattering [12]. Recent years have witnessed the design and development of several *wireless* and *battery-less* smart devices that exploit **Radio**

This work was carried out within the research project "SMARTOUR: intelligent platform for tourism" funded by the Ministry of University and Research with the Regional Development Fund of European Union (PON Research and Competitiveness 2007-2013).
Authors' address: M. Piva, A. Coletta, and G. Maselli, Sapienza University of Rome, CS Dept. Viale Regina Elena 295, 00161 Rome, Italy; emails: {mauro.piva, coletta, maselli}@di.uniroma1.it; J. A. Stankovic, CS Dept. University of Virginia, P.O. Box 400740 Charlottesville, Virginia 22904, USA; email: stankovic@cs.virginia.edu.

2577-6207/2021/04-ART14 $15.00
https://doi.org/10.1145/3448739

**Frequency Identification (RFID)** backscattering to operate [18]. In RFID technology, a backscatter device harvests RF energy, computes or senses data, and transmits data by reflecting the **Radio Frequency (RF)** signal. In practice, any device that operates at a relatively low power budget can be remotely powered through the backscattering of RF signals; this means it can harvest power from signals emitted by a dedicated powered device, namely a RFID reader.

Currently, several battery-less IoT devices have been proposed (a battery-less light switch [18], a battery-less joystick [19], a battery-free RFID camera [23], and a battery-free cell-phone [36]) and developed (an RF Field detector, a electronic relay, and a magnetic field sensor [9]). However, most works only present singular devices. Although stand-alone solutions can operate efficiently, their performance degrade significantly when they share the transmission channel with other devices. Experiments with three coexisting devices have shown that data delivery is delayed of a factor 2 with respect to the stand-alone counterpart [18]. These results highlight that coexistence of multiple devices need to be controlled, as they have widely varying communication requirements (transmission rates, ON/OFF activity, and deadlines).

Let us imagine a smart building, outfitted with a myriad of battery-less sensors and smart devices (e.g., cameras, presence sensors, smoke sensors, light sensors, thermostats, smart meters, those with real-time user interactions, etc.) that are used to reduce resource consumption and improve the quality of life. These smart systems will contain many devices with widely varying communications requirements depending on sensed events, transmission rates, number of bits per sensing sample, ON/OFF activity, and deadlines. Many of these devices will also have dynamically varying communications requirements based on their current operational mode. In fact, the nature of the sensor data generation is spontaneous; it is highly dependent on the events in the environment. Many applications are interested in changes in measured values: They want to obtain new data when the sensed value has changed since the previous sample rather than at the devices' current sensing rate. As an example, a temperature sensor has the capability of sensing the environment every 25 ms, but a significant change in the temperature (for example of at least 1°C), since the last reading may happen after 1 s in case of a fire or after 1 hour in case of normal conditions. To choose another example, a joystick may sense no changes for hours (while it is OFF), and then start sensing new data (while used for playing) at very different rates (from a few milliseconds to one or more seconds), depending on the game type and the player's activity. Thus, the rate at which sensors detect a new value depends on environmental events. In other words, it is the environment that changes and while the sensors sampling rates are set according to the Nyquist sampling theorem, to minimize overhead, wireless traffic, and handle large numbers of devices, it is often necessary to act only when there is a significant change in the sensed value. Consequently, in a smart building setting, *a device should transmit only when it has new/changed data to send*. This is a challenging task when there are many heterogeneous devices that have data to transmit at the same time. **Medium access control (MAC)** protocols for RFID-based devices have to follow a time slotted scheme (i.e., **time division mechanism (TDMA)**), because they need to be energized by the reader. However, current TDMA protocols based on random access (i.e., the tag randomly selects a slot to transmit) perform poorly: collision time is above 50% [15]. A better solution to avoid collisions is to directly poll each device that has a new data sample. But the reader would have to know when a device has generated new data to query it. Hence, there is a need for the reader to *guess* which devices should be queried.

For many devices, especially those interacting with humans, it is also necessary to deliver data by a *deadline*. IoT environments will also evolve with devices being added and removed from the environment, with a need for a zero configuration capability. Consequently, it is a significantly difficult challenge to design a communication protocol that dynamically discovers (current) environmental patterns, changes in those patterns, and required rate for reading/transmission, for devices

that are added and taken away over time. In other words there is a need for a communication protocol that is able to detect, learn and monitor devices' presence and transmission needs (depending on events in the environment) and dynamically adapt to current device states and requirements, including deadline requirements for packet delivery. Meeting deadline requirements are especially challenging when device requirements dynamically change and new devices activate, which requires schedules for packet delivery to be adjusted. Maintaining performance requirements in the transition period between the current and new schedule is also very difficult. Further, in these highly dynamic environments it will not always be possible to meet all performance requirements so it is necessary to establish solutions with *fairness*.

Given the resource and computation constraints typical of backscattering-based devices, as well as the large heterogeneity among devices in a smart home or building, realizing efficient and effective communication is a big challenge. While there is an incredibly large number of MAC protocols, to the best of our knowledge the literature presents only a protocol, named APT-MAC [20], that attempts to learn what devices are in the environment, and what their current communication requirements are. The idea behind APT-MAC is to adopt a bandit algorithm to learn devices behaviour and adapts information collection to such requirements: the system queries the device that has more likely sensed new data. However, the bandit algorithm provides a limited view on devices' behavior: It adapts to changes in the environment without considering the previous history of changes. This may cause slow protocol adaptation. For quicker reaction to changes in the environment, we propose a novel MAC protocol for battery free networks, called ReLEDF, which exploits a Q-learning-based approach coupled with an EDF-based device interrogation mechanism. The use of Q-learning is motivated by its ability to quickly adapt to changes (by considering previous history of changes), while EDF guarantees time requirements of scheduling packet transmissions. The mix of learning and scheduling is an innovative way to schedule the network, which guarantees high performance, but also adds a novel challenge: how to ensure that all devices meet deadlines without starvation. We address this issue by proposing a new mode-change scheduling approach, called **Earliest Deadline First with Inherited Jobs (EDF-IJ)**, which reduces delays and avoids device starvation in scheduling transmissions. In more detail, the main contributions of this article are as follows:

- a *new dynamic and adaptive MAC protocol*, called ReLEDF, able to optimize transmission slots assignment based on device needs, exploiting the combination of **Reinforcement Learning (RL)** techniques and EDF real-time system policies.
- A *new computational approach to learning* current active devices and their dynamic requirements from interaction with the environment, inspired by RL. Our system introduces the concept of sub-agents (multiple entities that are influenced by actions defined by the same agent) and allows building/maintaining knowledge to make predictions on the state of multiple devices inside a network, enabling for a highly dynamic transmission scheduling based on devices' behavior.
- A *no transition starvation* solution for EDF algorithm that is able to avoid starvation of low priority communicating devices while admitting new devices and changing schedules.
- A *new mode-change* scheduling approach based on EDF, called , that is able to reduce delays and avoid task starvation by immediately scheduling a new frame while keeping some information about previously scheduled jobs.
- A new theorem to prove *feasibility analysis* for EDF-IJ, which formally demonstrates that the feasibility of this new approach can be verified in constant time.
- A *fair loss policy* for EDF scheduling that is able to fairly penalize tasks and attain schedule feasibility when there is overload.

- A thorough *comparative performance evaluation* of ReLEDF with APT-MAC [20], TDMA, and optimal protocols. By means of extensive simulations we show that ReLEDF scales well as it is able to successfully always deliver over 95% of the packets with negligible delay, and outperforming state of the art solutions.
- An *experimentation on real prototype devices* to validate simulation results and show the applicability of our new protocol through real experiments.

## 2 RELATED WORK

As this article contains a twofold contribution, i.e., on communication protocols for battery-free devices and a new mode-change scheduling approach for EDF, we present the related work for both of these fields.

### 2.1 Communication Protocols for Battery-free Devices

In this section, we discuss existing solutions to collect data from sensor-augmented RFID tags and their limitations.

Current solutions consider mainly homogeneous environments, i.e., devices that have the same transmission needs. The works in References [8, 26] exploit hash functions to send a unique query to all devices. Many of the devices in the environment will answer sequentially, without the need of a query for each of them. Different from our system, these protocols can only reduce the number of empty slots and optimize transmissions, but they are not able to adapt to current tags needs (e.g., give priority to more demanding devices). Furthermore, as the reader can send only one query that contains information regarding many successive slots, these protocols are not able to dynamically adapt to burst data. The work in Reference [17] studies the problem of range query for sensor-augmented RFID systems, which is to classify the target tags according to the range of tag information. The goal of this work is to optimize the number of queried ranges, maintaining high query accuracy, that is quite different from our work, which learns and adapts to device needs. The work in Reference [25] uses EDF to optimize packet transmission from an energy point of view, but it does not consider the problem of adapting to different device requirements.

Closest to our work is the APT-MAC protocol [20] that dynamically collects data from sensor augmented RFID tags. Specifically, APT-MAC exploits the bandit algorithm to select the next device to query: If the device has a new sample, then the reward is positive; if instead the device does not have new data since the last reading, then the reward is negative. The main limitation of this bandit solution is the lack of previous history of changes, because it keeps only one state (i.e., the current one), and the reward taken under consideration is only the immediate one. Hence, bandit can be thought of as having a single-state episode, meaning that the algorithm is not able to remember past devices behaviours. Every time a device changes behavior, the bandit starts learning from the beginning.

We finally mention a class of protocols, whose goal is tag identification through a sequential [30] or a concurrent [14, 38] approach, which can be adapted to gather data from sensor augmented RFID tags. Specifically, sequential mechanisms can be adapted by storing the sensed value in the buffer used to store the tag ID (up to 256 bits in total [2]), and transmitting them together during the identification process. As these mechanisms rely on a slotted random access, they involve idle and collision slots that significantly delay the data gathering process. Previous studies have shown that the time wasted in idle and colliding slots is above 50%. Concurrent protocols instead are very efficient and can be used to periodically report their measurements to the reader [38] but require the use of signal processing techniques that involve significant computation time, resulting in an inability to deliver data by a deadline.

### 2.2 EDF Mode Change

We now present a number of useful related works for each of the scheduling topics we address in the article (a new scheduling policy, a new feasibility test, and a policy on how to manage overload scheduling).

Deadline scheduling has been deeply studied in the literature, for example Reference [32] offers a prospect of available algorithms in the context of real time systems. Even the issue of dealing with dynamic tasks received huge attention from the scientific community. The problem has been divided into two main areas: the first addresses dynamic workloads, in which some of the tasks to be executed dynamically change, while the second addresses systems with multiple working modes: in this case a change in the environment drives the system from an operating mode to another, asking a number of tasks to be deleted or released. In the case of EDF, a recent solution for transients (i.e., periods of time in which tasks leave and join the schedule), has been presented in Reference [7], where an online protocol is able to manage the admission control of a dynamic workload. In Reference [6], a scheduling frame with elastic coefficients related to each task has been proposed. In this case, the system is able to modify the period of tasks to adapt them to the transient, but no guarantees on precedent tasks deadlines are given, thus causing packets to be lost. Another solution, EDF-VD [3] (extended in Reference [21]), is able to manage task arrivals assigning shorter and "virtual" deadlines to critical tasks. In Reference [4], a partitioned scheduling scheme able to re-weight tasks has been proposed, but the article also points out how the proposed scheme is not capable of providing fairness or real-time guarantees. The main limitations of these works are in terms of starvation risks (less demanding tasks during mode changes could be excluded from the schedule) and in terms of fairness (most demanding tasks, in absence of priority, may completely saturate the communication channel at the expense of less demanding tasks).

In the context of how to deal with mode change the literature presents several solutions [11, 27, 29]. In particular, the work in Reference [29] proposes a protocol for mode change in a preemptive scheduling environment. In Reference [33] the authors address dynamic mode change in real-time systems under both EDF and Fixed Priority scheduling policies, but in this work during a mode change no other task changes are accepted. In a context more related to networking, Reference [34] investigates the adaptive reservation of resource provisioning for servers using a TDMA approach. Another proposal presents a protocol for multicore systems scheduling under the assumption that, in a mode change, only some of the tasks change requirements [24]. While some of these proposals are able to guarantee absolute deadlines handling mode change in real-time systems, they are inappropriate for a networking context as they do not consider the same execution priorities, and they do not allow continuous and independent changes in the tasks set. Even those proposals more related to networking do not match our problem: The tasks cannot independently change requirements and there are admission delays or admission control systems. Differently, we attempt to create a new scheduling policy that is compatible with every already presented TDMA-based MAC protocol, and instead of refusing new tasks we aim at *fairly* updating the task set to make it feasible.

Regarding the feasibility test, it is known that when a new task enters a system, to avoid missing deadlines, the task has to satisfy an admission test, which typically consists in a feasibility test. Generally, in a transient situation in which tasks are leaving and entering the system, a feasibility analysis that does not consider leaving tasks is not safe [6, 37]. Furthermore, in our case we also want to consider some jobs of the old scheduled frame while producing the new one. In Reference [10] two feasibility tests for EDF with mode changes have been proposed: The first one considers a fixed change sequence known *a priori*, while the second one allows the system to change on a limited a set of modes. Unfortunately, both these tests are not applicable to EDF-IJ, as they do not

consider jobs inherited from a number of previous schedules. Another approach to the transient problem consists in transforming temporally some periodic tasks to aperiodic. While an admission controller for tasks set composed of both periodic and aperiodic jobs has been proposed in Reference [1], this transformation has an impact on the whole scheduled frame, while in our proposal we produce changes only on the initial part of the scheduled frame.

For the management of over scheduling, an interesting approach is presented in Reference [6], where a resource manager is able to adapt tasks deadlines, depending on an elastic parameter, to make the task set schedule feasible. This approach, different from our work, does not take into account fairness between tasks while changing their periods.

## 3 OVERVIEW AND ASSUMPTIONS

Our goal is to design a protocol that dynamically discovers when devices have new data to send (different from the previous sampling value) and adapts to their different communication requirements. This protocol is practical to use in IoT environments, i.e., devices may dynamically change their transmission modes by detecting different type of events in the environment. In particular, we consider the case of battery-free devices [18], realized through sensor-augmented RFID tags, such as the Moo [40] or the Intel DL WISP [39] tags. These devices are computational RFIDs. Specifically, we use Moo tags, that are built on the prototype of WISP tags, and feature reprogrammable microcontrollers, sensors and actuators, nonvolatile memory, and small energy buffers that temporarily store harvested energy for computation [40]. Moo tags also allow for functions extension, thanks to their general-purpose I/Os, serial buses, and 12-bit analog-to-digital and digital-to-analog converter ports. The key feature of Moo devices is the energy provisioning: They operate through RFID backscattering, harvesting power from signals emitted by a dedicated RFID reader, to sense and communicate data. In particular, the reader emits a *continuous wave* that is absorbed by devices to get powered and sense. The devices eventually communicate by reflecting or not the continuous wave, modulating messages for the reader. For a survey on capabilities and applications of computational RFIDs, as well as more details on physical layer aspects we point the reader to Reference [18].

Importantly, these devices present specific peculiarities that significantly limit their communication ability. Backscatter communication introduces different MAC layer communication mechanisms. In particular, sensor-augmented RFID tags can communicate only with a dedicated device, i.e., the reader, and cannot practically communicate with each other (the network is single hop). In addition, they cannot perform carrier sense, nor transmit spontaneously, because they have no power source on-board. Hence, communication has to be controlled by the reader that has to query (and hence energize) devices to collect their sensed data. For this reason, we adopt a centralized approach to dynamically gather data from sensor-augmented RFID tags, as tags backscatter the signal received by the reader. In particular, in sensor-augmented RFID systems it is natural to designate the reader as the master of the communication protocol, a centralized entity that receives devices packets, and schedule device transmissions. This master also periodically runs an inventory query to handle devices entering or leaving the area.

Considering the typical characteristics of RFID technology, we make the following assumptions:

- The network has a star topology, with the reader that directly queries all devices. This approach is reasonable in a smart home, because the reader can be plugged in and the rest of the home devices (potentially many) have no batteries. It has been studied that the improvement achieved on the environment by making devices battery-free is greater than the cost of powering a continuous operating reader [20]. In case of large areas with multiple or vast rooms we assume that the RFID reader is equipped with multiple antennas, one

transmitting and one receiving for each room. Each antenna has a transmission range that is able to reach all devices in the same room. When the reader issues a query, all the transmitting antennas broadcast the query at the same time, reaching all the devices in the home. When tags receive a query, only the queried tag (whose ID is indicated inside the query message) backscatters the received signal to send its sensed data to the reader; all the other tags use the received signal to power on-board sensors and store the sensed data in a local buffer. At each query only one device in the home is interrogated. Multiple receiving antennas may receive its response (e.g., antennas in nearby rooms), eventually producing redundant information.

- To minimize configuration costs and handle the IoT world we assume that devices freely enter and leave the system without communicating any information about their type (e.g., temperature sensor, TV remote, etc.); they just receive an ID from the system and start operating. The reasons for assuming no prior knowledge of devices are twofold. First, to allow devices to declare their requirements without specific interaction with the user, the system should have access to a global up to date database, containing information on all possible devices, which would be laborious and error prone. Second, the rate at which sensors need to send data is not necessarily the rate at which they sense the environment. Sensor transmission rate requirements are given by the rate at which they detect a new value (different from the previous sampling), which depends on environmental events. Consequently, a device should transmit only when it has new/changed data to send. Thus, knowing the device sensing rate does not imply knowing the device communication needs that may be unpredictable. For example a joystick may have different transmission rates depending on its current use (e.g., on/off moments as well as different video-game requirements and player activity). For these reasons, we believe that it is simpler and easier to have an automatic and configuration/management free protocol, such that proposed in this article, that can adapt to any newly installed or invented device and any environment change/condition.

To achieve environment-driven communication in such a context, the primary issue that we need to deal with is learning and continuously monitoring environmental patterns and changes in those patterns, which require specific rates for reading/transmission. As sensor transmission rate requirements depend on new/changed sensed data, from now on we define *transmission rate as the rate at which a device needs to send data regarding a new event in the environment* (again, to be clear this is not the sensor sampling rate). The second important issue is to schedule channel access such that data is delivered in time.

We tackle these problems through an innovative MAC protocol that integrates two key components: a RL-based mechanism and an EDF real time scheduling policy. We also address the challenges of mixing learning and scheduling, which guarantees high performance, but also poses a critical issue on the way to *ensure that all devices meet deadlines without starvation.*

We propose a new MAC protocol, named ReLEDF, which quickly learns transmission rate requirements of active devices, without having any *a priori* knowledge on the type of devices, and properly schedules devices transmissions. The protocol relies on two key components: (1) a RL-based mechanism that builds and updates the devices behavioral models and (2) a channel access method based on EDF scheduling, improved to deal with schedule transitions, that maps devices behavior models into a network frame. Figure 1 presents the overall system. In particular, the figure shows a smart home with heating, fire alarm, light switches, cameras, joysticks, and a TV remote as possibly found in a smart home. As possibly multiple users activate any of these devices by using them, the RFID reader detects this and supports the appropriate scheduling and communications. Later, a home owner may add a smart humidity control for plants, and a home

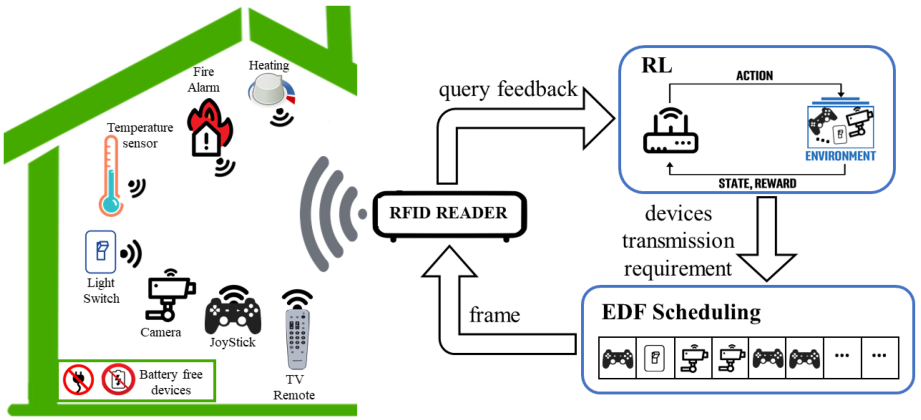


Fig. 1. While receiving data samples, the reader learns and builds behavioral models. In particular, the RL component estimates the current transmission requirement for each device. Then the reader maps transmission requirements into a communication schedule (frame) through an EDF policy.

security system. Such new devices are automatically handled together with the current devices if and when they are used. The two components (i.e., RL and EDF) continuously interact and change upon transmission feedback. In the following we detail the two components.

## 4 RELEDF: LEARNING DEVICES BEHAVIOR

The first component of our system is the construction of a dynamic behavioral model for each network device to learn required transmission rates (at the current time and in the near future). We briefly describe our idea and then present more details.

We want to model device behavior (i.e., the rate at which devices detect new events in the environment) with a graph, where nodes correspond to transmission rates and edges represent changes in transmission rate. As the system does not have prior knowledge on device's type, we consider several nodes in the graph, corresponding to a reasonable set of possible rate requirements $tr_1, tr_2, \ldots, tr_n$, where $tr_1$ represents the highest rate (for example transmission every 20 ms) and $tr_n$ the lowest (for example, once every 2 s).

The lowest transmission rate represents the maximum delay between the generation by a sensor of a new data sample and the transmission of the corresponding packet to the reader. This means that also a dormant sensor, e.g., a fire alarm that is rarely enabled, in our set up is queried at least every 2 s. Notice that, we consider a discrete finite set of transmission requirements to employ Q-learning algorithms. However, device behavior involves only a subset of the nodes: Devices typically move through a small subset of transmission rates. As an example, a temperature sensor mainly remains in a low state in (e.g., $tr_{n-2}$, $tr_{n-1}$) in case of normal conditions, while a joystick may pass from a stand-by state when it is OFF (e.g., $tr_n$ state) to a burst transmission state, when used for playing (e.g., $tr_1$ or $tr_2$ states, depending on user activity and game characteristics).

How do we represent the subset of states modeling a device behavior? We add weights on edges: The value associated with the edge $tr_i \rightarrow tr_j$ represents the probability to change rate from $tr_i$ to $tr_j$. Our goal is to have for each active device an instance of the graph that at each time truly represents current device communication behavior. The technique that we use to discover which state of the graph more likely reflects current device behavior is based on reinforcement learning.

*A brief primer on Reinforcement Learning*: In RL problems, an *agent* interacts with the environment and at each time $t$, is placed in a certain *state* $s$ in the set of states S, and takes an *action* $a$

in A. The agent decides the action to perform depending on a *policy* $\pi(a_t|s_t)$. The policy, for each action $a_t$ and each state $s_t$, defines a scalar *reward* $r_t$ and the transition to the next state $s_{t+1}$, with the help of the state transition probability $P(s_{t+1}|s_t, a_t)$ and the reward function $R(s, a)$. While running, the agent tries to reach a terminal state (with the highest reward). However, if the problem does not contemplate a terminal state, but keeps changing over time, as it is the case addressed by our article, then the problem is called non-stationary.

In the context of our work, considering a single device, the *agent* corresponds to the *interrogator* or *master*—the device dedicated to deciding which device to poll for transmission—*states* represent device's transmission *rate requirements*, *actions* concern *changes of transmission rate*, and the *reward* concerns *finding the current rate requirement* of the device (corresponding to the real rate requirement).

As we have multiple devices with heterogeneous rate requirements, and different variations over time, but only one master, we introduce the figure of *sub-agents*. A sub-agent receives the rate change from the agent and changes state. The master performs an action, i.e., selects a device to query, and the sub-agent changes state (i.e., the master polls a device and updates its transmission requirement, while the sub-agent transits eventually in a different state).

The set A of actions represents transitions between states. We have two types of actions: *mandatory* actions, in which the sub-agent has to move to another state, and *non-mandatory* actions, in which the agent asks the sub-agent to move to another state in a set of states, but sub-agent's decision depends on the corresponding expected reward (it may happen that it remains in the same state).

Let us now see how RL can be used to learn and monitor devices' behaviors. To keep track of device's states and actions, we introduce a graph $G_{dv} = \{S, E, W_{dv}\}$, in which $S$ is the set of states $\{s_i|1 \leq i \leq n\}$, $E$ is the set of directed edges, where $e_{i\to j}$ represents a transition from state $s_i$ to $s_j$, and $W_{dv}$ is set of weights labeling edges $w_{i\to j}$. In our model, a state $s_i$ corresponds to a transmission rate $tr_i$, an edge $e_{i\to j}$ corresponds to the action $a_j$ in $A$, with $(i, j)$ any couple of nodes in $G$, and a weight $w_{i\to j}$ is tied to the action of going from state $i$ to state $j$. While the sets $S$ and $E$ are the same for all devices, $W_{dv}$ depends on the specific transmission requirement of each device $dv$. As the transmission requirement depends on the specific device behaviour, each device is related to a different set $W_{dv}$. We define the reward function as in Equation (1),

$$R(s_i, a_j) = w_{i\to j}. \tag{1}$$

Each time a weight $w_{i\to j}$ is updated, we perform a Softmax [35] on the subset of rewards associated with the set of actions that can be taken from state $s_i$, so as to compress the values into the range of [0,1]. All the values of such a subset after Softmax sum to 1, meaning that the function $R(s_i, a)$ has the property that

$$\sum_{\forall a \in A} R(s_{current}, a) = 1. \tag{2}$$

Periodically, the master performs an inventory phase in which it assigns an ID to each device to identify it in the communication protocol. This inventory stage supports new devices being added or removed from the environment. Once the master knows all currently active devices in the system, it places them in the state with the lowest requirement (i.e., $s_n$). Then the agent schedules transmissions in a frame slots. Transmission scheduling is ruled by a EDF-based mechanism that is explained in Section 5. When a device is polled, it replies with a packet containing the last *data sample* and the value of a *counter of changes* representing the number of new samples generated since the last query.

When the master receives a packet from a device, it observes the counter of changes. If $counter = 1$, then the device has a new data sample since the last query, it means that polling

frequency is appropriate. In this condition, the master communicates to the corresponding sub-agent to remain in the current state. If $counter = 0$ (i.e., the device does not have new data since the last query) or $counter > 1$ (i.e., the device has generated multiple new data instances since the last query), then the sub-agent is scheduling transmissions with too high or too low frequency. The sub-agent should change state, requiring a different polling frequency. Then the master asks (in a non-mandatory way) the sub-agent to move to an upper or lower state (i.e., a state with a higher or a lower frequency).

In case counter=0 then the sub-agent is asked to move to a lower state belonging to a subset $S_{allowed} \subset S$ of possible states. For example, for a device $dv$ with counter=0 and transmission rate corresponding to $s_2$, the master asks the sub-agent, in a non-mandatory way, to move to a state with a lower requirement, e.g., a state in the set $S_{allowed} = \{s_3, s_4, \ldots, s_n\}$. Instead, for a device $dv$ with counter>1 and transmission rate $s_4$, the set of allowed states is $S_{allowed} = \{s_1, s_2, s_3\}$.

Devices can move to a state in $S_{allowed}$ or remain in the current one. How does a device choose the next state? The decision is probabilistic. The probability to move to a state $s_i \in S_{allowed}$ is given by the reward $R(s_{current}, a_i)$, where $a_i$ is the action of moving to $s_i$. The outgoing edge with the highest reward has the highest probability to be selected. Please note that for each node there is also a self-loop edge, and the reward associated to this edge is given by the formula in Equation (3),

$$R(s_{current}, a_{current}) = \sum_{\forall s_j \in (S - S_{allowed})} R(s_{current}, a_{sj}). \tag{3}$$

To guarantee a wide exploration of states, with probability $1 - \epsilon$ the master selects the next state based on the reward mechanism, while with probability $\epsilon$ (which typically assumes a value around 0.1 [35]) the master randomly selects the next state among the allowed states and the current state. Thus, our device modeling presents two main properties: (1) it explores the space of the states; (2) it rapidly moves a device to the most appropriate state, jumping less promising states, guaranteeing fast reaction to changes.

There is one last point that remains to be addressed: how are rewards updated? The idea is to repay successful actions—corresponding to edges that bring devices into states in which their counter remains unitary—and penalize those actions that bring devices in states in which their counter becomes 0 or greater than 1. To quantify this idea, let us suppose at time $t$ device $dv$ is in state $s_i$ and takes action $a_j$ (i.e., moves from state $s_i$ to state $s_j$, following the edge $e_{i \to j}$ that has weight $w^t_{i \to j}$). After the device performed the action, at time $t + m$, the weight $w^{t+m}_{i \to j}$ is updated as in Equation (4),

$$w^{t+m}_{i \to j} = w^t_{i \to j} + \alpha * return, \tag{4}$$

where $\alpha$ is a *weighting factor* and $return$ is the reward corresponding the execution of action $a_j$, and has been empirically evaluated as 2 if $counter = 1$, $-0.2$ otherwise.

## 5 RELEDF: SCHEDULING TRANSMISSIONS

We now describe how the reader assigns the transmission channel. Channel access is based on a TDMA controlled by the reader. Time is divided into frames, and each frame is divided into slots. All slots have the same length, and the reader signals the beginning of a new slot by sending a query containing the ID of the device that has to answer next. The reader exploits the feedback from the learning mechanism to schedule transmissions: The aim is to assign slots to devices based on their communication requirements, i.e., how frequently they need to transmit data.

An efficient way to coordinate the polling process is through a dynamic real-time scheduling algorithm, following a single processor approach (we have a single reader controlling the system). In particular, we adopt a EDF algorithm, as it allows to optimally schedule devices transmissions,

guaranteeing deadlines (i.e., periods). The idea is to map devices polling to tasks whose deadlines are based on devices' communication needs. This means that a device with a high transmission rate will have a short deadline, while a device with low transmission rate will have a longer deadline. Tasks are periodic and deadlines may cause a *mode change*: A joystick may need to be polled once per second while not in-use (only to check if a player is starting using it) and hundreds of times per second while in use. Passing from the off state to the in-use state, the joystick causes a change in the deadline of the related task, requiring a new scheduling and thus requiring a mode change.

Another aspect to be considered regards the size of data samples. Some devices need to send only a few bits, others instead need to send a large number of bits. In each slot it is possible to transmit a limited number of bits, depending on the encoding adopted. When a tag has long data to transmit to the reader, it segments the information in multiple packets. Thus, in each slot, a single packet is transmitted. For this reason, the system can be considered as *preemptive*: Even if a device needs multiple slots to send its data sample, its assigned slots do not have to be contiguous, as the reader is able to reassemble the information segmented in different packets.

### 5.1 Brief Introduction to EDF

We now give a brief introduction to EDF and real time scheduling, while an extensive description can be found in Reference [32].

EDF is a priority policy for scheduling tasks based on deadlines. A task $\tau_i$ consists in a minimum atomic executable entity of work and is characterized by a worst-case execution time $C_i$ (the maximum amount of time within which the work has to be accomplished) and a time constraint, *deadline*, $P_i$ (the period within which the task has to be performed). Given a set of real-time tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ to be scheduled, at each time slot EDF will execute the task closest to its deadline. In this work a task represents a communication between two devices. To guarantee that each device will communicate at least once in a certain time, we consider only periodic tasks, i.e., their instance must execute regularly once per period. Moreover, we set their period equal to their deadline $P_i$. Furthermore, only synchronous tasks are considered, meaning that all their first instances are released at the same time, commonly considered time zero. Tasks executions are scheduled by EDF inside a *frame*, which is a sequence of time slots. In each slot only one task can be executed, and EDF decides which one. A task can be scheduled multiple times in a frame and each execution is called *job* of the task. Each job $j$ is characterized by a *release time* $r_j$ (defined as the point in time at which the job becomes ready to be executed) and a *deadline* $d_j$ (defined as the point in time by which the job must be completed).

Table 1 reports the notation used in the rest of the article.

The set $TR$ of rate requirements is defined as $\{tr_1 = 1/20, tr_2 = 1/50, tr_3 = 1/100, tr_4 = 1/200, tr_5 = 1/500, tr_6 = 1/2{,}000\}$, where $1/x$ means once every $x$ slots. This set has been defined empirically to include a reasonable set of values. However, other rates may be employed. Regarding the mapping $L$, please note that at any instant of time the same requirement may represent the state of multiple devices, while other requirements may not represent any device. A task $\tau_i$, as described in Reference [32], should have a name and a period. In our case there is a one to one correspondence between devices and tasks. The name of task $\tau_i$ is given by the *id* associated with the device $dv_i$ and the period of task $\tau_i$ is the current requirement of device $dv_i$.

### 5.2 Mapping Learned Transmission Rates to Tasks

Each device $dv_i \in DV$ is linked in $L$ to a specific transmission rate $tr_j \in TR$, depending on the device current behavior. For example at time $t_1$, three devices, $DV = \{dv_1, dv_2, dv_3\}$, may be mapped on the same transmission rate, $L = \{(dv_1 \rightarrow tr_1), (dv_2 \rightarrow tr_1), (dv_3 \rightarrow tr_1)\}$. ReLEDF performs an

Table 1. Notation

| Notation | Description |
|---|---|
| MS | the master, a device dedicated to poll all other devices in the network |
| $DV$ | set $\{dv_1, dv_2, \ldots, dv_n\}$ of devices that want to communicate with MS |
| $TR$ | set $\{tr_1, tr_2, \ldots, tr_m\}$ of rate requirements (i.e., polling frequencies) ordered by the highest to the lowest |
| $L$ | dynamic mapping ($dv_i \in DV \rightarrow tr_i \in TR$) reflecting the current state of each device |
| $\mathcal{T}$ | a set $\{\tau_1, \tau_2, \ldots, \tau_n\}$ of current periodic tasks to be scheduled by the master |
| $\tau_i$ | the $i$th task |
| $j_k^i$ | the $k$th job of the $i$th task |
| $r_j$ | the release time of the job j |
| $d_j$ | the deadline of the job j |
| $\mathcal{M}$ | the set of mode changes |
| $m_0$ | system initial mode |
| $m_z$ | the $z$th mode change |
| $t_{m_z}$ | the time in which happens the $z$th mode change |
| $\mathcal{T}^z$ | set of real-time tasks after the mode change $m_z$ |
| $C_i^z$ | the execution time of the $i$th task after $z$th mode change |
| $P_i^z$ | the period/deadline of the $i$th task after $z$th mode change |

inventory phase at the beginning of each frame so as to detect new devices joining the network. As the behavior of the devices is not known, they are mapped to a medium transmission rate, i.e., $tr_{i/2}$, where $i$ is the cardinality of set $TR$. Protocol operation will then allow for an eventually more appropriate transmission rates.

Transmission rates are used to create the related tasks that are scheduled by EDF. A task $\tau_i$ is defined by a name, that in our case is the *device id*; a period $P_i$; and an execution time $C_i$. Each device $dv_i \in DV$, with $(dv_i \rightarrow tr_j) \in L$, has an associated task $\tau_i$ with a unitary execution time ($C_i = 1$) and period $P_i = \frac{1}{tr_j}$. We consider unitary execution time as each task instance is executed in a single slot, due to the TDMA approach.

Given a set of tasks, EDF creates a schedule that satisfy all the tasks deadlines. It is demonstrated that if a feasible schedule exists, then it will be found by EDF [32]. In case one or more devices change transmission requirements, the related set of task has to change deadlines. Let us suppose that at time $t_1$ the set of devices $DV = \{dv_1, dv_2, dv_3\}$ has mapping $L = \{(dv_1 \rightarrow tr_1), (dv_2 \rightarrow tr_1), (dv_3 \rightarrow tr_1)\}$, while at time $t_2$, the mapping changes in $L = \{(dv_1 \rightarrow tr_3), (dv_2 \rightarrow tr_1), (dv_3 \rightarrow tr_1)\}$. The device $dv_1$ could be a presence sensor that at time $t_1$ needs transmission rate corresponding to $tr_1$ because of the presence of many people, while it requires a lower transmission rate, e.g., $tr_3$, at time $t_2$ when few people are present in the environment. Then device $dv_1$ causes a mode change in the system, requiring EDF to produce a new schedule, as presented in Section 5.5.

### 5.3 Case Study

We show an example that serves as case study throughout the article. Let us consider a set $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_8\}$ of eight tasks with unitary execution time. For each task $\tau_i$, EDF generates a set of jobs, each one identified by $j_k^i[r, d)$, where $i$ represents the task, $k$ the job instance, $r$ the release time, and $d$ the deadline. Figure 2 shows the jobs generated by EDF for each task in $\mathcal{T}$, and its corresponding period. Notice that, the jobs are generated according the **Least Common Multiple (LCM)** of tasks' periods, i.e., each task $\tau_i$ has exactly $\frac{LCM}{P_i}$ number of jobs in each frame. Figure 3 shows the scheduled frame that is composed of 20 slots according to the LCM. Time slots in the frame are indicated by $t_s$, where $s$ is the index of the slot. Notice that EDF uses a priority scheduling policy based on deadlines, i.e., at any time the algorithm will execute the job closest to its deadline.

| Task | Period | Generated Jobs | | | |
|---|---|---|---|---|---|
| $\tau_1$ | 5 | $j_1^1[0,5)$ | $j_2^1[5,10)$ | $j_3^1[10,15)$ | $j_4^1[15,20)$ |
| $\tau_2$ | 5 | $j_1^2[0,5)$ | $j_2^2[5,10)$ | $j_3^2[10,15)$ | $j_4^2[15,20)$ |
| $\tau_3$ | 5 | $j_1^3[0,5)$ | $j_2^3[5,10)$ | $j_3^3[10,15)$ | $j_4^3[15,20)$ |
| $\tau_4$ | 10 | $j_1^4[0,10)$ | $j_2^4[10,20)$ | | |
| $\tau_5$ | 10 | $j_1^5[0,10)$ | $j_2^5[10,20)$ | | |
| $\tau_6$ | 20 | $j_1^6[0,20)$ | | | |
| $\tau_7$ | 20 | $j_1^7[0,20)$ | | | |
| $\tau_8$ | 20 | $j_1^8[0,20)$ | | | |

Fig. 2. Jobs generated by EDF.

| Time slot | $t_0$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ | $t_9$ | $t_{10}$ | $t_{11}$ | $t_{12}$ | $t_{13}$ | $t_{14}$ | $t_{15}$ | $t_{16}$ | $t_{17}$ | $t_{18}$ | $t_{19}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job | $j_1^1$ | $j_1^2$ | $j_1^3$ | $j_1^4$ | $j_1^5$ | $j_2^1$ | $j_2^2$ | $j_2^3$ | $j_1^8$ | $j_1^6$ | $j_3^1$ | $j_3^2$ | $j_3^3$ | $j_2^4$ | $j_2^5$ | $j_4^1$ | $j_4^2$ | $j_4^3$ | $j_1^7$ | |

Fig. 3. Initial schedule.

### 5.4 EDF Feasibility Analysis

We now present the main techniques to assess scheduling feasibility for a set of tasks $\mathcal{T}$, i.e., to establish whether it is possible to schedule all tasks such that so as to meet their deadlines. To this end we first introduce some definitions.

Given a set of real-time tasks $\mathcal{T}$, and an interval of time $[t_1, t_2)$, we define:

- the *process demand*, as the computational time units required by the tasks instances (jobs) in that interval of time: $h[t_1, t_2) = \sum_{t_2 \le r_k, d_k \le t_1} C_k$.
- the *loading factor*, as the fraction of the interval needed to execute its jobs: $u_{[t_1, t_2)} = \frac{h_{[t_1, t_2)}}{t_2 - t_1}$

By means of these definitions it is possible to define the necessary feasibility condition for the scheduling algorithm. The condition requires that, for any given interval $[t_1, t_2)$ of the scheduled frame, the *loading factor* $u_{[t_1, t_2)}$ is not greater than 1 (i.e., the jobs can be executed within a fraction of the interval). We recall that for periodic tasks, the frame scheduled by EDF is composed of the minimum sequence of jobs such that it is possible to create all task instances, with release times and deadlines. Once the scheduled frame is created, the scheduling sequence continuously repeats. Thus, to verify the feasibility of a scheduled frame, we introduce the *absolute loading factor*, which is defined as the maximum of all possible intervals loading factor: $u = \sup_{0 \le t_1 < t_2} u_{[t_1, t_2)}$.

Each set of real-time tasks is feasibly scheduled by EDF if *and* only if its *absolute loading factor* $u$ is such that $u < 1$, according to Spuri Theorem [31]. Nevertheless, this feasibility analysis requires to check all the possible intervals $[t_1, t_2)$, requiring significant computational complexity. Liu and Layland [16] propose an optimized condition for feasibility under EDF for synchronous periodic tasks. Without loss of generality, they consider only the execution time ($C_i$) and period ($P_i$) of tasks in $\mathcal{T}$, without having to study each job in each possible period of the scheduled frame.

Corollary 1 (Liu and Layland). *Any set of n synchronous periodic tasks with processor utilization* $\mathcal{U} = \sum_{i=1}^{n} \frac{C_i}{P_i}$ *is feasibly scheduled by EDF if and only if* $\mathcal{U} \le 1$.

Corollary (1) can be easily demonstrated as a consequence of Spuri Theorem, the formal proof is given in Reference [32].

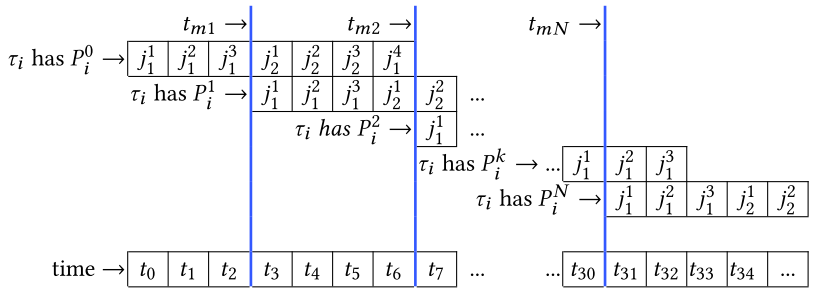


Fig. 4. Example of a frame that is interrupted, with un-executed tasks to be rescheduled.

### 5.5 EDF with Mode Changes

Let us now consider the application of EDF in our dynamic environment. During the execution of a scheduled frame, it may happen that (i) a task changes its period (i.e., a device changes its transmission requirement), (ii) new tasks enter $\mathcal{T}$ (i.e., a new device joins the network), and (iii) one or more tasks leave $\mathcal{T}$ (i.e., devices leaving the network). All these events produce a mode change that has to be addressed.

EDF adopts one of the following two policies: (1) Wait for the end of the current frame and then create a new schedule with updated tasks and (2) interrupt the current frame, create a new schedule and immediately start it. However, both policies are inefficient in our context. The first one can introduce delays, i.e., a task that has changed its period could wait long time before being executed as required. For example, if $\tau_i$ changes its period at the beginning of a schedule, from $P_i^0$ to $P_i^1$ s.t. $P_i^1 \ll P_i^0$, then it could wait long before being executed with the new period $P_i^1$, due to other jobs left in the scheduled frame. The second policy can lead to starvation, i.e., a task may never be executed. For example, assume that $\tau_i$ is a task placed at the end of the schedule, and there are some tasks at the beginning of the schedule that continuously change their periods, then the policy will continuously interrupt the current schedule to create a new one. Thus, as long as there are changes, $\tau_i$ will be postponed and never executed.

To overcome these limitations we propose a new scheduling approach based on EDF, able to reduce delays and avoid task starvation.

## 6 EDF-IJ: A NEW SCHEDULING POLICY

We propose a new scheduling policy, EDF-IJ. The main idea is to immediately interrupt the current schedule when a mode change happens, and create a new schedule, keeping some information regarding the un-executed jobs of the interrupted frame. The quick creation of a new schedule allows fast changes and reduces delays, while the information about the previous one helps to avoid starvation.

Let $\mathcal{M} = \{m_0, m_1, m_2, \ldots\}$ be the set of possible mode changes during the execution, and let $m_0$ be the initial system mode. In EDF-IJ for any mode change $m_i$, an interrupt is generated, the current schedule is cut, and consequently a new schedule is produced. Figure 4 shows an example of a frame that is interrupted, leaving some jobs un-executed; they should be rescheduled. Let $t_{m_1}$ be the time slot in which the first mode change $m_1$ happens (i.e., first cut), and let $t_{m_2}$ the time slot of the second mode change (i.e., second cut), and so on. We define $P_i^0$ as the period of task $\tau_i$ before the first mode change $t_{m_1}$, $P_i^1$ the period of task $\tau_i$ before $t_{m_2}$ and after $t_{m_1}$, and so on. For a mode change $z$, if the task $\tau_i$ changes its period, then $P_i^{z-1} \neq P_i^z$. Instead, if a task $\tau_i$ does not change its period, then $P_i^{z-1} = P_i^z$. In the new frame all tasks should be scheduled keeping their periods, except for those that have caused the interrupt (as they have new periods). We also mention that, to

| task | New Period | Inherited Deadline | First job Deadline |
|---|---|---|---|
| $\tau_1$ | 5 | 20-13=7 | 5 |
| $\tau_2$ | 5 | 20-13=7 | 5 |
| $\tau_3$ | 5 | 15-13=2 | **2** |
| $\tau_4$ | 10 | 20-13=7 | **7** |
| $\tau_5$ | 10 | 20-13=7 | **7** |
| $\tau_6$ | 20 | // | 20 |
| $\tau_7$ | 20 | 20-13=7 | **7** |
| $\tau_8$ | **10** | // | **10** |

Fig. 5. Inherited jobs table.

avoid starvation, the policy keeps some information regarding previous un-executed jobs. If these jobs are not scheduled by taking care of their old deadlines, then it is possible to cause starvation: Task instances placed at the end of the schedule will never be executed if they are preceded by jobs of tasks that continuously change their periods). Thus, we introduce the inherited deadlines. For each task that has not changed its period, the first un-executed instance after the mode change is inherited in the new schedule and, this instance, will have its deadline equals to the minimum between the inherited job deadline and the newly generated one. The inherited deadline is equal to the deadline of the first un-executed job (after the mode change) minus the time at which the mode change occurred plus 1.

In particular, the inherited deadlines are assigned to any first instance of $\tau_i \in \mathcal{T}$ s.t. $\exists\, j_j^i[r_j, d_j)$ not yet scheduled with $d_j > t_m \geq r_j$. Their inherited deadlines are computed using the following formula:

$$d_j = min(P_j, d_j - (t_{m_z} + 1)). \tag{5}$$

Only the first instances of tasks may have inherited deadlines; these jobs represent the inherited jobs from the previous schedule. For example, let us consider the beginning of a schedule and let us suppose there exists an instance of the task $\tau_i$ not yet executed at time slot $t_{m_1}$, the time at which a task changes its period and generates a cut in the schedule. Let $P_i$ be the task period of $\tau_i$, with $P_i^0 = P_i^1$. Thus, the *new schedule* will have as first job instance of $\tau_i$ ($j_1^i$), i.e., *the inherited job*, with a release time 0 and deadline equals to $(P_i - (t_{m_1} + 1))$. This deadline is the minimum between the new generated deadline, $P_i$, and the inherited one, $P_i - (t_{m_1} + 1)$. The next instances of $\tau_i$ in the *new schedule* will be normally scheduled according to $P_i$ (i.e., $[P_i, 2 \cdot P_i)$, $[2 \cdot P_i, 3 \cdot P_i)$ and so on).

The EDF-IJ scheduling policy is shown in Algorithm 1. Every time a mode change happens, the current schedule is interrupted and the algorithm provides a new one. Practically, it computes the job instances for all the tasks; checks the feasibility of the schedule (Section 7) and, in case it is not feasible, it uses the **Distributed Loss resolution algorithm (DL-resolution)** algorithm (Section 8) to solve the overload schedule problem; finally, it computes the scheduled frame by using a similar EDF policy.

### 6.1 Example of the New Scheduling Policy

Let us consider the set of tasks $\mathcal{T}$ depicted in Figure 2, the schedule of Figure 3, and the occurrence of a mode change $m_1$ at the end of time slot $t_{12}$, induced by task $\tau_8$, which changed period from 20 to 10. Then, EDF-IJ creates a table (see Figure 5) where

(1) The first column contains the names of tasks in $\mathcal{T}$.

**ALGORITHM 1:** EDF-IJ

**Input**: The current schedule, a set of real-time tasks $\mathcal{T}$, a mode change $m$
**Result**: A feasible schedule

**set** $\mathcal{T}' \leftarrow \mathcal{T}$ after the mode change $m$;
**set** $\mathcal{J} \leftarrow$ jobs set to schedule;
**for** $\tau_i \in \mathcal{T}'$ **do**
  **if** $\tau_i \notin \mathcal{T} \vee P_i^m \neq P_i^{m-1}$ **then**
    $\mathcal{J} += \{j_1^i[0, P_i^m),\ j_2^i[P_i^m), 2P_i^m), \ldots\}$;
  **end**
  **else**
    **if** $\exists j_z^i[r_z, d_z)$ *not scheduled s.t.* $d_z > t_m > r_z$ **then**
      $d_z = min(P_z, d_z - t_m - 1)$;
    **end**
    **else**
      $d_z = P_z$;
    **end**
    $\mathcal{J} += \{j_1^i[0, d_z),\ j_2^i[P_i^m, 2P_i^m), \ldots\}$;
  **end**
**end**
**if** *not feasibility*$(\mathcal{T}', \mathcal{J})$;
**then**
  DL-resolution$(\mathcal{T}', Ld)$ ;
**end**
**return** EDF$(\mathcal{J})$;

(2) The second column indicates the new period after $m_1$. In this case, all jobs have the same period depicted in Figure 2, apart from task $\tau_8$, which just changed it.
(3) The third column contains for each task the "inherited deadline." If a task has an un-executed job, its "inherited deadline" is given by the deadline of the first un-executed job of such task (after $m_1$) minus the time at which $m_1$ occurred plus 1 (namely the time of the slot after the cut, $t_{12+1}$). Looking at Figure 3, after $t_{12}$ the jobs we consider are $j_4^1, j_4^2, j_3^3, j_2^4, j_2^5, j_1^6$. For example, $\tau_5$ has a deadline of 20, $m_1$ occurs at $t_{12}$, so its inherited deadline is $20 - (12 + 1) = 7$.
(4) The fourth column shows the minimum between the second and the third column. In case the third column is empty, it considers the second column value.

Now EDF is able to generate the new jobs, as presented in Figure 6. In particular, for the first instance of each task, the deadline will be the time reported in column "First Job Deadline" in Figure 5. The new schedule is depicted in Figure 7. Whenever a task changes its periods, and before starting a new schedule, the policy must assert the feasibility (or un-feasibility) of the new tasks set $\mathcal{T}$ under EDF and, moreover, it must be aware of inherited jobs and deadlines.

## 7 FEASIBILITY ANALYSIS

The new scheduling policy requires after each mode change a feasibility analysis. As tasks can change period, it may happen that the new periods do not allow the creation of a feasible schedule. For example, let $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ be the task set s.t. all of them have an execution time 1 and a period $P = 4$. If $\tau_1$ reduces its period $4 \Rightarrow 3$, then $\mathcal{U} = \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{3} = \frac{13}{12} > 1$, and thus no feasible schedule exists according to Corollary 1.

| task | Period | Generated | jobs | | |
|---|---|---|---|---|---|
| $\tau_1$ | 5 | $j_1^1$[0,5) | $j_2^1$[5,10) | $j_3^1$[10,15) | $j_4^1$[15,20) |
| $\tau_2$ | 5 | $j_1^2$[0,5) | $j_2^2$[5,10) | $j_3^2$[10,15) | $j_4^2$[15,20) |
| $\tau_3$ | 5 | $j_1^3$**[0,2)** | $j_2^3$[5,10) | $j_3^3$[10,15) | $j_4^3$[15,20) |
| $\tau_4$ | 10 | $j_1^4$**[0,7)** | $j_2^4$[10,20) | | |
| $\tau_5$ | 10 | $j_1^5$**[0,7)** | $j_2^5$[10,20) | | |
| $\tau_6$ | 20 | $j_1^6$[0,20) | | | |
| $\tau_7$ | 20 | $j_1^7$**[0,7)** | | | |
| $\tau_8$ | 10 | $j_1^8$**[0,7)** | $j_2^8$**[10,20)** | | |

Fig. 6. Jobs generated by EDF after cut0.

| Time slot | $t_0$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ | $t_9$ | $t_{10}$ | $t_{11}$ | $t_{12}$ | $t_{13}$ | $t_{14}$ | $t_{15}$ | $t_{16}$ | $t_{17}$ | $t_{18}$ | $t_{19}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job | $j_1^3$ | $j_1^1$ | $j_1^2$ | $j_1^4$ | $j_1^5$ | $j_1^7$ | $j_1^8$ | $j_2^1$ | $j_2^2$ | $j_2^3$ | $j_3^1$ | $j_3^2$ | $j_3^3$ | $j_1^6$ | $j_2^8$ | $j_4^1$ | $j_4^2$ | $j_4^3$ | $j_2^4$ | $j_2^5$ |

Fig. 7. Initial schedule.

| **Task** | **First schedule** | | **1° mode change** | | **2° mode change** | |
|---|---|---|---|---|---|---|
| | Period | Jobs | Period | Jobs | Period | Jobs |
| $\tau_1$ | 4 | $j_1^1[0,4)$ | $\underline{\emptyset}$ | $\underline{\emptyset}$ | $\emptyset$ | $\emptyset$ |
| $\tau_2$ | 4 | $j_1^2[0,4)$ | 4 | $j_1^2[0,3)$ | $\underline{2}$ | $j_1^2[0,2)$, $j_2^2[2,4)$ |
| $\tau_3$ | 4 | $j_1^3[0,4)$ | 4 | $j_1^3[0,3)$ | 4 | $j_1^3[0,2)$ |
| $\tau_4$ | 4 | $j_1^4[0,4)$ | 4 | $j_1^4[0,3)$ | 4 | $j_1^4[0,2)$ |
| $\mathcal{U}$ | **1** | | $\underline{3/4}$ | | $\underline{1}$ | |

Fig. 8. Example in which Corollary 1 is not applicable.

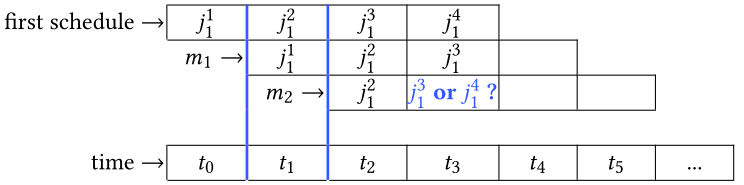


Fig. 9. Example in which Corollary 1 is not applicable.

Furthermore, as presented in Section 6, the new proposed scheduling policy considers also inherited deadlines from the previous schedule. Even if these deadlines affect only the first instance of some tasks they could make the feasibility analysis (Corollary 5) for EDF not applicable in our case.

### 7.1 In-applicability of EDF Feasibility Analysis

In Figure 8 and Figure 9 we show an example in which Corollary 1 is not sufficient to provide schedule feasibility in presence of inherited deadlines. In this example $\mathcal{T}$ is composed of four periodic synchronous tasks: $\tau_1, \tau_2, \tau_3, \tau_4$, all of them with a period $P = 4$ and execution time 1. After time $t_0$, in which $j_1^1$ is executed, the task $\tau_1$ is excluded from $\mathcal{T}$ as it is not required anymore. Notice that even if a task leaves $\mathcal{T}$ it is useful to spawn a mode change and generate a new schedule as it is

possible to optimize the execution of tasks, especially in relation with the new overload scheduling policy presented in Section 8. After time $t_1$, in which $j_1^2$ is executed, the task $\tau_2$ changes its period to a more demanding one ($4 \Rightarrow 2$). Thus, a second mode change ($m_2$) is spawned. The feasibility analysis, carried out using Corollary 1, shows that the task set $\mathcal{T}$ is feasible, as $\mathcal{U} = \frac{1}{4} + \frac{1}{4} + \frac{1}{2} = 1$. Instead, as the scheduling policy inherits also deadlines from the previous schedule for tasks $\tau_2$ and $\tau_3$, jobs $j_1^2$, $j_1^3$, and $j_1^4$ will have the same deadlines equals to time slot $t_2$, and thus the schedule will be unfeasible. From this example it is possible to conclude that Corollary 1 is not enough to test the schedule feasibility for our proposal, and hence a new feasibility test for the proposed schedule policy is needed.

### 7.2 A New Feasibility Analysis for Inherited Jobs

As mentioned in the previous section, Corollary 1 is not enough to test the schedule feasibility when a mode change happens and we have some inherited jobs and deadlines. Thus, we first define the *initial process demand* and *initial loading factor* with inherited jobs and we propose a new feasibility test. We recall that, the job set IJ is composed of all the inherited jobs, i.e., the first task instances in the new schedule with inherited deadlines.

*Definition 2.* Given a set of real-time tasks $\mathcal{T}$, a set IJ of inherited jobs, and an interval of time $[0, t_1)$, the *initial process demand with inherited jobs* is defined as the computational units required, by the task instances and inherited jobs, in the first interval of time up to $t_1$.

$$h_{IJ}[0, t_1) = \sum_{\substack{\tau_i \in \mathcal{T} \ s.t. \\ \nexists \ j_k^i \in IJ}} \left\lfloor \frac{C_i}{P_i} \cdot t_1 \right\rfloor + \sum_{\substack{\tau_i \in \mathcal{T} \ s.t. \\ \exists \ j_k^i \in IJ \ \wedge \ d_j < t_1}} C_i + \max\left(0, \left\lfloor \frac{C_i}{P_i} \cdot t_1 - C_i \right\rfloor\right). \tag{6}$$

Equation (6) is composed of two main summations.

- The first summation regards only tasks without instances in $IJ$. For each of these tasks, the summation returns how many jobs they have with deadlines less than $t_1$. Notice that, the floor function is used to comply with the equation on process demand, i.e., the fractional part of the sum argument represents a job that has a release time less than $t_1$ but a deadline greater than $t_1$.
- The second summation regards only tasks with an instance in $IJ$ with deadline less than $t_1$. Notice that, these instances have always deadlines less or equals than their original task deadlines, as they are inherited. For each of these tasks, we are counting one job ($C_i$) as the one from $IJ$ with inherited deadlines, plus the number of their normal instances in this interval, minus the one inherited. Again, the floor function is used to comply with the process demand equation. Notice also that, the "minus $C_i$" is relevant only when $\frac{t_1}{P_i} \geq 1$.

*Definition 3.* Given a set of real-time tasks $\mathcal{T}$, a set IJ of inherited jobs, and an interval of $[0,t_1)$, the *initial loading factor with inherited jobs* is defined as the fraction of the interval needed to execute its jobs plus the inherited ones,

$$u_{IJ}[0, t_1) = \frac{h_{IJ}[0, t_1)}{t_1}. \tag{7}$$

In Theorem 4, we propose a new feasibility analysis for sets of real-time tasks plus inherited jobs.

Theorem 4. *Each set of real-time tasks $\mathcal{T}$ with a set of inherited jobs, $IJ$, is feasibly scheduled by EDF-IJ if and only if:*

- *Corollary 1 is verified on* $\mathcal{T}$*, i.e.,* $\mathcal{U} \leq 1$
- $\forall j \in IJ$ *the loading factor* $u_{IJ}[0, d_j) \leq 1$.

Proof. We prove that our Theorem satisfies the necessary feasibility condition for scheduling real-time tasks (Spuri Theorem 5.4). The first condition guarantees that, in absence of inherited jobs IJ, all the intervals $[t_1, t_2)$ of the schedule have a loading factor $u_{[t1,t2)} \leq 1$, as direct consequence of Corollary 1 and Theorem 5.4. This is a necessary condition for the feasibility test also with inherited jobs. However, the inherited jobs could still overload some intervals and make the first condition not sufficient. In particular, they can overload only intervals of the form $[0, d_j)$, where $j \in IJ$: As they can be only the first instances of tasks and, according to the definition of inherited deadlines, they can only be anticipated respect to the normal execution, they can overload only the intervals at the beginning of the schedule, up to their inherited deadlines. Therefore, we show that the second condition can check only these intervals to complete the feasibility analysis for inherited jobs. In particular:

- It is not necessary to check any interval $[t_i, d_j)$, i > 0, because if there exists an overload in $[t_i, d_j)$, then also $[0, d_j)$ must be overloaded: As we are working with periodic synchronous tasks, i.e., they start together at time 0, by Theorem 3.14 in Reference [32] it is not possible to have an idle time slot before an overload, thus any overload must involve also the slot 0.
- It is not necessary to check any interval $[0, d_j - i)$, $0 < \mathrm{i} < d_j$, because an overload on it can be caused only by a job not in $IJ$, which would be already revealed by Corollary 1, or by another job in $IJ$ with deadline different from $d_j$, which would be revealed by our feasibility analysis.

In summary, the first theorem condition guarantees that, for all couples $[d_j, t_k)$, s.t. $\mathrm{k} > \mathrm{d} \wedge j \in IJ$, the loading factor will be less than 1, as the inherited jobs will not affect them; the second condition of the feasibility analysis guarantees that the loading factor in the intervals $[0, d_j)$, where $j \in IJ$, will be less than 1, by definition of initial loading factor with inherited jobs, Equation (7). □

### 7.3 Example of Application of the Feasibility Test on an EDF-IJ Schedule

Let us take again $\mathcal{T} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$, all with the same period 4 and execution time 1. An example is depicted in Figures 8 and 9. At time $t_0$, $IJ$ is empty. In this case, the EDF-IJ policy works exactly as *EDF*, as no instance should be "anticipated." The feasibility test is performed, following theorem 7.2, in the following way:

(1) Verify Corollary 1 on $\mathcal{T}$. This job is easy performed, and returns that: $\mathcal{U} = \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} = 1$.
(2) As $IJ$ is empty, no further actions are required.

A feasible schedule generated from $\mathcal{T}$ is shown in Figure 9, indicated as "First schedule." After $t = 0$, task $\tau_1$ leaves $\mathcal{T}$: A mode change ($m_1$) interrupts the current schedule. $\mathcal{T}^1 = \{\tau_2, \tau_3, \tau_4\}$ and in this case $IJ = \{j_1^2[0, 3), j_1^3[0, 3), j_1^4[0, 3)\}$. As before, a feasibility test is required. The first condition of the test is easily verified as $\mathcal{U} = \frac{1}{4} + \frac{1}{4} + \frac{1}{4} = \frac{3}{4} \leq 1$. In this case, as $IJ$ is not empty, we should also verify that, for each deadline $d_j$, $j \in IJ$, $u_{IJ}[0, d_j) \leq 1$, as shown in Equation (8). Notice that all tasks in $\mathcal{T}^1$ have an inherited job in $IJ$,

$$u_{IJ}[t_0, t_3) = \frac{1 + max\left(0, \left\lfloor \frac{3}{4} - 1 \right\rfloor\right) + 1 + max\left(0, \left\lfloor \frac{3}{4} - 1 \right\rfloor\right) + 1 + max\left(0, \left\lfloor \frac{3}{4} - 1 \right\rfloor\right)}{3} \leq 1. \quad (8)$$

As both conditions are verified, there exists a schedule on $\mathcal{T}^1$ that is feasible, and this schedule is presented in Figure 9 indicated with $m_1$. After $t_1$, task $\tau_2$ changes its period, from 4 to 2. A new

mode change ($m_2$) happens, $\mathcal{T}^1$ is updated becoming $\mathcal{T}^2$ and a new schedule should be created. As before, the feasibility test on $\mathcal{T}^2$ is performed. The first condition is verified, as $\mathcal{U} = \frac{1}{2} + \frac{1}{4} + \frac{1}{4} = 1$. As demonstrated in Section 7.1 this test is not enough to guarantee the feasibility of the schedule with inherited jobs. Now $IJ = \{j_1^3[0,2), j_1^4[0,2)\}$. Notice also that as task $\tau_2$ changed period, now two jobs are generated: $j_1^2[0,2)$ and $j_2^2[2,4)$. The second condition of the feasibility test must now be executed on current deadlines $d_j$, $j \in IJ$, as shown in Equation (9),

$$u_{IJ}[0,t_2) = \frac{\frac{2}{2} + 1 + max\left(0, \left\lfloor \frac{2}{4} - 1 \right\rfloor\right) + 1 + max\left(0, \left\lfloor \frac{2}{4} - 1 \right\rfloor\right)}{2} = \frac{3}{2} > 1. \tag{9}$$

As $u_{IJ}[0,t_2)$ is bigger than 1, the feasibility test indicates that does not exist a schedule feasible under EDF. Effectively, as shown in Figures 8 and 9, in $m_2$ at $t_3$ both $\tau_3$ and $\tau_4$ instances have a deadline of 2, and must be executed, but there is space just for one job. This condition introduces an overload scheduling problem.

## 8 A NEW OVERLOAD SCHEDULING POLICY

The literature proposes the following solutions to handle the overload scheduling condition [32]:

(1) Best effort - each new task is added to $\mathcal{T}$, even if the schedule is not feasible: Only tasks with shorter deadlines are executed.
(2) Admission control scheme - if the new task makes $\mathcal{T}$ not feasible, then the new task is refused.
(3) Robust scheme - when a new task enters the set $\mathcal{T}$, some tasks are removed from $\mathcal{T}$ until it becomes feasible depending on another policy different from EDF (e.g., priority policy).

These solutions perform well for real-time systems. In case of overload, they attempt to carry out as much as possible critical jobs and dismiss those with higher period (according to the first approach), the new ones in the second approach, or those with lowest priorities (according to some priority policy). Instead, in a networking context, these solutions may have severe drawbacks, bringing to an unfair channel allocation, in which some devices never communicate. While for safety critical jobs missing a deadline is not acceptable, for IoT systems such as a smart home would be preferable to miss a deadline rather than never communicate (missing a deadline introduces delay and data loss but does not compromise the system).

To solve this issue, we propose a new overload scheduling policy that accepts any change in the task set $\mathcal{T}$, and, if the new schedule is not feasible, then it fairly rules the task periods to re-establish the feasibility without removing any task. As overload scheduling often arises when some tasks require to be executed more frequently, (i.e., require shorter periods but there are no enough free slots), the policy stretches the tasks periods to resolve the overload. When applied in a network context, such policy makes possible to communicate with all the devices.

Notice that each task period is closely related to the bandwidth allocated to the corresponding device. If a task has period $P_i$, then the associated device can transmit up to $\frac{1}{P_i}$ rate, thus, as the task period increases the available bandwidth for the device decreases as well, causing data loss. In particular, if an algorithm for overload scheduling increases each period by 1, meaning that each task will miss its deadline by at most one slot, then the subsequent data loss could be unfair. For example, given two tasks $\tau_1$ and $\tau_2$, with periods $P_1 = 2$ and $P_2 = 7$, which are supposed to be increased by 1 unit, the data loss of $\tau_1$ will be around 33% while data loss of $\tau_2$ will be around 13%. To overcome this problem the increment of tasks periods is done according to the possible data loss. Thus, our proposal increases periods while it guarantees a fair distributed loss among tasks. Notice that, the execution time of tasks cannot be reduced to resolve an overload scheduling,

because it is the minimum amount of time required to have a complete and meaningful execution of a task.

### 8.1 DL-resolution Algorithm

The overload scheduling policy is implemented in the DL-resolution, shown in Algorithm 2, which takes in input a set of tasks $\mathcal{T}$ and a parameter $Ld$, and, until the set of tasks is not feasible, increases the periods of some tasks. Only the periods of those tasks that maintain their data loss below a predefined threshold, namely *lossThreshold*, are increased. This threshold is updated at each main iteration by means of the input parameter $Ld$, which controls the *fairness* of the algorithm and its applicability in network context. By setting $Ld$ to a small enough value it is possible to iteratively increase the tasks periods such that the tasks lose the same percentage of data. However, by setting $Ld$ to 1, it is possible to increase all the tasks periods by 1, such that all of them may fairly miss deadlines by one slot, regardless the potential data loss in a network context.

**ALGORITHM 2:** DL-resolution

**Input**: A set of real-time tasks $\mathcal{T}$ and a loss data step $Ld \in (0, 1]$
**Result**: A feasible set of real-time tasks $\mathcal{T}$

Set $\mathcal{T}' \leftarrow \mathcal{T}$;
Map $\mathcal{I}_{\mathcal{P}} \leftarrow \{i : 0\}_{0 \leq i < |\mathcal{T}|}$;
float lossThreshold $\leftarrow 0$;
**while** $\mathcal{T}'$ *is not schedulable* **do**
    lossThreshold += $Ld$;
    **for** $\tau_i \in \mathcal{T}$ **do**
        $P'_i \leftarrow \mathcal{I}_{\mathcal{P}}[i] + 1$;
        expectedLoss = $1 - \frac{P_i}{P'_i}$;
        **if** *expectedLoss* $\leq$ *lossThreshold* **then**
            $\mathcal{I}_{\mathcal{P}}[i]$ += 1;
            $\mathcal{T}'.update(\tau_i, P'_i)$;
        **end**
    **end**
**end**
$\mathcal{T} \leftarrow \mathcal{T}'$;

In summary, the proposed overload scheduling policy allows always scheduling all input tasks by increasing their periods, and, by means of the input parameter $Ld$, which controls the data loss threshold, it is possible to apply either a data loss fair policy, or a fair deadline miss policy.

## 9 RELEDF ALGORITHM

The ReLEDF algorithm combines the learning mechanism with the EDF-based scheduling and device polling. The main idea is that the master keeps polling devices until a change happens in devices transmission requirements. Each time a device is polled, its state in the behavioral graph $G$ is updated depending on its counter value. After such update, if the device remains in the same state, then the master keeps polling following the same schedule. Otherwise, the device model changes state allowing for a new transmission rate, and a new schedule has to be generated. The generation of a new schedule requires the creation of the *transition schedule* and, in case of overscheduling, the application of the *distributed loss* algorithm. After defining and executing the transition schedule, the master generates a schedule from $L$ by following the standard EDF policy. The pseudocode presenting the most significant steps is given in Algorithm 3.

**ALGORITHM 3:** ReLEDF Algorithm

```
   note: Variables names refers to Section 5 and Section 9
 1 Master MS; Set DV; Schedule S_current; Map L; DV = MS.obtainDevices();
 2 for dv ∈ DV do
 3 |   L.add(dv,new DevModel())
 4 end
 5 S_current = MS.schedule(L);
 6 Bool changes;
 7 while true do
 8 |   queriedDevice,ChangesCounter = MS.query();
 9 |   changes = L[queriedDevice].update(ChangesCounter);
10 |   if changes then
11 |   |   Set IJ = MS.getStarvJobs();
12 |   |   overSched, S_current = MS.schedule(L,IJ);
13 |   |   while overSched do
14 |   |   |   distributeLoss(L);
15 |   |   |   overSched, S_current = MS.schedule(L,IJ);
16 |   |   end
17 |   end
18 end
```

At the beginning, the algorithm declares the master MS, the set of devices $DV$, the running schedule $S_{current}$, and the mapping $L$. Then the algorithm, through the *MS.obtainDevices* method, starts collecting the IDs of connected devices, assigned through a preliminary inventory process. For each device, the master creates the corresponding sub-agent, to which will be assigned the lowest transmission rate state (hence the first frame will have slots equally assigned to each device). After setup, the algorithm generates a new schedule (*MS.schedule(L)*) and starts polling active devices, through the *MS.query* method. For each poll, the algorithm asks the sub-agent corresponding to the polled device to update its state, depending on the device counter value. If the sub-agent remains in the current state, then the algorithm polls the next device in the schedule. If the sub-agent changes state, then the algorithm creates a new schedule following the *Transition without Starvation Scheduling*, described in Section 6. The method *MS.getStarvTasks* returns, for each device, the first unexecuted task instance in the schedule, if there exists. If the schedule that is generated by the updated $L$ is feasible, then the algorithm updates $S_{current}$ and starts polling. Otherwise, it applies the Distribute Loss policy, described in Section 8, until it does not reach a feasible schedule. In the pseudocode we present only the most important instructions, omitting operations like the rescheduling after a "Transition without Starvation Schedule."

## 10 PERFORMANCE EVALUATION

In this section, we evaluate the performance of our protocol ReLEDF through both simulations and experiments. First, we evaluate the effectiveness of the proposed EDF-IJ scheduling, comparing it with EDF and evaluating also the overload scheduling approach against other solutions. Then, we evaluate the overall proposed protocol, namely ReLEDF, in different settings. Finally, we present the results of real experiments.

### 10.1 Scenarios

Evaluated scenarios represent a typical smart home, involving a variable number of heterogeneous smart devices (e.g., temperature, presence, remotes and cameras). As shown in Table 2,

Table 2. Scenarios Descriptions: Each Scenario Includes a Different Number of Sensors of Different Types

| | Experiments | | | | Simulations | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Case / Sensor | E1 | E2 | E3 | E4 | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 |
| Environmental | 2 | 4 | 10 | 12 | 37 | 35 | 34 | 32 | 32 | 32 | 80 | 120 |
| Remote | 1 | 2 | 3 | 4 | 2 | 3 | 3 | 4 | 4 | 4 | 8 | 18 |
| JoyStick | 1 | 1 | 2 | 3 | 1 | 2 | 3 | 4 | 4 | 4 | 8 | 8 |
| Camera | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 4 | 4 |
| Total | 4 | 8 | 16 | 20 | 40 | 40 | 40 | 40 | 41 | 42 | 100 | 150 |

experimental scenarios include 4 to 20 devices (scenarios E1 to E4), so as to increase scenario complexity. E1 represents a single room with a light sensor, a temperature sensor, a simple TV remote and a joystick. E2 represents a two-room apartment, with a light sensor and a temperature sensor per room, two TV remotes, one joystick, and one security camera outside the door. E3 and E4 represent two larger apartments, with three or four rooms and multiple sensors for each room. Simulations consider even more complex scenarios, including 40 to 150 (scenarios S1 to S8) heterogeneous devices. We remark that the value 40 has been used also in Reference [20].

Devices are simulated through Markov chains, as described in Reference [20]. Devices have a set of states (representing possible transmission rates of devices), and a set of transitions between states, where transition probabilities reflect the device probabilities of producing new data (e.g., sense a new value). If a device changes state, then a new packet is generated and sent to the destination device. Let us consider the case of a temperature sensor, which generates a new data sample each time a significant change in the temperature is supposed to be sensed. We imagine that the temperature changes on average 4 times per hour (this value can be set as preferred), therefore the probability to change state should encode around 4 changes per hour. The traffic generator flips a made-up coin every 10ms and decides whether to remain in the same state, expressing no change in sensed data, or to change state, sending the new temperature packet. Thus the probability to pass from a value X to a value Y is defined as $P(change) = 4/(3600 * 100)$, where the value $3,600 * 100$ is the number of flips per hour. More complex devices, which present multiple outgoing edges from the same state, follow a similar logic. However, transitions between states are more frequent. Consider the case of devices that interact with the user, such as TV remotes or joysticks. They have variable behavior, as they produce traffic burst when they are used but remain almost inactive when they are not used.

### 10.2 Metrics

We measure the following metrics:

- **Data Delay**, defined as the time between the generation of a new packet on a device and its collection by the master. In case a tag reads new data multiple times before being polled, the data delay is measured since the first undelivered data collection.
- **Packet Delivery Ratio**, calculated as the number of packets containing new samples that are received by the master over the total amount of new samples packets generated by all devices.
- **Fairness**, intended as the Jain's fairness index [13] calculated on the Packet Delivery Ratio.

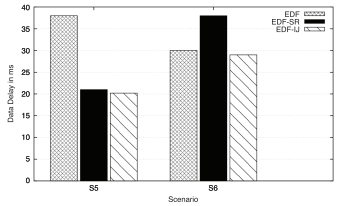


Fig. 10. Data delay comparison among EDF, EDF with interruption (EDF-SR), and EDF-IJ.

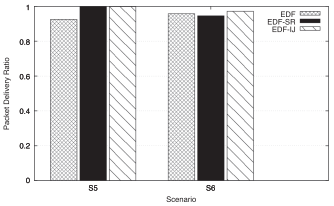


Fig. 11. Packet delivery ratio comparison among EDF, EDF with interruption (EDF-SR), and EDF-IJ.

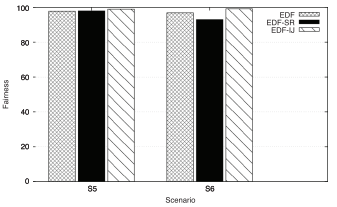


Fig. 12. Fairness comparison among EDF, EDF with interruption (EDF-SR), and EDF-IJ.

### 10.3 EDF vs. EDF-IJ: Simulation Results

*10.3.1 EDF-IJ Effectiveness.* We now present simulation results on the effectiveness of EDF-IJ. For comparison we consider two versions of EDF. The first one is based on a pure implementation of EDF, where in case of changes in the task set, the new schedule is issued only at the end of the current frame. The second one creates a new schedule at each task change, without considering any information regarding the interrupted schedule jobs. We indicate this second version of EDF as *EDF-SR*.

Figure 10 shows the results for data delay in a typical smart home scenario (about 40 smart devices), including a cameras continuously streaming data (scenario S5). EDF-SR and EDF-IJ are comparable, resulting around 65% faster than EDF. This is because EDF is not very reactive to changes, and it is slower to schedule a new frame with respect to the other two approaches. Increasing the number of cameras that continuously stream data (scenario S6) requires massive access to the communication channel, and introduces a significant number of overload scheduling events. In this case EDF-IJ is more efficient than EDF-SR. The superiority of EDF-IJ becomes more evident when looking at packet delivery ratio (see Figure 11) and fairness (see Figure 12): Not only does EDF-IJ guarantee high delivery ratio in both cases (above 97%), but it also results more fair (0.99) than the other policies (0.98 and 0.92 respectively for EDF and EDF-SR in Case S6). Thus, our first set of results clearly show the benefits of EDF-IJ.

*10.3.2 Resiliency to Overload Scheduling.* We now investigate the resiliency to overload scheduling of DL-resolution (it equally distributes loss of data), against common approaches. As explained in Section 8 there are three approaches to address overload scheduling: Best Effort, Admission control, and a Robust scheme. For comparison we consider the first two (Best Effort, Admission control), because the third one is not applicable to our context, as we do not have a priority list. The goal of the evaluation is to investigate the impact of DL-resolution of protocol performance in terms of data delay, data loss and fairness (we are interested in communicating with all the devices in an optimal but fair way) than the baselines.

Results are depicted in Figures 13, 14, and 15. For all metrics DL-resolution performs significantly better than the other two approaches, increasing its effectiveness in the more complex scenario. S6 represents a "saturation scenario," as the number of cameras continuously streaming data introduces a significant number of overload scheduling events. DL-resolution not only makes device communication faster (in S6 it takes on average only 28 ms, while the delay for the Admission Control is nearly 300 ms), but also reduces data loss (0.04 against 0.21 and 0.71 of Admission Control and Best Effort in S6). In any case, DL-resolution results in more fairness then the other two approaches (see Figure 15), as it achieves always 0.99 of fairness independently of the

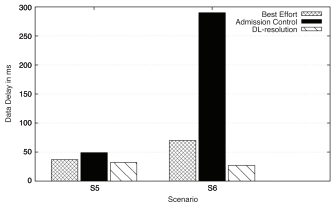


Fig. 13. Data delay of three mode-change approaches under EDF.

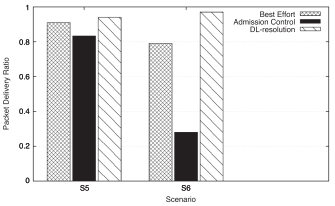


Fig. 14. Packet delivery ratio of three mode-change approaches under EDF.

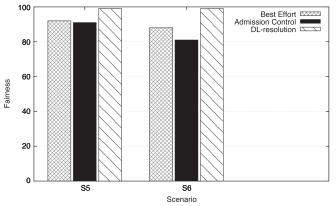


Fig. 15. Fairness of three mode-change approaches under EDF.

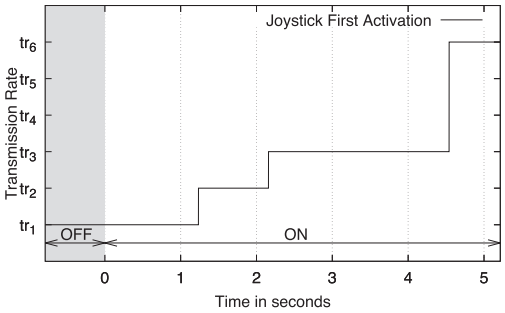


Fig. 16. First activation of a joystick.

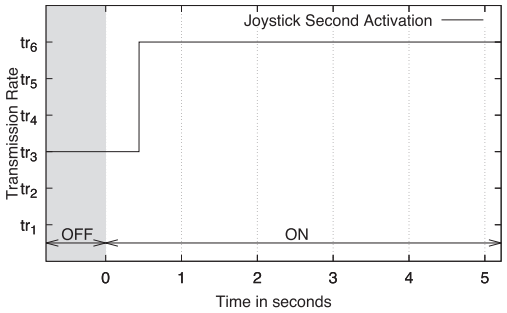


Fig. 17. Second activation of a joystick.

scenario. Thus this second set of results clearly show the benefits of using our DL-resolution to handle overload scheduling.

### 10.4 ReLEDF: Simulation Evaluation

We now turn to a thorough simulation evaluation of ReLEDF. We compare our protocol with APT-MAC [20], which is the closest to our work. To the best of our knowledge APT-MAC is the only solution so far proposed to address communication for battery-free devices. We also compare to the optimum and TDMA. The optimal solution, called OPT, is abstract as it always knows the best action to perform (i.e., which device to query next to avoid any data loss). TDMA is a baseline strategy that assigns slots in a round-robin TDMA schedule. For workloads we consider again typical smart-home scenarios, involving around 40 heterogeneous smart devices (temperature, presence, remote, joystick, camera) [20]. However to investigate scalability we consider also bigger networks (up to $N = 150$ devices). Furthermore, we consider a wide range of possible uses of devices (S1 to S8). We have performed actual experiments with real-devices. The simulations use parameter values such as frame rates of the cameras that match these real devices. The simulations are also validated by these experiments. Further, Table 2 shows for each of the seven simulations how many environmental, remotes, joysticks and cameras are simulated. We also chose to simulate 40 devices to represent a reasonable current generation smart home and 150 devices to demonstrate scaling for larger buildings. Complexity increases from S1 to case S8 as the number of burst devices increases.

*10.4.1 Learning Time.* The evaluation first looks at the effectiveness of our learning components, studying how fast ReLEDF learns and reacts to device behavior. Figure 16 shows the results on the time taken by ReLEDF to learn the required transmission rate for a joystick since it is switched on. For simplicity we indicate the relative time on the graph, meaning that the joystick activation occurs at relative time 0. When the joystick is activated for the first time, it takes the

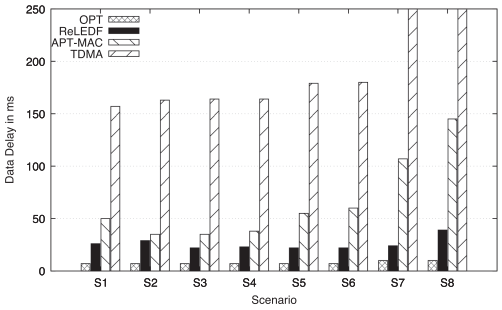


Fig. 18. Packet delay for 40 devices by varying types of devices.

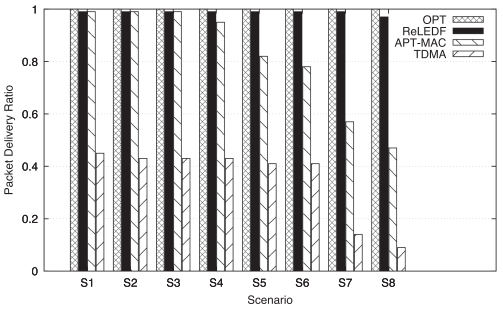


Fig. 19. Packet Delivery Ratio by varying types of devices.

learning components a bit more than 1 s to understand that the joystick is on, and around 4.5 s to learn its communication requirement (i.e., $tr_6$), that is a reasonable time at system start up. At the second activation instead the system achieves the proper requirement much faster, as it takes only 0.5 s to reach state $tr_6$, that is short enough for practical usage (see Figure 17). Thus our first results clearly show the benefits of using a RL-based approach, which quickly learns devices behavior.

*10.4.2 System Efficiency.* The second set of simulations investigates system efficiency. In terms of speed, Figure 18 shows data delay for a variable number of devices. Our ReLEDF outperforms APT-MAC taking only from 26.87 ms (case 1) to 39.8 ms (case 8), to deliver a new data sample, resulting respectively 2 and over 3.5 times faster than APT-MAC in cases 1 and 8. ReLEDF is very close to the OPT data delivery, performing only from 2.3 times slower in case 7 to 3.9 times in case 2. This performance trend is confirmed also when looking at the packet delivery ratio. As shown in Figure 19, ReLEDF delivers 99% of newly generated packets independently of the simulated case, being very close to the OPT that achieve always 100%, while APT-MAC loses 5% of new data in case 4 and up to 53% in case 8. APT-MAC clearly suffers the presence of cameras in the environment (cases 5 to 8). TDMA loses more than 50% of new sample data, successfully delivering only 9.92% of new data in case 8 and around 45% in the other cases. Increasing the number of devices (up to 150) does not have a significant impact on ReLEDF performance. Packet delay remains close to the optimum, reducing up to 6 times with respect to APT-MAC. In case of 150 devices, ReLEDF spends only 52 ms to deliver new data, against 316 ms for APT-MAC. Packet delivery ratio also remains high, resulting over 95% for up to 150 devices. In contrast, APT-MAC packet delivery ratio drops to only 47% in the case of 150 devices, while it is below to 10% for TDMA. Thus ReLEDF has almost optimal efficiency in assigning channel access to heterogeneous devices. We point out that these results are important in that they show that even ReLEDF's centralized solution is quite scalable.

## 10.5 ReLEDF: Experimental Evaluation

The goals of our experiments are twofold: (i) to investigate the real applicability of ReLEDF and (ii) compare performance of ReLEDF with that of baseline TDMA and state-of-the-art protocols like APT-MAC [20]. To this end we implemented ReLEDF, APT-MAC, TDMA, and OPT on prototype devices and run experiments. Before presenting experimental results we give a brief description of our testbed.

*10.5.1 Testbed.* Figure 20 shows our testbed, which is composed of (i) an USRP RFID reader, running RFID Gen2 Reader protocol developed by Buettner [5], and equipped with two 860- to 910-MHz antennas; and (ii) four smart devices—a temperature sensor, a presence sensor, a remote, a joystick—realized leveraging sensor augmented RFID Moo tags (see Figure 21) [40]. Moo

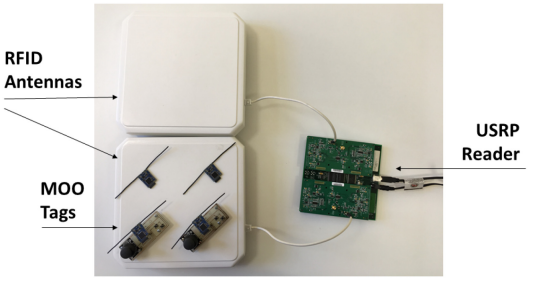


Fig. 20. Testbed: four Moo Tags, two 860-910 MHz antennas, a USRP Reader and two Flex 900 daughterboard.

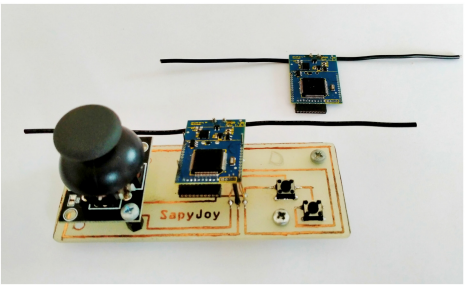


Fig. 21. A joystick and an environmental sensor.

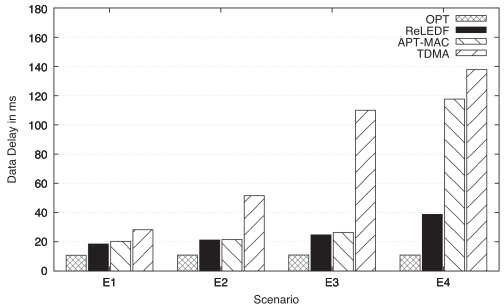


Fig. 22. Experimental Packet delay by varying MAC protocol.

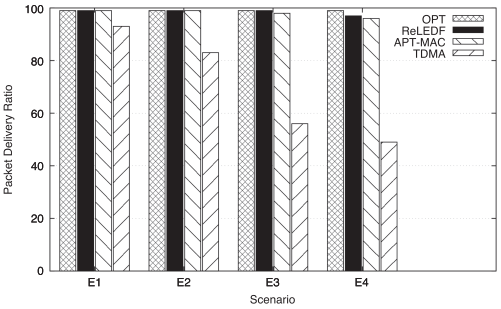


Fig. 23. Experimental Packet Delivery Ratio by varying MAC protocol.

platforms are equipped with an ultra low power MSP processor, a temperature sensor and a series of input peripheries, like buttons and joysticks, and are powered only by the energy harvested from the RFID reader. The communication protocols (ReLEDF, APT-MAC, TDMA, and OPT) have been implemented introducing new primitives to the EPC Gen 2 [2], which is the standard protocol for RFID identification. EPC issues frames of slots, whose length has been empirically estimated, observing the effect of communication errors or inferences (all packets are validated through a cyclic redundancy check). During experiments we noticed that the packet error rate changes depending on the slot length: with 6-ms slots, the system reaches a 18.9% of packet error rate, with 15 ms a 8.1% while with 25 ms gets lower than 2%. Although slots of reduced size have high packet error rates, the redundancy introduced by multiple queries makes ReLEDF work better with reduced size slots, reaching an optimum at 6 ms per slots. For this reason we fixed the slot length at 6 ms.

Each Moo tag emulates multiple smart devices so that we can perform experiments with a number of devices ranging from 4 to 20. Specifically, we use the traffic generators defined for the simulation environment to emulate the different devices. The Moo tags transmit data according to device models. In this way we can collect data on different environments, e.g., with 4, 8, 16, or 20 smart devices.

*10.5.2 Results.* We evaluated Data Delay and Packet Delivery Ratio while changing the number and the type of connected devices (see E1 to E4 scenarios in Table 2). Results are averaged over 100 repetitions.

Figure 22 shows that all protocols are very fast in delivering data, taking less than 40 ms in scenario E1, when there are few burst devices (only one remote and one joystick). Introducing a

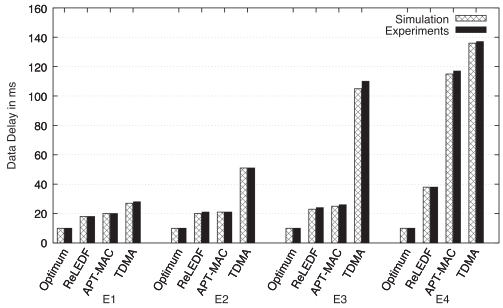


Fig. 24. Data Delay comparison among MAC protocols (Optimum, ReLEDF, APT-MAC, TDMA) by varying the number of devices both in terms of Simulation and Experiments.

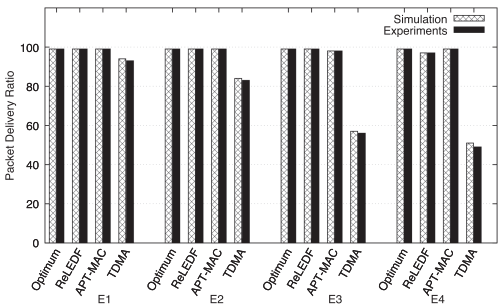


Fig. 25. Packet Delivery Ratio comparison among MAC protocols (Optimum, ReLEDF, APT-MAC, TDMA) by varying the number of devices both in terms of Simulation and Experiments.

camera and increasing the number of burst devices (1 joystick and 2 remotes in E2, and 2 joysticks and 3 remotes in E3) ReLEDF and APT-MAC keep low the data delay (below 25 ms), while TDMA results more than 4 times slower, taking on average 110 ms to deliver data in E3. The advantage of using ReLEDF becomes evident when the number of burst devices further increases (i.e., 1 camera, 3 joysticks and 4 remotes in scenario E4), as it takes ReLEDF on average only 39 ms to deliver data. Indeed, this time is only one third of the time taken by APT-MAC in the same scenario. The good performance of ReLEDF is confirmed by results on packet delivery ratio (see Figure 23), as it is able to deliver above 97% of new data, independently of the scenario. Thus ReLEDF is practical for smart homes.

*10.5.3 Validation.* We validated our simulations with experiments in the following way. We simulate the experimented scenarios, i.e., E1 to E4. Figure 24 shows the data delay for the four protocols. It is clear that the two evaluation techniques show comparable results, as simulation produces results that deviate by about only 4.7%, independently of the scenario. Please note that the experimental results of Figure 24 are the same as Figure 22. This outcome is confirmed by Figure 25, where performance difference between experiments and simulations in packet delivery ratio is below 4.8%.

*10.5.4 Energy Evaluation.* We now evaluate the energy impact of ReLEDF. As the whole protocol runs on the reader, the protocol does not have any effect on the energy consumption of battery-free devices. Hence we estimate the energy consumption of the reader, which is a wired-powered device. We measured the energy consumed by our reader, and quantified its consumption in a year, that is very low. The reader employed in the experimentation has a power consumption of around 1.3 A at 6 V, for around 8 W. Publicly available data specify that the maximum cost for energy in the USA is less than 20 cent per KW/h. Given these data, we can estimate the system consumption in around 70 KW/h per year, for a total of less than 20$ per year, that is very low. We also notice that, from an environment-friendly point of view, the reader represents the most convenient solution, as we don't have any waste of batteries, nor any pollution impact on the environment. Finally we notice that as stated in Reference [28] RFID technology should be deployed as part of a building's physical infrastructure, just like running water, lights, and heat. Hence, exploiting such readers, our proposed devices can work with the emitted energy, without requiring batteries, or other sources of energy. Hence, it is true that our system requires a continuously operating RFID reader but any smart environment involves the deployment of an RFID reader [22].

## 11 CONCLUSIONS

In this article, we developed ReLEDF, a new communication protocol that enables the coexistence of multiple battery-free smart devices in the same smart building or home. ReLEDF is able to learn and monitor the current and dynamic behavior (depending on changes in the environment that involves what devices are currently being used and what their current communication requirements are) of a variety of battery-free devices, and properly coordinate channel access, so as to satisfy heterogeneous device requirements. Our protocol exploits RL to build and update behavioral device models that represent device communication needs based on events detected in the environment, and a channel access method based on EDF scheduling, but modified to deal with schedule transitions. The mix of learning and scheduling is an innovative way to schedule the network, which guarantees high performance, and ensures that all devices meet deadlines without starvation. For this reason we propose a new mode-change scheduling approach (EDF-IJ), which reduces delays and avoids devices starvation in scheduling transmissions. Our simulation results show the benefits of using ReLEDF: It outperforms state-of-the-art solutions both in data collection efficiency and effectiveness, allowing to a high number of smart devices to communicate with an almost optimal packet delivery ratio. Finally, real experiments demonstrate the applicability of our protocol. The overall value of the solution is to support many IoT devices in a smart space without needing batteries.

## REFERENCES


[1] Björn Andersson and Cecilia Ekelin. 2007. Exact admission-control for integrated aperiodic and periodic tasks. *J. Comput. System Sci.* 73, 2 (2007), 225–241.

[2] Anonymous. 2015. EPC UHF Gen2 Air Interface Protocol - EPC/RFID. Retrieved from https://www.gs1.org/standards/epc-rfid/uhf-air-interface-protocol.

[3] Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D'angelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. 2015. Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. *J. ACM* 62, 2, Article 14 (2015), 1–33.

[4] Aaron Block and James Anderson. 2006. Accuracy versus migration overhead in real-time multiprocessor reweighting algorithms. In *Proceedings of the IEEE 18th International Conference on Parallel and Distributed Systems (ICPAD'06)*, Vol. 1.

[5] Michael Buettner and David Wetherall. 2010. A Gen 2 RFID monitor based on the USRP. *ACM SIGCOMM Comput. Commun. Rev.* 40, 3 (2010), 41–47.

[6] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. 2002. Elastic scheduling for flexible workload management. *IEEE Trans. Comput.* 51, 3 (2002), 289–302.

[7] Daniel Casini, Alessandro Biondi, and Giorgio Carlo Buttazzo. 2019. Handling transients of dynamic real-time workload under EDF scheduling. *IEEE Trans. Comput.* 68, 6 (2019), 820–835.

[8] S. Chen, M. Zhang, and B. Xiao. 2011. Efficient information collection protocols for sensor-augmented RFID networks. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM'11)*. 3101–3109.

[9] Farsens. 2020. Battery Free RFID Sensors. Retrieved from http://www.farsens.com/en/products/battery-free-rfid-sensors/.

[10] Nathan Fisher and Masud Ahmed. 2011. Tractable real-time schedulability analysis for mode changes under temporal isolation. In *Proceedings of the IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia'11)*. 130–139.

[11] Gerhard Fohler. 1993. Realizing changes of operational modes with a pre run-time scheduled hard real-time system. In *Responsive Computer Systems*. Springer, 287–300.

[12] Shyamnath Gollakota, Matthew S. Reynolds, Joshua R. Smith, and David J. Wetherall. 2013. The emergence of RF-powered computing. *IEEE Comput.* 47, 1 (2013), 32–39.

[13] Raj Jain, Arjan Durresi, and Gojko Babic. 1999. Throughput fairness index: An explanation. *ATM Forum Contribution* 99, 45 (1999).

[14] Linghe Kong, Liang He, Yu Gu, Min-You Wu, and Tian He. 2014. A parallel identification protocol for RFID systems. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM'14)*. 154–162.

[15] Thomas F. La Porta, Gaia Maselli, and Chiara Petrioli. 2010. Anticollision protocols for single-reader RFID systems: Temporal analysis and optimization. *IEEE Trans. Mobile Comput.* 10, 2 (2010), 267–279.

[16] Chung Laung Liu and James W. Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* 20, 1 (1973), 46–61.
[17] Xiulong Liu, Jiannong Cao, Keqiu Li, Jia Liu, and Xin Xie. 2018. Range queries for sensor-augmented RFID systems. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM'18)*. 1124–1132.
[18] Gaia Maselli, Matteo Pietrogiacomi, Mauro Piva, and John A Stankovic. 2019. Battery-free smart objects based on RFID backscattering. *IEEE IoT Mag.* 2, 3 (2019), 32–36.
[19] Gaia Maselli, Mauro Piva, Giorgia Ramponi, and Deepak Ganesan. 2016. JoyTag: A battery-less videogame controller exploiting RFID backscattering. In *Proceedings of the ACM Annual International Conference on Mobile Computing and Networking (MOBICOM'16)*. 515–516.
[20] Gaia Maselli, Mauro Piva, and John A. Stankovic. 2019. Adaptive communication for battery-free devices in smart homes. *IEEE IoT J.* 6, 4 (2019), 6977–6988.
[21] Alejandro Masrur, Dirk Muller, and Matthias Werner. 2015. Bi-level deadline scaling for admission control in mixed-criticality systems. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'15)*. 100–109.
[22] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. 2012. Internet of things: Vision, applications and research challenges. *Ad Hoc Netw.* 10, 7 (2012), 1497–1516.
[23] Saman Naderiparizi, Aaron N Parks, Zerina Kapetanovic, Benjamin Ransford, and Joshua R. Smith. 2015. WISPCam: A battery-free RFID camera. In *Proceedings of the IEEE International Conference on Radio Frequency Identification (RFID'15)*. 166–173.
[24] Vincent Nelis, Björn Andersson, José Marinho, and Stefan M. Petters. 2011. Global-edf scheduling of multimode real-time systems considering mode independent tasks. In *Proceedings of the 23rd IEEE Euromicro Conference on Real-Time Systems 2011*. 205–214.
[25] Carlos Pereira and Ana Aguiar. 2017. Modelling and optimisation of packet transmission scheduling in M2M communications. In *Proceedings of the IEEE International Conference on Communications Workshops (ICC Workshops'17) 2017*. 576–582.
[26] Yan Qiao, Shigang Chen, Tao Li, and Shiping Chen. 2015. Tag-ordering polling protocols in RFID systems. *IEEE/ACM Trans. Netw.* 24, 3 (2015), 1548–1561.
[27] Jorge Real and Alfons Crespo. 2004. Mode change protocols for real-time systems: A survey and a new proposal. *Real-time Syst.* 26, 2 (2004), 161–197.
[28] M. Roberti. Oct. 2018. RFID needs to be part of the building. *RFID J.* https://www.rfidjournal.com/rfid-needs-to-be-part-of-the-building.
[29] Lui Sha, Ragunathan Rajkumar, John Lehoczky, and Krithi Ramamritham. 1989. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Syst.* 1, 3 (1989), 243–264.
[30] Dong-Her Shih, Po-Ling Sun, David C. Yen, and Shi-Ming Huang. 2006. Taxonomy and survey of RFID anti-collision protocols. *Comput. Commun.* 29, 11 (2006), 2150–2166.
[31] Marco Spuri. 1995. Earliest deadline scheduling in real-time systems. *Ph.D. Dissertation, Scuola Superiore S. Anna, Pisa* 34.
[32] John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo. 2012. *Deadline Scheduling for Real-time Systems: EDF and Related Algorithms*. Vol. 460. Springer Science & Business Media.
[33] Nikolay Stoimenov, Simon Perathoner, and Lothar Thiele. 2009. Reliable mode changes in real-time systems with fixed priority or EDF scheduling. In *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 99–104.
[34] Nikolay Stoimenov, Lothar Thiele, Luca Santinelli, and Giorgio Buttazzo. 2010. Resource adaptations with servers for hard real-time systems. In *Proceedings of the 10th ACM International Conference on Embedded Software 2010*. 269–278.
[35] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. MIT Press.
[36] Vamsi Talla, Bryce Kellogg, Shyamnath Gollakota, and Joshua R. Smith. 2017. Battery-free cellphone. *Proceedings of the the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 2 (2017), 25.
[37] Ken W. Tindell, Alan Burns, and Andy J. Wellings. 1992. Mode changes in priority preemptively scheduled systems. In *Proceedings of the IEEE Real-Time Systems Symposium 1992*.
[38] Jue Wang, Haitham Hassanieh, Dina Katabi, and Piotr Indyk. 2012. Efficient and reliable low-power backscatter networks. In *Proceedings of the ACM Association for Computing Machinery's Special Interest Group on Data Communications (SIGCOMM'12)*. 61–72.
[39] Daniel J. Yeager, Alanson P. Sample, and Joshua R. Smith. 2017. Wisp: A passively powered uhf rfid tag with sensing and computation. In *RFID Handbook*. CRC Press, 261–276.
[40] Hong Zhang, Jeremy Gummeson, Benjamin Ransford, and Kevin Fu. 2011. Moo: A Batteryless Computational RFID and Sensing Platform. *University of Massachusetts Computer Science Technical Report UM-CS-2011-020* (2011).

Received April 2020; revised December 2020; accepted February 2021