Scheduling Offset-Free Systems Under FIFO Priority Protocol

Matheus Ladeira, Emmanuel Grolleau, Fabien Bonneval, Gautier Hattenberger, Yassine Ouhammou, Yuri Hérouard

Scheduling Offset-Free Systems Under FIFO Priority Protocol

Matheus Ladeira
ISAE ENSMA, Chasseneuil, France
University of Poitiers, France

Emmanuel Grolleau
ISAE-ENSMA, Chasseneuil, France
University of Poitiers, France

Fabien Bonneval
Ecole Nationale de l'Aviation Civile, Toulouse, France

Gautier Hattenberger
Ecole Nationale de l'Aviation Civile, Toulouse, France

Yassine Ouhammou
ISAE-ENSMA, Chasseneuil, France
University of Poitiers, France

Yuri Hérouard
ISAE-ENSMA, Chasseneuil, France

Abstract

On UAVs, telemetry messages are often sent following a FIFO schedule, and some messages, depending on the FIFO queue state may suffer long delays, and can even be lost if the FIFO queue is full. Considering the high complexity of the problem of assigning offsets to periodic tasks, we propose a new heuristic, called GCD+, that we compare to the methods of the state of the art, showing that GCD+ significantly outperforms them on synthetic tasks sets. Then we use a real UAV use case, based on Paparazzi autopilot, to show that GCD+ behaves well. The proposed algorithm is meant to be the new Paparazzi's automatic offset assignment method for messages.

2012 ACM Subject Classification Computer systems organization → Real-time system architecture; Computer systems organization → Embedded software

Keywords and phrases Scheduling, non-preemptible, heuristics, FIFO, autopilot

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2022.11

Supplementary Material Source code for an implementation of GCD+ in Python was submitted under LGPL license to artefact evaluation, and can be found in: `https://github.com/lias-laboratory/gcdplus`
Software (ECRTS 2022 Artifact Evaluation approved artifact):
`https://doi.org/10.4230/DARTS.8.1.4`

Funding This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 826610. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Austria, Belgium, Czech Republic, France, Italy, Latvia, Netherlands.

1 Introduction

When conceiving an Unmanned Aerial System (UAS), i.e., a flying drone and every supporting device in the drone's network, the communication between the drone and the Ground Control Station (GCS) or the remote pilot may be critical, since it may determine if the drone is

certifiable or not for a specific mission [15]. Without a reliable communication link and the guarantee that the GCS or the remote pilot can take control of the drone, safety and security of every entity in the surroundings must be insured only by its autonomous functionalities. This requires the development of very complex and, therefore, expensive embedded skills.

Hence, guaranteeing the nominal behaviour of the communication system is an important step in drone development and configuration. Nevertheless, not only a larger communication band can improve the exchange of messages between the different parts of the UAS: the choice of the moments in which these messages are sent, by orchestrating their periods and the moments of the first message transmissions, can play an important role in order to avoid large waiting queues and, consequently, message delays or even losses. Some message delays can make the GCS conclude falsely that the communication is lost and enter in fail-safe mode.

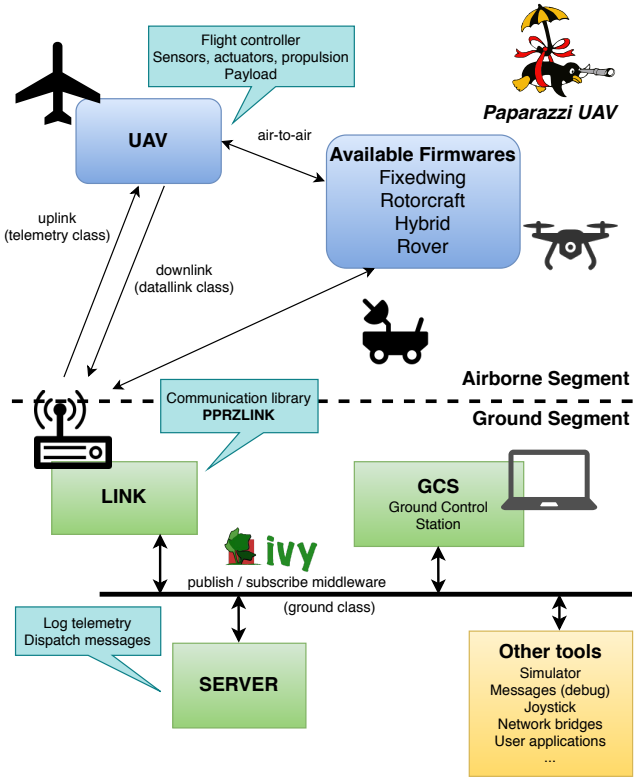


Figure 1 Paparazzi system architecture.

Amongst current solutions for the problem, we can highlight the one used in Paparazzi [3, 11, 16], the open source autopilot developed by ENAC (École Nationale d'Aviation Civile, France) and used in many research-related domains. It is conceived using modular functions for the control algorithm, that are called in a single execution thread, one after the other according to a defined pattern of execution. In parallel, a telemetry thread is responsible for sending telemetry data through a wireless channel to a ground station, as depicted in Figure 1. Each type of message has a defined range of possible sizes and a specific period in which it will be sent. The messages are sent following a First-In-First-Out (FIFO) scheduling policy.

In their current approach, message periods are chosen and adapted so that they yield a very high hyperperiod, i.e., the time it takes for the pattern of message transmission to start repeating (mathematically, it is the Lowest Common Multiple of all message periods). If

every message is called to be sent in the same moment in time, this will happen only once in every hyperperiod. Therefore, by adjusting periods such that their LCM is high enough, the system will have only one simultaneous activation of all the messages in a very long period of time – days, years, millennia, or even further. However, this simultaneous call can still happen, while there are other methods that may prevent it completely.

The problem of message transmission in a single communication link is analogous to the problem of scheduling non-preemptive tasks on a single processor. More specifically, this is equivalent to the problem of finding offsets in an offset-free system of periodic tasks under FIFO scheduling (thus tasks are non-preemptive), which has been studied in the literature.

In this article, we analyse the problem of message and task scheduling, proposing GCD+, a new method to seek for better offsets and to avoid deadline misses. Section 2 explores in depth the motivating case for this development. Section 3 presents a mathematical model for the problem. Section 4 brings to light other heuristics that deal with the same problem. Section 5 describes our contribution: the new heuristic method. Section 6 compares our heuristic to three heuristics found in the literature. Section 7 applies the new method to a real case involving the Paparazzi autopilot, and finally, Section 8 concludes the paper.

2 Motivating Example

Currently, in order to avoid critical events (several simultaneous message calls), the adopted strategy in Paparazzi consists of two methods applied in parallel. The first one is changing message periods in small amounts (from about 1% to 10%), so that the critical case for any two messages only happens once in a very long interval (the hyperperiod, calculated as the LCM of the periods). For example, instead of using the periods 3 and 5 seconds, which will compose an execution pattern that will repeat every 15 seconds (i.e. the hyperperiod is 15 seconds long), one can choose to use periods of 3.1 and 5.1 seconds, which yield a hyperperiod of 158.1 seconds. This technique can help reduce the interference specially considering sets of several messages, which will have a very large hyperperiod, meaning that concurrent calls become less probable. However, it does not guarantee that critical cases will not happen, and it might lead to unexpected long delays that are very hard to reproduce.

The second method, complementary to the first, relies on the use of offsets, so the critical instant does not happen at $t = 0$. The first message to be evaluated is assigned an offset of 0; the second one, an offset equivalent to 10% of its period; the third one, 20% of its period; and so on (and, when the counter reaches 100%, it resets to 0 so that every offset is smaller than the respective period).

Using offsets is a well-known strategy to increase the schedulability of task systems: finding the right combination of offsets for each set of tasks in order to avoid overlapping [7]. The problem consists in choosing offsets in an offset-free system such as to avoid load peaks. Hence, instead of being called at every instant $t = n \cdot T_i$ (where $n \in \mathbb{N}$ and T_i is the period in which τ_i is executed), a task would be called at every instant $t = O_i + n \cdot T_i$ (where O_i is an initial offset given to τ_i). By coordinating the individual offsets of each task according to their periods and to the other periods in the system, it might be possible to completely prevent critical cases or, at least, reduce the maximum possible interference between messages – and in a mathematically predictable way.

The analogy between message and task scheduling holds due to the fact that the transmission of a message happens at a defined rate, such as the processing of individual instructions in a CPU, and the message sizes are analogous to the execution times of the tasks. These, however, must be considered non-preemptible, since once a message starts being sent, the

process cannot be interrupted by a new arriving message. Also, due to implementation constraints (namely a ring buffer used to store output messages) the first message to arrive in the transmission queue will be the first to be sent, and therefore they are under a FIFO scheduling policy.

Paparazzi uses a configuration file to describe its telemetry, where each message is characterized by a period and an offset. Therefore, increasing schedulability by changing the offset does not require changing a single line of the C code of the autopilot, saving testing and validation efforts.

In our case, the goal is not only to meet deadlines, but also to minimize the response time of the messages. This goal is related to the freshness of data sent over the datalink. Indeed, the longer a message is kept in a queue before being sent, the less representative of the state of the system it is.

In the following sections, we will analyse the behaviour of task systems (but the analogy between processing non-preemptible tasks and sending telemetry messages holds for every case).

3 Problem Statement

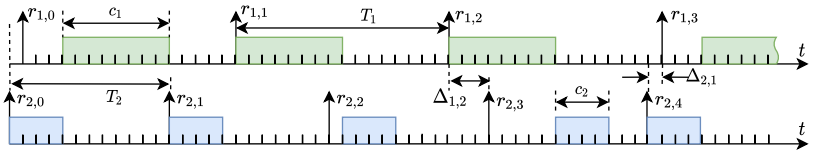


Figure 2 Model of execution of a set of two periodic tasks under FIFO with different offsets. The up arrows represent task releases.

An offset-free task system [7] Φ of n periodic tasks executing in a single processor under FIFO scheduling (and, therefore, non-preemptive), each task τ'_i (such that $i \in [1, n] \subset \mathbb{N}$) being given by a tuple composed of two positive integers – its period T_i and its Worst Case Execution Time (WCET) c_i –, is given by Equation 1.

$$\Phi \triangleq \{\tau'_i \mid \tau'_i = (T_i, c_i)\ \forall i \in [1, n]\} \tag{1}$$

We are interested in finding a vector of n integers O_i such that the concrete task system Θ, defined by Equation 2, is schedulable and, in order to guarantee data freshness, where the tasks have the smallest starting time. Since the scheduler is non-preemptive, an earlier starting time means a lower response time.

$$\Theta \triangleq \{\tau_i \mid \tau_i = (\tau'_i, O_i)\ \forall i \in [1, n]\} \tag{2}$$

In order to do that, like in [7, 18, 1], we will make use of Theorem 1. Nevertheless until now, it has been used to guarantee space between releases of pairs of tasks, but in our GCD+ heuristic, we use it to guarantee space between more tasks than two, hence the name of our heuristic.

▶ **Theorem 1** ([18]). *Given two concrete tasks τ_i and τ_j, the minimum distance between the release of a job of τ_i and the next job of τ_j is given by:*

$$\Delta_{i,j} = (O_j - O_i) \bmod \mathrm{GCD}(T_i, T_j) \tag{3}$$

Using the example depicted in Figure 2 and considering that τ_1 has an offset $O_1 = 1$, a period $T_1 = 16$ and an execution time $c_1 = 8$, and that τ_2 has an offset $O_2 = 0$, a period $T_2 = 12$ and an execution time $c_2 = 4$, the minimum distance between the release of a job of τ_1 and the next release of τ_2 is:

$$\Delta_{1,2} = (O_2 - O_1) \bmod \text{GCD}(T_1, T_2) = (0 - 1) \bmod 4 = 3$$

Similarly, the minimum distance between the release of a job of τ_2 and the next release of a job of τ_1 is:

$$\Delta_{2,1} = (O_1 - O_2) \bmod \text{GCD}(T_1, T_2) = (1 - 0) \bmod 4 = 1$$

By taking Equation 3 and comparing its result to c_i, if $c_i > \Delta_{i,j}$ and jobs use their WCET, it is guaranteed that there will be a delay in the execution of a job of τ_j, and this delay will be of at least $c_i - \Delta_{i,j}$. Therefore, we can define a lower bound $I_{i,j}$ for the largest interference caused by the execution of a job of τ_i on the execution of the next job of τ_j as being the amount of time the execution of τ_i overlaps into the minimum distance from its request to the subsequent request of τ_j. So, we have Equation 4.

$$I_{i,j} \triangleq \begin{cases} 0 & \text{if } c_i \leq \Delta_{i,j}, \text{ or } i = j \\ c_i - \Delta_{i,j} & \text{otherwise} \end{cases} \tag{4}$$

Note that we use a FIFO scheduling policy, and that if a job is released before another one, then the former will be executed before the latter. The interference that a job under analysis can suffer when it is released is therefore the remaining execution time of the job in execution (if any) plus the sum of the worst-execution times of the jobs which are ready at or before the release of the job under analysis. The value of $I_{i,j}$, if positive, shows the lowest bound for the biggest amount of delay that will be caused in the execution of a job of τ_j due to the execution of a job of τ_i. The maximum delay can still be larger than $I_{i,j}$ in case any backlog occurs, as will be seen in a following example.

Using the example in Figure 2, the lower bound for the maximum delay that τ_1 causes in τ_2 is:

$$I_{1,2} = c_1 - \Delta_{1,2} = 8 - 3 = 5$$

And the minimal value for the maximum delay that τ_2 causes in τ_1 is:

$$I_{2,1} = c_2 - \Delta_{2,1} = 4 - 1 = 3$$

It can be seen from Figure 2 that the maximum delays suffered by tasks τ_1 and τ_2 are effectively correspondent to $I_{2,1}$ and $I_{1,2}$, respectively. However, considering the same system but with $c_2 = 6$ instead of 4, $I_{2,1} = 5$, but instead the simulation shows that a maximum delay of 6 happens. This can be explained by the presence of a backlog (an accumulation of preceding delays).

This lower bound can be far from the real maximum delay in the case for more than two tasks in a system, since delays can more easily accumulate. Yet, reducing the lower bound for the maximum interference in task pairs is shown to improve the system's schedulability [7, 8].

▶ **Proposition 2.** *If $I_{i,j} = 0 \; \forall i, j$, then there is no delay in the execution of the system, i.e., every task job $\tau_{i,k}$ ends its execution time in the instant $r_{i,k} + c_i \; \forall i, k$.*

Proof. We prove it by recurrence. If $I_{i,j} = 0 \; \forall i, j$, the first task job to be requested will certainly finish its execution before the next job request to happen, since no task executes before it. Hence, the next job will not have any initial delays, and since $I_{i,j} = 0 \; \forall i, j$, it will also finish its execution before the next request, and so on. ◀

The goal of this paper is to present a method to look for the state of null interference discussed in Proposition 2 if this state exists.

4 State of the Art

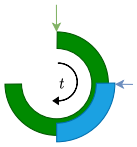
Some methods have been proposed to assign offsets to systems independently of their scheduler (FIFO, Earliest Deadline First, etc.), such as the heuristic proposed in this paper. Given the complexity of the problem, well described by [7], the use of heuristics is a common approach to solving it. In [7], the authors have proposed an algorithm (hereafter called Goossens's Heuristics) to order every possible task pair in a set according to a decreasing order of the GCD of their periods, so each pair of tasks (starting from the highest GCD) is given a pair of offsets which are apart by half of their GCD. The goal is to try to separate as much as possible the task calls. The complexity of the algorithm is $\mathcal{O}(n^2 \cdot (\log T_{max} + \log n^2))$. We can think of a slight modification to this method by separating the "half instant" (call instant plus half of the execution time) of every task, instead of their calls. This modification is hereafter called Modified Goossens's Heuristics.

Later, a modification of the previous heuristics is proposed in [8], regarding the priority that determines the order in which task pairs are evaluated, but keeping the principle of separating as much as possible the task calls. Four new priority assignments are evaluated, based on expressions containing the pairwise GCDs. The algorithm complexity is, therefore, considered to be the same.

Also, a method is proposed in [9] (hereafter called CAN Message Heuristics) to look for the longest least-charged time interval in the interval $[0, T_{max})$, and then put the task call in the middle of that interval (starting from the task with the lowest period. Its complexity is said to be $\mathcal{O}(n \cdot T_{max})$, therefore it is pseudo-polynomial.

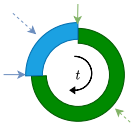
Paparazzi currently uses a method where each message has an offset correspondent to a multiple of 10% of its period (modulo 100%) [17]. The multiple is defined at the moment the message is added to the set, such that: the first message has an offset correspondent to 0% of its period; the second, 10%; the third, 20%; and so on. In mathematical notation: $O_i = ((i-1) \bmod 10) \cdot 0.1 \cdot T_i$. Its complexity is, hence, $\mathcal{O}(n)$.

Moreover, some methods were proposed specifically for FIFO schedulers. In [1], the authors explore several properties for FIFO schedulers and derive a sufficient (but not necessary) test to assess the schedulability of a task set (given the tasks' periods, execution times and offsets). However, they do not propose a method for calculating the tasks' offsets other than assigning random values, while in [12] the authors do present such a method. Their proposal consists in using one or several offsets per task in order to reproduce, under FIFO, the schedule constructed by another scheduler (Earliest Deadline First, for example). Nevertheless, implementing this approach in the case of the Paparazzi autopilot would require the autopilot's code to be modified, retested and revalidated, while a single offset per task requires only a specific configuration (already expected by the code).



(a) $O_1 = 0$ and $O_2 = 1$.

(b) Solution presented by Goossens's heuristics: $O_1 = 0$ and $O_2 = 2$.



(c) Solution presented by modified Goossens's heuristics: $O_1 = 0$ and $O_2 = 3$.

Figure 3 Representation of the execution of a system around a modular circle of size 4, with $T_1 = 16$, $c_1 = 3$, $T_2 = 12$ and $c_2 = 1$.

5 Contribution

This paper's contribution relies on the extension of a simple yet very useful technique to analyse pairs of tasks, allowing it to be used on whole sets of tasks. This technique is based on the modular arithmetic around the GCD of their periods, which is based on Theorem 1. Therefore, it needs a "sufficiently large" GCD to function properly, i.e., the GCD of all the periods cannot be smaller than any execution time. Otherwise, this method will give poor results.

If we analyse the modular circle around the GCD(T_i, T_j), hereafter also referred to as $\text{GCD}_{i,j}$, and we represent the execution of those tasks from their release to their end, we can rapidly see if there will be any overlap in execution, such as in Figure 3a. The overlap in the GCD circle means that there will be an instant during the execution where there will be a task release, but another one will already be in execution. Similarly, if there is no overlap, then there will not be any delay whatsoever if there is no backlog.

The method presented here intends to reduce execution overlaps, but considering the system as a whole instead of pairs of tasks. For this, we take advantage of a property of semi-harmonic task sets, which means their task periods are all multiple of a given value Ω. This can be the case for several task systems, specially when we consider that there are several techniques to adapt task periods in given ranges so that they can approach this semi-harmony [2, 20, 4, 19, 13].

To better understand the method, we can start with an example. Let us suppose we want to assign an offset for a task τ_3 of period $T_3 = 8$ and execution time of $c_3 = 2$ in the system represented by Figure 3c. We might have the impression that adding any task to the system would cause an overlap in execution times. However, even if the circle looks fully occupied, the system still has some free spaces, as seen in Figure 4.

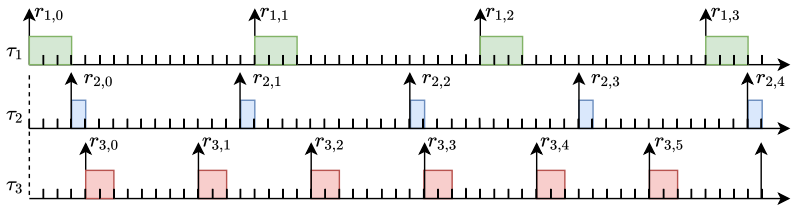


Figure 4 Execution of τ_1 (top, green), τ_2 (middle, blue) and τ_3 (bottom, red).

In fact, when comparing the GCD circle with the real execution of the system, τ_1 is released only once every $\frac{T_1}{\text{GCD}_{1,2}} = 4$ GCDs. On the right side of Figure 5, it is represented in green in GCDs 0, 4 and 8. This means that there is still room to execute, without interference

with the two considered tasks, within the 3 GCDs out of 4 which are not used by τ_1, using τ_1 reserved slot on the GCD circle. This reserved slot is called a section, and is represented on the left part of Figure 5 as a yellow sector in the central pie chart. Sections are partitions of the GCD cycle that will be allocated to tasks which have to be collocated on the same GCD cycle. For example, since τ_1 and τ_2 have respective periods of 16 and 12, there will be one time interval of size 4 such that τ_1 and τ_2 are both released each LCM of the periods. Now, if we consider τ_3, of period 8, since $\text{GCD}_{1,3}$ is twice as large as $\text{GCD}_{1,2,3}$, τ_3 can be added to the system using the section allocated to τ_1 because it can always alternate with τ_1. Since this section is a partition over the GCD cycle, it will not interfere with τ_2 either. This is represented in Figure 5, where task τ_3 is assigned to the second GCD circle, at the beginning of the yellow section, and will then use this section on circles numbered $1 + 2k$ for any natural integer k.

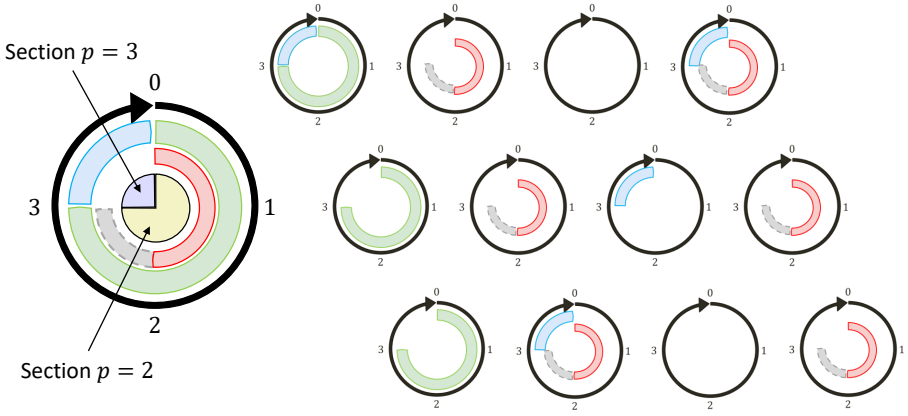


Figure 5 Representation of the execution of τ_1 (green), τ_2 (blue), τ_3 (red) and τ_4 (dashed grey) over the overall GCD circle (left) and the cycles 0 to 11 composing the hyperperiod (right).

Several other tasks could also be added without interfering with the rest of the system, in the unused spaces seen in Figure 5. For example, if we consider τ_2, the ratio $\frac{T_2}{\text{GCD}_{1,2}} = 3$ allows execution within the 2 GCDs over 3 which are not used by τ_2. Therefore, we could add, without creating interference, one or several tasks using the section used by τ_2 (represented in purple on Figure 5) two GCDs over three in the same way as τ_3 used τ_1's unused GCD section. In both cases, we can observe that this addition of tasks without interference is always possible if the period of these additional tasks is a multiple of $\text{GCD}_{1,2}$. As a result, by construction, if $n - 2$ tasks were added to the initial system of two tasks using this method, $\text{GCD}_{1,\dots,n} = \text{GCD}_{1,2}$.

Note that this condition is sufficient but not necessary to add a new task to a system without interference. For example, if we consider the system $\Phi = \{\tau_1 = (T_1 = 6, c_1 = 1), \tau_2 = (T_2 = 10, c_2 = 1)\}$, an additional task $\tau_3 = (T_3 = 15, c_3 = 1)$ can be added without interference while T_3 is not a multiple of $\text{GCD}_{1,2}$. We do not seek a necessary and sufficient condition of non overlap, because we know that finding non overlapping intervals is a hard problem. Indeed, the simultaneous in-congruence problem, which is at the root of this problem, is NP-hard in the strong sense [5].

In the problem tackled by this paper, we do not have the freedom to change periods, but we can retain the main idea by working with the GCD of all the tasks to try to find offsets minimizing the interference. In an attempt to consider the system holistically instead

of a union of independent pairs of tasks, the proposed heuristics of this paper considers the GCD of all the periods. In our context, the time unit being the duration of a bit on the network, and the periods of the telemetry messages often being multiples of tens, hundreds or thousands of milliseconds, we expect the global GCD to be large in general in real applications.

The global GCD, hereafter called Ω, is defined in Equation 5.

$$\Omega = \mathrm{GCD}_{1,2,\ldots,n} \tag{5}$$

If two tasks have a GCD between their periods which is greater than Ω, it is possible to arrange them as it was done for τ_3 in the preceding example. On the other hand, if their GCD is exactly Ω, then they must be put in a modular circle around Ω to be analysed, because over the hyperperiod, there will be a time window of size Ω where both tasks will be released.

Our method seeks that sort of arrangement in any given system by checking how many distinct sections (reserved slots, such as the one shared by τ_1 and τ_3 in the example) will need to co-exist in Ω, so that groups of tasks can be scheduled according to their respective sections and that each section will have its own reserved interval of time, without ever overlapping with other sections. The amount of sections will be given after analysing the quotient of each task period by Ω: if the GCD between two quotients is greater than one, it means that their respective tasks can be put in the same section in the mod Ω circle without creating an overlap in execution times. This quotient will be called a task's subperiod as defined in Equation 6.

$$T_{S_i} = \frac{T_i}{\Omega} \tag{6}$$

From the example in Figure 4, $T_{S_1} = 4$, $T_{S_2} = 3$ and $T_{S_3} = 2$. Note that T_{S_1} and T_{S_3} are multiple of 2 (which is why we characterize their section as prime $p = 2$ in Figure 5), and that T_{S_2} is co-prime with the other two. Since $T_{S_2} = 3$, the section assigned to τ_2 is characterized by a prime $p = 3$.

If we divide the whole execution of the system into cycles of Ω units of time such as in the right side of Figure 5, each task can be released periodically every T_{S_i} cycles. So, τ_1 can be released every four cycles, and τ_3, every two cycles. The algorithm shall be able to schedule τ_3 so that it avoids being released in the same cycle as τ_1. This can be done by analysing the possible cycles it can be assigned to: since its subperiod is only two, it can only be assigned to cycle zero or cycle one (assigning it to cycle two is the same as assigning it to cycle zero due to its periodicity). For cycle zero, the section for multiples of two is already occupied by τ_1, but cycle one is free. Then, it can be assigned to start at the beginning of the section of prime two in cycle one.

To sum up, we have three values to choose for each task:

1. The cycle number between 0 and $T_{S_i} - 1$;
2. The section within the cycle, acting as a partition, whose prime has to be a divisor of the subperiod;
3. An offset inside the section, in case two tasks share the same cycle and section (e.g., if we want to add τ_4 with $T_4 = 8$ and $c_4 = 1$ in cycle one, in the section of prime $p = 2$, right after every execution of τ_3, represented in dashed gray on Figure 5).

For each assignment above there is a partial offset, respectively O_C, O_S and O_I, such that the final offset will be given by Equation 7. Tasks assigned to start at the same cycle will have the same O_C, while those that use the same section (multiples of the same prime) will share the same O_S, and if they share both O_C and O_S they will have to have distinct O_I.

$$O_i = \Omega\ O_{C_i} + O_{S_i} + O_{I_i} \tag{7}$$

Using our example in Figure 5, τ_1 and τ_2 have $O_C = 0$ since they are executed in the first cycle, while τ_3 has $O_C = 1$. As τ_1 and τ_3 share the same section, they have the same $O_S = 0$, while τ_2 has $O_S = 3$, which is the moment in the GCD circle when their section begins. And, since every task starts in the same time as their respective section, every $O_I = 0$. A new task τ_4 with $T_4 = 8$ and $c_4 = 1$, set to be executed right after τ_3 finishes its execution (as seen in Figure 5), would share the same values of O_C and O_S with τ_3, but would have $O_I = 2$.

Note that, if we want to add a task τ_5 such that $T_5 = 20$ and, therefore, $T_{S_5} = 5$, independently of the execution time of the task, GCD+ would have to create a new section: those for subperiods multiple of 5. The section will be put to begin its execution right after the one for multiples of 3 (i.e., for τ_5, $O_C = 4$). It will necessarily result in an interference in every task in the section for multiples of 2, as GCD+ was not able to find a better solution.

Algorithm 1 represents the algorithm to determine the values of each partial offset. First, it calculates Ω, the overall GCD of all the periods. Based on this value, it assigns tasks one by one to the section that will increase the least in size (i.e., for every cycle, the longest it takes for its tasks to finish their execution). By doing this, it can calculate the partial offsets O_{C_i} and O_{I_i} so that the task can start as soon as possible in the section without overlapping with any other. When every task is assigned, it can then calculate each section size and finally apply an offset to each section, so one is released only after the previous has finished its execution. Its complexity is $\mathcal{O}(n \cdot (\log T_{max} + \sqrt{T_{max}} + \frac{\log T_{max}}{\log \log T_{max}} \cdot (\log T_{max} + T_{max})))$ using the Euclid's algorithm for finding GCDs in $\mathcal{O}(\log T_{max})$, a sieve to factor numbers in approximately $\mathcal{O}(\sqrt{T_{max}})$ and considering the number of distinct prime factors of a number to be at most $\frac{\log T_{max}}{\log \log T_{max}}$. Note that the last T_{max} is only a superior bound to T_{S_i}, and the algorithm generally works very far from this bound.

Algorithm 1 GCD+.

Require: $\Phi = \{(T_i, c_i) \mid i, T_i, c_i \in \mathbb{N} \wedge i \in [1, n]\}$
Ensure: $O = \{O_i \mid i, O_i \in \mathbb{N} \wedge i \in [1, n] \wedge O_i < T_i\}$
1: $\Omega \leftarrow \text{GCD}(\{\forall\ T_i\})$
2: **for all** $\tau_i \in \Phi$ **do**
3: **if** $T_i = \Omega$ **then** assign$(\tau_i, 1)$
4: **else**
5: Find prime factors of T_{S_i} (Equation 6)
6: Choose prime p such that assigning τ_i to it increases its section size by the least possible amount
7: assign(τ_i, p), defining the cycle O_C and internal offset O_I
8: **end if**
9: **end for**
10: calculate every O_S (section sizes)
11: **for all** $\tau_i \in \Phi$ **do** calculate O_i according to Equation 7
12: **end for**

6 Performance comparison

6.1 Experimental setup and chosen metrics

Random task sets are generated according to the methodology proposed in [10], to generate an unbiased distribution of utilization factors.

In order to generate representative task periods, there are two available methods in the literature. The first one, used in [1] for generating semi-harmonic task sets, chooses a random integer between defined lower and upper bounds, and multiplies the random number to a defined value, of which every period will be a multiple of. The second one [6], more general than the first, generates task periods from a finite set of possibilities according to a given factor distribution. The latter was chosen due to it being more representative of real-world scenarios.

The method for generating task periods works as follows: a set of prime numbers $\{p_i\}$ is given to the algorithm as an input. Each prime p_i related to a unique vector $\{x_j \in \mathbb{N}\}$ of size n_i. The vector represents the probability distribution that the corresponding exponent will be chosen to compose the final period. In other words, the probability that the factor $p_i^{(j-1)}$ will compose the generated period is $x_j / \sum_k x_k$). The possible values for the periods are limited to $\prod_i p_i^{(n_i-1)}$, and so is the hyperperiod of the system.

The generating set of vectors for each task period is obtained from the factors present in the periods of a real set of messages from Paparazzi[1]. This file, in addition to being used in the real telemetry system of rotorcraft equipped with Paparazzi, contains periods that were purposefully tuned to increase the hyperperiod (using 11.1 seconds instead of 10, for example). This causes the overall GCD to be reduced, approaching then a worst-case scenario in our domain of application. Yet, the overall GCD has proven to be sufficiently large.

The generated periods and generated utilization factors are multiplied to obtain the execution times. These are then rounded to the nearest integer. Task sets that have any execution time rounded to 0 are discarded, since they will not have the effective number of tasks that were demanded. Also, only sets for which the GCD of all the periods is greater or equal to the largest execution time will be used. This case is often found in real applications such as Paparazzi, and is needed for our heuristics to work properly.

Then, the algorithms for finding the offsets are used in each task set. After each offset assignment algorithm is run, the respective concrete task sets are simulated over an interval equivalent to two hyperperiods plus the greatest offset and, for every task in every set, its maximum delay η_i is registered (the largest period of time between a task release and the beginning of its execution). This value represents the largest interference a task suffers. The delays can then be put directly in a box plot to be analysed in their brute form, where each task in each set will have its data point indicating the value of its maximum delay, and these data points will be condensed in boxes where each box represents a single algorithm for offset assignment.

However, while a certain absolute value for a delay (for example, 1000 time units) can mean a significant deadline miss (e.g., a task with period 500 with implicit deadline), for others it can mean only an "affordable" amount of 10% of its period (e.g., a task with period 10000 in a system with few and low-utilization tasks). Therefore, we must also evaluate the ratio between each delay and each corresponding task period to better evaluate the obtained results.

[1] https://github.com/paparazzi/paparazzi/blob/master/conf/telemetry/default_rotorcraft.xml

If a task was delayed at most by 100 time units in a system in which the shortest task executes in 200 time units, then it performed better than one that had the same maximum delay but in a system where the maximum execution time is 50, indicating that task delays have been chained and, in our case, probably more messages are waiting in the queue. Hence, to look for signs of chaining delays, we also analyse the ratio between the delay and the maximum execution time of other tasks in the set.

To evaluate the response time of each task, the maximum delay summed to the execution time of each task is normalized with respect to each execution time. Values close to 1 will indicate a small relative change to the response time.

A final analysis takes into account the schedulability of each task set. If any implicit deadline was missed, the task set is marked as unschedulable under FIFO. Then, the amount of schedulable sets is counted for each offset-assignment method, and they are compared between each other for each value of the total utilization factor.

In summary, the metrics used in the analysis are in the following list:

1. interference of other tasks η_i
2. interference normalized by period η_i/T_i
3. interference normalized by concurrent tasks' maximal duration $\eta_i/\max_{\{\forall \tau_k \neq \tau_i\}}(c_k)$
4. maximum response time $(\eta_i + c_i)/c_i$
5. schedulability

6.2 Results

The codes of some heuristic methods cited in Section 4, as well as the new contribution were implemented and executed in Python 3.9.6, using a laptop with Ubuntu 18.04.1, Intel Core i7-4710MQ CPU (2.50GHz, 4 cores, 8 threads, 2054MHz), and 16 GB of RAM.

GCD+ is comparatively evaluated using sets of 8 and 16 tasks. The limit on the number of tasks is due to the duration of the simulation used to compute the metrics. The time for each method to yield an offset assignment is measured and, later, the resulting offset assignments are evaluated in a simulation according to the criteria presented before. The simulation stores the maximum delays of each task (the difference between the time it starts its execution and the time it was called). Also, simulations with 1000 sets and then 2000 sets resulted in identical graphs, so we use only 1000 sets as a sufficient sample. The offset calculation time for each group of sets is shown in Table 1 for sets with 70% utilization. Other utilization values did not present significant variations.

Table 1 Average time to assign offsets (results from 1000 filtered sets at 70% utilization).

Method	Time - 8 tasks (µs)	Time - 16 tasks (µs)
GCD+	54.8	151
Paparazzi method [17]	4.78	9.11
Goossens's method [7]	22.8	74.7
CAN message method [9]	45.7	346

Table 1 shows that the time to calculate offsets is very small from a human perspective for every method. It also shows that every method is scalable, and therefore we can focus solely on their results.

Box-plots are used to represent simulation results, where each heuristic method has its box. In the plots, the median (50^{th} percentile) is represented as an orange continuous bar, the limits of the box are the first quartile (25^{th} percentile) and third quartile (75^{th} percentile),

and whiskers span from a box limit until the furthest value such that its size is, at most, 1.5 times the distance between the first and third quartile. All values beyond the whisker limits are considered outliers and are not represented in order to allow readability. However, the representations showing outliers follow the same tendencies as shown in the following figures. The mean value is also represented, as a dashed green bar.

For sets of 8 tasks, the behaviour of the systems can be seen in Figure 6 for 70% utilization and in Figure 7 for 90% utilization. Similarly, the behaviour for 16 task systems is seen in Figure 8 for 70% utilization and in Figure 9 for 90% utilization. In these figures, the box plot furthest to the left shows the first metric (η_i), the one at its right side shows the second metric (η_i/T_i) and so on.

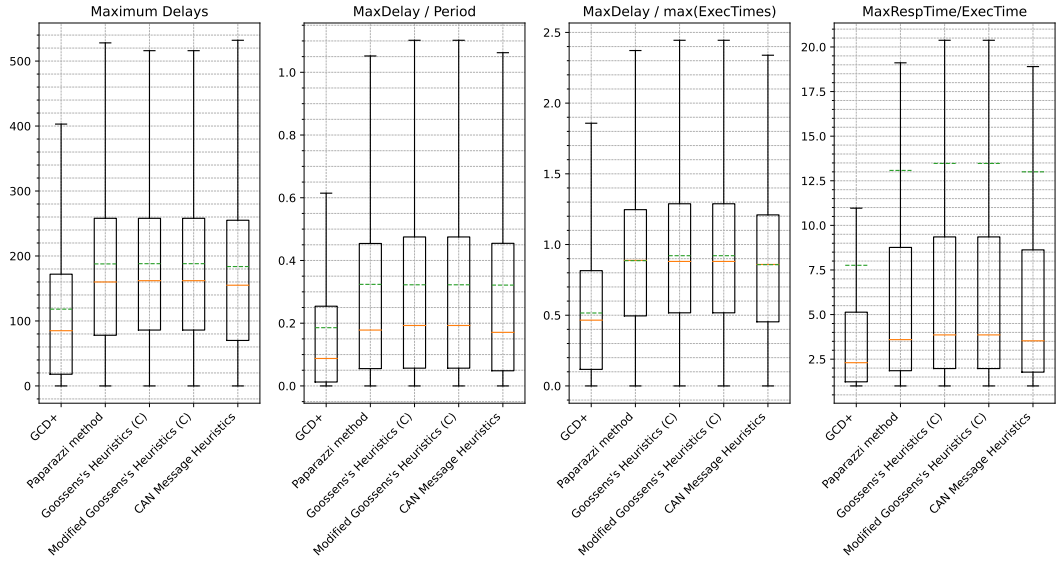


Figure 6 Delays extracted from simulations for 1000 filtered sets of 8 tasks, with $U = 70\%$.

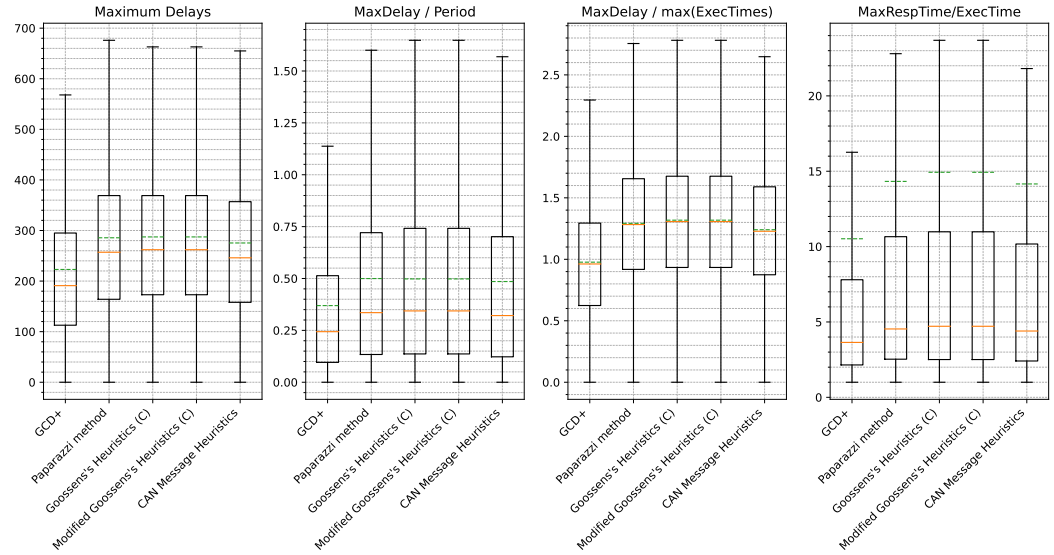


Figure 7 Delays extracted from simulations for 1000 filtered sets of 8 tasks, with $U = 95\%$.

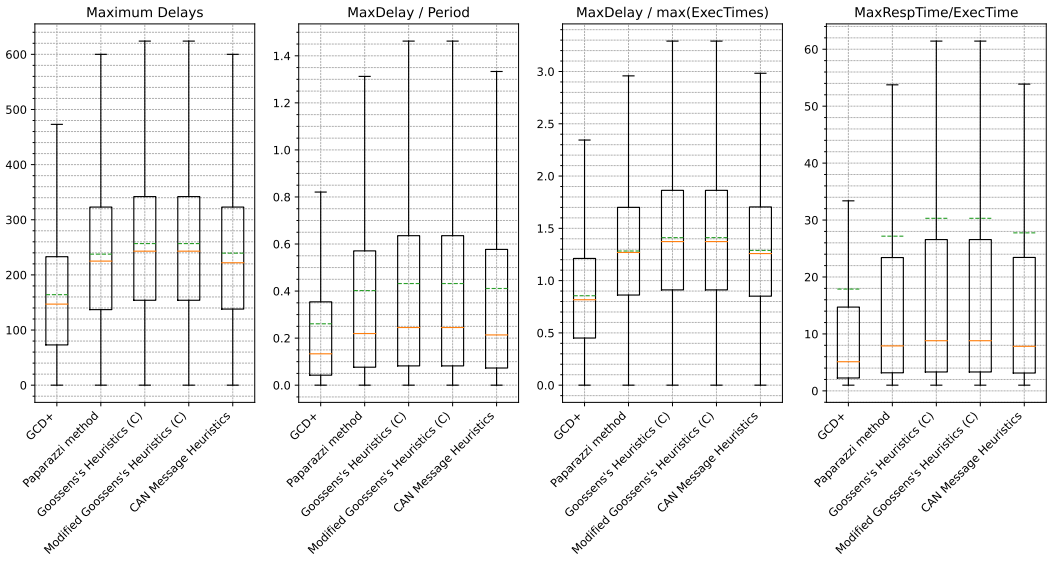


Figure 8 Delays extracted from simulations for 1000 filtered sets of 16 tasks, with $U = 70\%$.

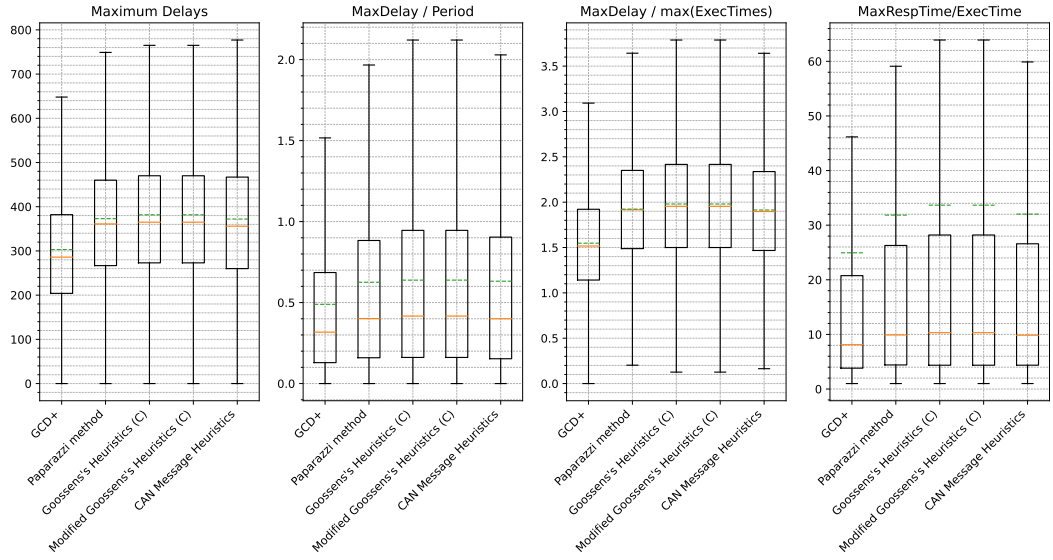


Figure 9 Delays extracted from simulations for 1000 filtered sets of 16 tasks, with $U = 95\%$.

From the box plots, it can be seen that, for GCD+, every quartile (first, second and third) is significantly closer to the ideal value in every situation than those for the other heuristics, as well as the boundaries of the whiskers and the mean values. For utilization factors of 70%, GCD+ was able to keep the delay of the great majority of tasks under 60% and 80% of their periods in the case of sets of 8 and 16 tasks, respectively. Other heuristics, however, show whiskers going beyond the value of 100% in these same situations.

It is also noticeable how every heuristic method has more trouble reducing delays when the task count increases, notably with respect to the ratio between maximum response times and execution times – this ratio tripled when the task count doubled. Increasing the utilization factor is another factor that worsens the maximum delays for every heuristic method.

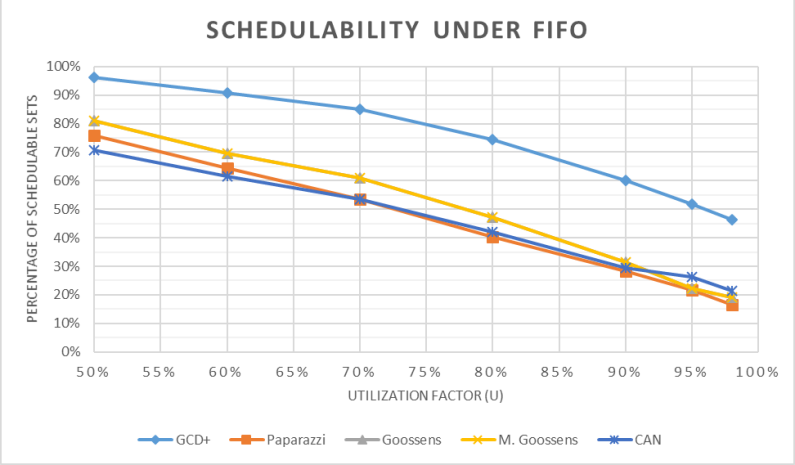


Figure 10 Schedulability for 1000 filtered sets of 8 tasks, from $U = 50\%$ to 98%.

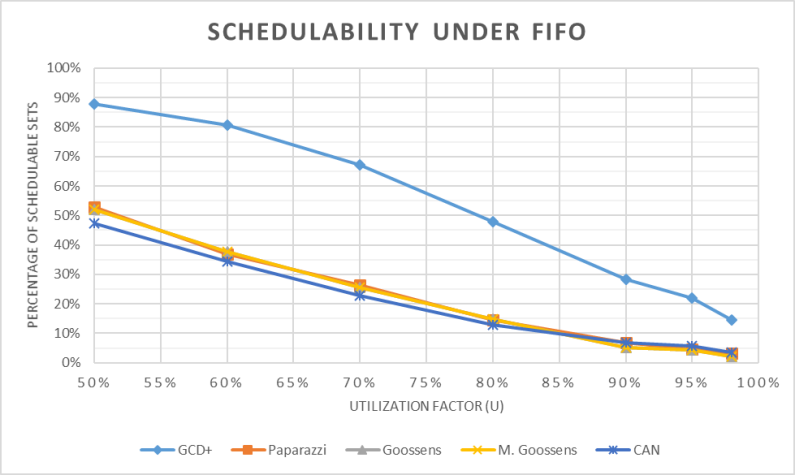


Figure 11 Schedulability for 1000 filtered sets of 16 tasks, from $U = 50\%$ to 98%.

In order to evaluate the schedulability of these task sets with respect to the utilization factors, an implicit deadline was considered for every task, and if a set has a task that missed a deadline, it is considered unschedulable. Figure 10 and Figure 11 were then obtained for sets of 8 and 16 tasks, respectively. GCD+ shows a significant advantage relative to every other heuristic in every situation that is shown. For sets of 8 tasks, at 80% utilization factor, there is an increase of at least 50% in the number of schedulable sets, while for 90% utilization there are about twice as much schedulable sets when using GCD+. For sets of 16 tasks, the increase is even more pronounced: from about 2x at 50% utilization to about 4x at 90%.

The significant improvement seen in the presented results can be explained by certain limitations of the other methods. Paparazzi method tries to distribute the offsets without taking into account the relations between the messages. The CAN Message heuristic method, when trying to distribute the first calls over the maximum period, ignores the effects over the hyperperiod, during which two tasks that were initially put far away can fall into a critical case. The Goossens's heuristic method, although it considers some interactions between

pairs of tasks during the whole hyperperiod, has a trap: when several tasks have the same period, it will tend to assign the same offset to all of them, intentionally creating a critical case. Furthermore, its random component might lead to undesired critical cases. The fact that it does not take into account the execution times of tasks has no apparent effect in its results, since the Modified Goossens's Heuristic (which does take them into account) had the exact same results as the original method.

7 Case Study

An adapted real-world scenario was set up in order to measure the applicability of GCD+. A telemetry configuration file was used to define the messages that a Paparazzi autopilot implementation sends periodically, such that the telemetry is composed of the messages described in Table 2. The column c indicates the content size of the message, without adding any protocol header or tail.

Table 2 Paparazzi telemetry values.

i	ID	T (s)	c (B)
1	ALIVE	2	17
2	ROTORCRAFT_FP	1	58
3	INS_REF	1	32
4	ROTORCRAFT_NAV_STATUS	1	15
5	ENERGY	1	21
6	DATALINK_REPORT	1	11
7	DL_VALUE	0.2	5
8	ROTORCRAFT_STATUS	0.2	20
9	STATE_FILTER_STATUS	0.2	4
10	AIR_DATA	0.2	28
11	INS	0.2	36
12	GPS_INT	0.1	57
13	IMU_GYRO_SCALED	0.04	12
14	IMU_ACCEL_SCALED	0.04	12
15	IMU_ACCEL_RAW	0.02	12
16	IMU_GYRO_RAW	0.02	12

The messages are sent to a UART channel, which is normally connected to an antenna to transmit the data, but for this test and to ignore any effects related to the transmission of radio waves, messages are read directly in the UART channel in our experiment. According to its protocol, every byte has a start and an end bit added to it, which makes it transmit 10 bits per byte. It was configured to transmit 57600 bits per second. In addition, the Paparazzi protocol used (Pprzlink V2.0) adds 8 bytes to each message for header and checksum[2].

Therefore, to use the time it takes for a bit to be transmitted as a unit of time, i.e., 1/57600 second, values for periods have to be multiplied by 57600, and the amount of bytes in every message, after the 8 Pprzlink bytes are added, have to be multiplied by 10. With these converted values, we can analyse this case using GCD+. Its results are presented in Figure 12.

Figure 12 shows that, using GCD+, every message has a delay under 10% of its period, in comparison to 80%, 170% and even 270% in other heuristics. These delays represent at most 20% of the maximum length of other messages, while the third metric shows that there was a significant chaining of interferences for all the other heuristics. Also, deadline misses were registered for the methods Goossens and Paparazzi, while the system was schedulable using offsets from GCD+ and CAN Message heuristics.

[2] `https://wiki.paparazziuav.org/wiki/Messages_Format`

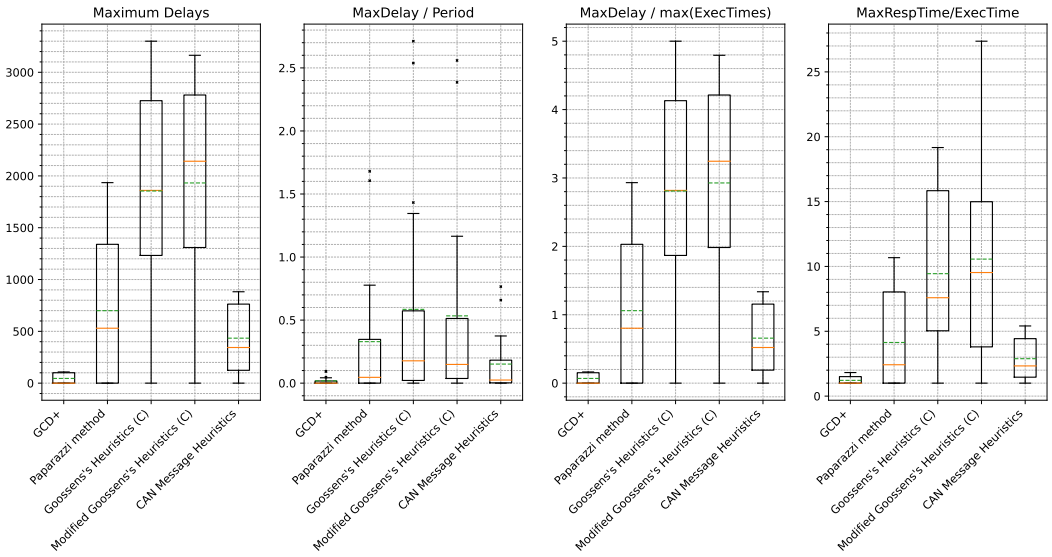


Figure 12 Delays extracted from simulations for the case study.

From the physical analysis, before the implementation of GCD+, some messages were lost due to a full buffer, as it can be seen in Figure 13. The figure is a screen capture from the software Logic 2, for analysing the data acquired from the UART setup. The first line (Channel 0) represents the raw bits sent in the UART port; the second one (Channel 1) goes from a low to a high state every time the autopilot blocks the UART buffer, as it is writing a message to be sent in the channel, and from high to low when in releases the buffer; the third line (Channel 2) goes from high to low or from low to high when the buffer is already full (a lot of messages in the queue) and, hence, the buffer refused to accept a message from the autopilot (i.e., a message loss). It can be seen that there is a message loss at every 2 seconds. This problem was avoided with the offsets given by the heuristics presented in this paper, as it can be seen in Figure 14, where no message is ever lost. The image shows less than 3 seconds of data capturing, but the complete test was made for 20 seconds.

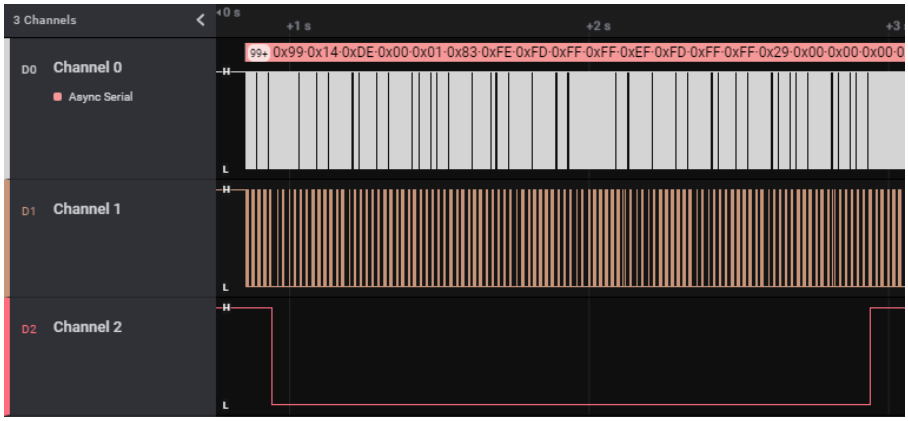


Figure 13 Screen capture from the analysis of a Paparazzi telemetry case with a message loss at every 2 seeconds.

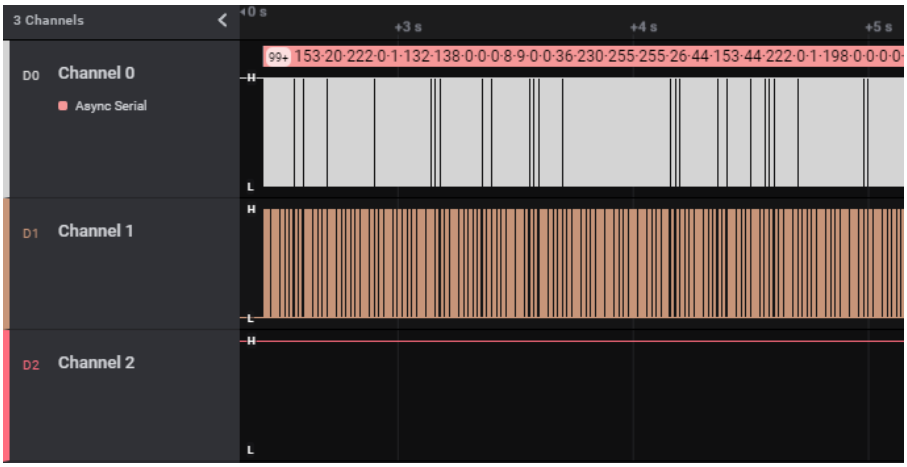


Figure 14 Screen capture from the analysis of an improved Paparazzi telemetry case with no more message losses.

This case study confirms what can be seen in simulations. Moreover, the case studies in telemetry usually confirm the fact that the global GCD is large, giving an advantage to GCD+.

8 Conclusion

In this article, we have proposed GCD+ a new method for finding suitable offsets for tasks in execution or messages in transmission under FIFO scheduling. It was compared in simulations to State-of-the-Art methods using a random task set generator and in a real case, presenting better results than the others in both scenarios. In addition, the new method was compared against the one used in Paparazzi in a physical setup of the autopilot, confirming the better result obtained in the simulation. GCD+ proved its efficiency, notably for the cases of a semi-harmonic set where the maximum execution time is not greater than the GCD of all task periods.

GCD+ can be extended in the future to support other cases. For example, GCD+ could be extended to handle multi-periodic precedence constraints, such as described in [14]. It would allow applying this method to the computation of offsets in the case of monolithic periodic tasks integrating a static scheduler, which are central in most UAV autopilots. Also, the integration of GCD+ into the configuration tool of Paparazzi is planned in the near future.

References

1 Sebastian Altmeyer, Sakthivel Manikandan Sundharam, and Nicolas Navet. The case for fifo real-time scheduling. Technical report, University of Luxembourg, 2016.

2 Chaitanya Belwal and Albert MK Cheng. Generating bounded task periods for experimental schedulability analysis. In *2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing*, pages 249–254. IEEE, 2011.

3 Pascal Brisset, Antoine Drouin, Michel Gorraz, Pierre-selim Huard, and Jeremy Tyler. The Paparazzi Solution. *HAL*, 2006.

4 Vicent Brocal, Patricia Balbastre, Rafael Ballester, and Ismael Ripoll. Task period selection to minimize hyperperiod. In *ETFA2011*, pages 1–4. IEEE, 2011.

5 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

6 Joel Goossens and Christophe Macq. Limitation of the hyper-period in real-time periodic task set generation. In *In Proceedings of the RTS Embedded System (RTS'01*. Citeseer, 2001.

7 Joël Goossens. Scheduling of offset free systems. *Real-Time Systems*, 24(2):239–258, 2003.

8 Mathieu Grenier, Joël Goossens, and Nicolas Navet. Near-optimal fixed priority preemptive scheduling of offset free systems. In *14th International Conference on Real-Time and Networks Systems (RTNS'06)*, pages 35–42, 2006.

9 Mathieu Grenier, Lionel Havet, and Nicolas Navet. Pushing the Limits of CAN - Scheduling Frames with Offsets Provides a Major Performance Boost. *4th European Congress on Embedded Real Time Software (ERTS 2008)*, 2008.

10 David Griffin, Iain Bate, and Robert I Davis. Generating utilization vectors for the systematic evaluation of schedulability tests. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 76–88. IEEE, 2020.

11 Gautier Hattenberger, Murat Bronz, and Michel Gorraz. Using the paparazzi UAV system for scientific research. In *IMAV 2014, International Micro Air Vehicle Conference and Competition 2014*, page 247, 2014.

12 Mitra Nasri, Robert I Davis, and Björn B Brandenburg. Fifo with offsets: High schedulability with low overheads. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 271–282. IEEE, 2018.

13 Mitra Nasri and Gerhard Fohler. An efficient method for assigning harmonic periods to hard real-time tasks with period ranges. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 149–159. IEEE, 2015.

14 Thanh-Dat Nguyen, Yassine Ouhammou, Emmanuel Grolleau, Julien Forget, Claire Pagetti, and Pascal Richard. Design and analysis of semaphore precedence constraints: A model-based approach for deterministic communications. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 231–236. IEEE, 2018.

15 Réda Nouacer, Mahmoud Hussein, Huascar Espinoza, Yassine Ouhammou, Matheus Ladeira, and Rodrigo Castiñeira. Towards a framework of key technologies for drones. *Microprocessors and Microsystems*, 77:103142, 2020.

16 Paparazzi developers. Paparazzi home page. `https://paparazziuav.org`. Accessed: 02-02-2022.

17 Paparazzi developers. Paparazzi offset generation source code. `https://github.com/paparazzi/paparazzi/blob/master/sw/tools/generators/gen_periodic.ml`. Accessed: 02-02-2022.

18 Rodolfo Pellizzoni and Giuseppe Lipari. Feasibility analysis of real-time periodic tasks with offsets. *Real-Time Systems*, 30(1-2):105–128, 2005. `doi:10.1007/s11241-005-0506-x`.

19 Ismael Ripoll and Rafael Ballester-Ripoll. Period selection for minimal hyperperiod in periodic task systems. *IEEE Transactions on Computers*, 62(9):1813–1822, 2012.

20 Jia Xu. A method for adjusting the periods of periodic processes to reduce the least common multiple of the period lengths in real-time embedded systems. In *Proceedings of 2010 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*, pages 288–294. IEEE, 2010.