Verification of Two Real-Time Systems Using Parametric Timed Automata

Youcheng Sun, Giuseppe Lipari, Étienne André

Verification of Two Real-Time Systems Using Parametric Timed Automata

Youcheng Sun
Scuola Superiore Sant'Anna
Pisa, Italy

Étienne André
LIPN, Université Paris 13,
Sorbonne Paris Cité, France

Giuseppe Lipari
CRIStAL - UMR 9189
University of Lille, France

***Abstract*—In this paper we propose solutions to the FMTV challenge of a distributed video processing system using the formalism of Parametric Timed Automata (PTA). The first challenge is harder because of the very large number of states to be analysed, so we only provide upper bounds. The second challenge consists of a real-time scheduling problem for which we provide exact solutions by using a scheduling analysis based on the critical instant, and a PTA model.**

I. Introduction

With the increasing size and complexity of concurrent systems, the need for formal verification techniques has become higher and higher in the past decades. Systems mixing time and concurrency are especially subject to undesired behaviours (deadlocks, race conditions, etc.); often, their complexity makes the use of formal methods very challenging.

In this paper, we address the Formal Methods for Timing Verification Challenge (FMTV 2015) proposed by Thales (http://waters2015.inria.fr/challenge) by using the formalism of *parametric timed automata* (PTA). Timed automata (TA) [AD94] are a well-known formalism for specifying and verifying concurrent real-time systems. TA extend finite-state automata with a set of clocks (real-time variables growing linearly) that can be compared with integer constants. TA are used in several powerful tools such as Uppaal [LPY97] or PAT [SLDP09]. However, the binary answer ("yes" or "no") output by model checking is not always satisfactory; indeed, it does not allow to change or optimize some values of the system constants, nor (in general) to evaluate the system robustness, *i.e.,* the infinitesimal variation of timing constants while preserving the reachability. PTA [AHV93] extend TA with rational-valued parameters allowed in place of constants. PTA are particularly well suited to verify systems where some timing delays are known with uncertainty. The natural drawback of PTA is the infamous state space explosion, that may prevent the verification to be truly scalable. We will use the tool Imitator [AFKS12] that takes PTA as an input formalism.

Outline: We briefly review PTA and Imitator (Section II). Then, we recall the challenge (Section III). We derive solutions to Challenge 1A empirically (Section IV), and formally by using Imitator (Section V). We provide a different PTA model for the much harder Challenge 1B (Section VI) for which we derive an upper bound. Finally we analyse the problem in Challenge 2 and provide a solution based on schedulability analysis (Section VII-A) and a solution based on a PTA model (Section VII-B). Finally, we conclude in Section VIII.

This work is partially supported by the ANR research program PACS (ANR-14-CE28-0002), and the INS2I PEPS JCJC 2015 PSyCoS project.

II. Parametric Timed Automata

Timed automata are finite-state automata augmented with clocks, *i.e.,* real-valued variables increasing uniformly, that are compared within guards and invariants with timing delays [AD94]. Parametric timed automata (PTA) [AHV93] extend timed automata with parameters, *i.e.,* unknown constants, that can be used in guards and invariants.

Given a set X of clocks (real-valued variables) and a set P of parameters (unknown rational-valued constants), a constraint C over X and P is a conjunction of linear inequalities on X and P[1]. Given a parameter valuation (or point) v, we write $v \models C$ when the constraint where all parameters within C have been replaced by their value as in v is satisfied by a non-empty set of clock valuations.

Definition 1: A parametric timed automaton (PTA) $\mathcal{A}$ is $(\Sigma, L, l_0, X, P, I, E)$ with Σ a finite set of actions, L a finite set of locations, $l_0 \in L$ the initial location, X a set of clocks, P a set of parameters, I the invariant assigning to every $l \in L$ a constraint over X and P, and E a step relation consisting of elements (l, g, a, R, l'), where $l, l' \in L$, $a \in \Sigma$, $R \subseteq X$ is the set of clocks to be reset, and the guard g is a constraint over X and P.

The semantics of a PTA $\mathcal{A}$ can be found in, *e.g.,* [AHV93], [AS13].

Most problems related to PTA (*e.g.,* the parametric reachability of a location) are undecidable [AHV93], [JLR15], with some decidable syntactic subclasses related to the use of the parameters [HRSV02], [BL09], [JLR15] or on the number of clocks [AHV93], [BO14], [BBLS15]. We do not consider it as drawback, as we use semi-algorithms that "often" terminate in practice; we will see this is also the case for solving challenge 1A.

Imitator [AFKS12] is a tool for modeling and verifying systems modeled using parametric timed automata. In its latest

[1]Note that this is a more general form than the strict original definition of PTA [AHV93]; since most problems for PTA are undecidable anyway, this has no practical incidence, and increases the expressiveness of the formalism.

version (2.7-beta2), IMITATOR implements several algorithms, among which:

- parametric reachability analysis: "find all parameter valuations such that some state is reachable"
- parametric robustness analysis: "given a parameter valuation v, find other valuations for which the discrete (untimed) behavior is the same as v"
- behavioral cartography: "find all parametric subspaces in which the discrete behavior is uniform".

IMITATOR extends PTA with some useful constructions, such as constants, global discrete integer variables, strong broadcast synchronization between components, and stopwatches (*i.e.,* the power of stopping some clocks in some locations).

The input syntax of IMITATOR is a text file, which makes it possible to write models either manually or using scripts (*e.g.,* for large models derived from another formalism). For example, in the past several parametric schedulability analysis models were generated automatically using scripts, and asynchronous circuits of arbitrary size can be modeled in IMITATOR from VHDL syntax using the tool VHDL2TA[2].

III. A BRIEF DESCRIPTION OF THE SYSTEM

We only briefly recall the systems and the challenges; the full description is available at the workshop Web page.

A. Challenge 1

There are four tasks T1, T2, T3 and T4, distributed in different processing units and performing respective functionalities. The task T1 periodically receives frames from the camera and pre-processes them. Task T2 embeds further tracking information into the video frame pre-processes by Task T1. Task T2 then inserts the video frame into a register, denoted as Register23. Then Task T3 reads the frame from the register, removes the noise and tries to put the resulting video frame into a buffer, denoted as Buffer34. In the end, Task T4 reads frames from the buffer, converts them from digital to analogue and sends the final frame to the display.

Tasks T1, T3 and T4 are periodic, but their triggering clocks are subject to drift. That is, their periods P_1, P_3 and P_4 are unknown constants. More specifically, $\mathrm{P}_1 \in [40 - 40 \times 0.01\%, 40 + 40 \times 0.01\%]$ms, $\mathrm{P}_3 \in [\frac{40}{3} - \frac{40}{3} \times 0.05\%, \frac{40}{3} + \frac{40}{3} \times 0.05\%]$, and $\mathrm{P}_4 \in [40-40\times0.01\%, 40+40\times0.01\%]$ms. Task T2 is triggered by the completion of T1.

Each task has its Best-Case and Worst-Case Execution Time (BCET and WCET) or Latency (BCL and WCL): $\mathrm{BCET}_1 = \mathrm{WCET}_1 = 28$ms, $\mathrm{BCL}_2 = 17$ms, $\mathrm{WCL}_2 = 19$ms, $\mathrm{BCET}_3 = \mathrm{WCET}_3 = 8$ms. As for task T4, when it reads Buffer34 and there is no frame within the buffer, it performs an empty cycle with execution 1ms; otherwise, it executes 10ms and sends the result to display.

For such a video frame processing subsystem, we aim to tackle the following challenges.

[2]http://www-soc.lip6.fr/~bara/valmem/vhdl2ta/rapports/vhdl2ta-web/vhdl2ta.html

Challenge 1A:
Compute the minimum and maximum latencies for a given frame from the camera output to the display input, for a buffer size $n = 1$ (challenge 1A.1) and $n = 3$ (challenge 1A.3).

Given the difficulty of Challenge 1A, we will focus on deriving the lower bound and upper bound of the end-to-end latency.

Challenge 1B:
Due to the different clock drifts, all frames with a same index may be discarded at the entrance of the buffer at the input of the task T4. Compute the minimum time distance between two frames produced by the camera that will not reach the display, for a buffer size $n = 1$ (challenge 1B.1) and $n = 3$ (challenge 1B.3).

B. Challenge 2

The complete system also includes a camera tracking subsystem that identifies objects on the camera images and commands the camera motors so to follow the objects. This subsystem consists of 3 additional tasks: Task T6 is a periodic task with period $\mathrm{P}_6 = 100$ and it can start with a certain jitter J_6. Task T5 is activated by Task T6 with a synchronous call. Task T7 is activates asynchronously by Task T6 and controls the motors. Task T6 execution time is: $C_{6,1} = 4$ms before invoking Task T5; $C_{6,2} \in [9, 10]$ ms after the completion of Task T5 and before the invocation of Task T7; $C_{6,3} \in [4, 5]$ ms after the invocation to Task T7. Task T5 has an execution time of $C_5 \in [4, 7]$ ms. Task T7 has an execution time of $C_7 \in [11, 14]$ ms. All tasks execute on the same processor together with Task T2 (described in the first challenge), and they are scheduled by a fixed priority scheduler. Task T2 has a computation time of $C_2 = 17$ms. Finally, task priorities are assigned so that T2 $>$ T6 $>$ T5 $>$ T7.

Challenge 2A:
1) Compute the best and worst-case end-to-end latencies from the activation of Task T6 to the completion of Task T7 when $J_6 = 0$.
2) Compute the best and worst-case end-to-end latencies when $J_6 = 20$ms.

Challenge 2B:
Tasks T2 and T5 have access to a shared mutually exclusive resource protected by the priority ceiling protocol. The access to the shared resource takes 2ms for both tasks. 1) Compute the best-case and worst-case end-to- end latencies from activation of T6 to termination of T7 for a jitter value $J_6 = 0$ms 2) compute the best-case and worst-case end-to-end latencies from activation of T6 to termination of T7 for a jitter value $J_6 = 20$ms. 3) The optimum priority assignment minimizing the worst-case latency for a jitter value $J_6 = 0$ms and $J_6 = 20$ms.

IV. SOLVING CHALLENGE 1A EMPIRICALLY

First, we can try to find the latency using a manual estimation. We claim that the minimum latency occurs in the following scenario (times are given in ms):

1) The frame is output by the camera at $t = 0$
2) The frame is processed immediately by Task T1 $t = 28$
3) The frame is processed immediately by Task T2, which takes a minimal time (BCL_2=17) $t = 45$
4) The frame is processed immediately read task 3 $t = 45$

5) After task 3's execution, the frame is inserted into an empty buffer $t = 53$
6) Immediately, the frame is got by task 4 $t = 53$
7) Task T4 processes the frame and sends it to display $t = 63$

In fact nothing prevents this scenario to happen: with tasks appropriately synchronised with each other, this scenario can indeed happen. That is, 63ms is the exact minimum latency. Even with $n = 3$, the fastest scenario built above still holds. And the minimum latency is still 63 ms.

Then, we claim that an upper bound on the maximum latency occurs in the following scenario (times are given in ms):

1) The frame is output by the camera $t = 0$
2) The frame is processed immediately by task 1 $t = 28$
3) The frame is sent to task 2, which takes a maximal time (WCL_2=19) $t = 47$
4) The frame can stay within Register23 for at most P_1 ms before it is overwritten by the successive frame $t \leq 87.004$
5) Task T3 processes the frame and puts it in Buffer34 $t \leq 95.004$
6) Buffer34 can host the frame for at most P_4 ms $t \leq 135.008$
7) Task T4 takes 10ms to execute the frame $t \leq 145.008$

Thus, an upper bound to the maximal latency is 145.008ms.

For the maximum latency with $n = 3$, based on the result with $n = 1$, we can derive a safe upper bound immediately. This upper bound is $\leq 145.008 + 2 \times \mathrm{P}_4 \leq 225.016$ms.

V. Solving Challenge 1A Using Imitator

In this part, we formally solve the challenge using PTA. At first, we will solve the case with $n = 1$ for Buffer34. The corresponding PTA model is shown in Figure Fig. 1. Later, we show how to adapt this model to a larger buffer.

Experimental environment: We used the latest version of Imitator (v2.7-beta2, build 1073) with no specific modification of the tool. We just used a small Python script to parse the long list of intervals that Imitator outputs, and to produce a single minimum and maximum. Sources, binary and models are available at [XP].

A. Camera, Task T1, Task T2

In order to reduce the state space, we model the camera, Task T1 and Task T2 into a single PTA (Fig. 1a). We also use this PTA to non-deterministically initialize the buffer and the frame currently processed by Task T4.

We choose an arbitrary frame with index target for end-to-end latency estimation and *we start from the exact point such that the target frame is handled from Task T1 to task T2*. A clock ckT1T2 is *initialised to be* WCET_1 and measures the end-to-end latency of target frame. Discrete variables frame_in_3 and frame_in_4 represent the index of frames in Task 3 and Task 4 respectively. The value 0 is used to denote that there is no frame in a task. $\mathrm{reg}_{2,3}$ and $\mathrm{buffer}_{3,4}$ are for frames within Register23 and Buffer34. While we assume frame index in the system is monotonically increased, $\mathrm{highest}_{3,4}$ denotes the highest frame index among frames having been stored inside the buffer. For the register, buffer and Task 3, they may or may not initially contain a frame.

We do not model the period of the camera (or task 1), since we are only interested in a single frame. Let us now model the buffer. Imitator does not support other kinds of global variables than discrete integer variables. For a buffer with one slot ($n = 1$), its status can be modeled by using two discrete variables $\mathrm{buffer}_{3,4}$ and $\mathrm{highest}_{3,4}$.

- $\mathrm{buffer}_{3,4}$ denotes the index of current frame inside Buffer34;
- $\mathrm{highest}_{3,4}$ is the highest index recorded so far.

B. Task T3

The period of Task T3 is a *parameter* P3_uncertain, that is initialised as follows:

$$\mathrm{P3\_uncertain} \in [40 - \mathrm{P3\_delta}, 40 + \mathrm{P3\_delta}]$$

where $\mathrm{P3\_delta} = 0,05\,\% \times 40 = \frac{1}{150}$. Recall that parameters in PTA are unknown *constants*, *i.e.,* the value of which cannot evolve during the execution; this is exactly what we need to model P3_uncertain.

Task T3 is modeled by a periodic PTA in Fig. 1b. ckT3 is a clock variable for recording task 3's activation and execution. At the initial point, the PTA T3 is non-deterministically waiting for a new activation or executing. When T3 finishes execution, it writes into Buffer34 if the buffer is empty and its current frame has not been put into the buffer. We define a function call for this writing operation write_by_T3():

$$\mathrm{buffer}_{3,4} := \mathrm{highest}_{3,4} := \mathrm{frame\_in\_3}$$

Imitator does not support function call in a model; here we still utilise the notation of function call for simplicity. Otherwise, task 3's writing fails. As shown in PTA T3, we utilise the operator "||" ("or") in the edge representing writing failure. Again, this is for saving some space.

C. Task T4

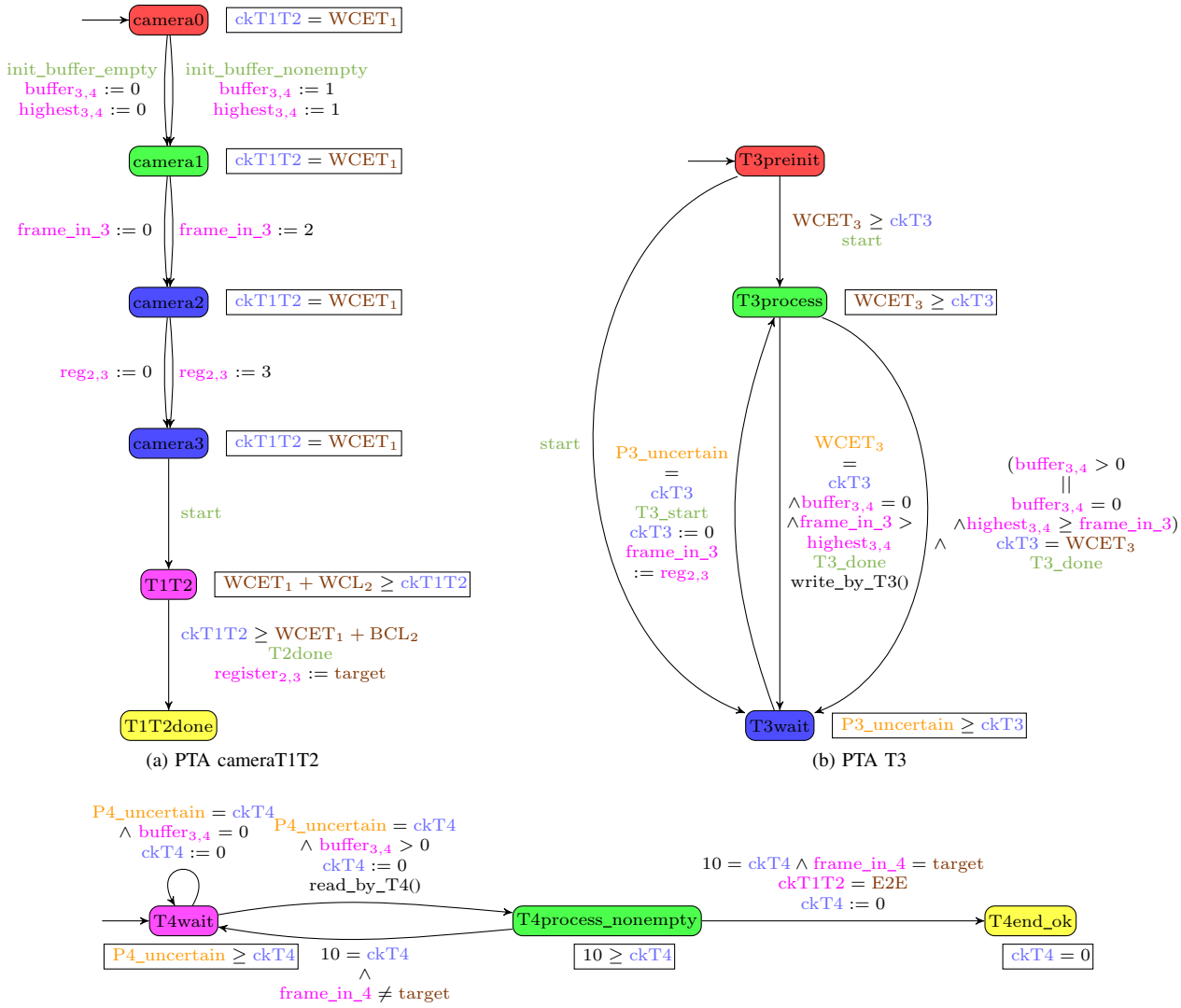
We use here a modeling mechanism similar to Task T3. The period of Task T4 is a *parameter* P4_uncertain, that is initialised as follows:

$$\mathrm{P4\_uncertain} \in [40 - \mathrm{P4\_delta}, 40 + \mathrm{P4\_delta}]$$

where $\mathrm{P4\_delta} = 0,01\,\% \times \mathrm{P4} = 0.004$.

Task T4 is modeled by a periodic PTA as in Fig. 1c. A clock variable ckT4 is used for task 4's periodic activation and execution. When T4 activates, If the buffer is empty, T4 goes directly back to another waiting cycle. According to the system specification, there should an empty processing session for task 4. However, such an empty processing does not affect the end-to-end latency of any frame, and we omit it in PTA T4. If the buffer is not empty, task 4 reads a frame from the buffer by the function call read_by_T4():

$$\mathrm{frame\_in\_4} := \mathrm{buffer}_{3,4},\ \mathrm{buffer}_{3,4} := 0$$



(a) PTA cameraT1T2

(b) PTA T3

(c) PTA T4

Fig. 1: Modeling the system of Challenge 1A for $n = 1$

Task T4 takes 10ms to process a frame, after the processing, if its current frame is the target one, ckT4 is reset and T4 moves to an ending location. E2E ≥ 0 is the parameter for representing all possible end-to-end latencies of the target frame.

D. Deriving the Latency for $n = 1$

As we have seen, in order to avoid exploring the exact configurations in the system, we target a single frame (which explains the non-cyclic behavior of the PTA modeling the camera, tasks 1 and 2) that is output from the task 1 at $t = \text{WCET}_1$. The main idea is that, at $t = \text{WCET}_1$, the initial state must be *arbitrary*, *i.e.,* encode all possible configurations that could happen in the system. However, such a model may be pessimistic for containing behaviors that cannot really happen in the system. Again, we aim to derive upper and lower bounds on end-to-end latency of an arbitrary frame.

After developing the model, we use IMITATOR to perform parametric reachability analysis of location T4end_ok, that is we ask IMITATOR to return all parameter valuations such that T4end_ok is reachable. Then, IMITATOR hides (using existential quantification) all parameters except E2E, and then returns a list of intervals for E2E. After some post-processing to unify intervals, we get

$$\text{E2E} \in [63, 145.008].$$

This result is compatible with our empirical estimation. The maximum end-to-end latency is empirically bounded by explicitly considering the time a successive frame arrives; while the PTA model does not consider frames after the target one, the result by IMITATOR still matches the empirical reasoning.

E. Deriving the Latency for $n = 3$

For the case of $n = 3$ for Buffer34, we can keep the same IMITATOR model, with the exception of the buffer modeling.

We assume that access to elements in the buffer follows a FIFO manner.

Let us model Buffer34 for $n = 3$: besides the variable $\text{highest}_{3,4}$, we need 3 more discrete variables for frames within each slot in the buffer: $\text{buffer}^1_{3,4}$, $\text{buffer}^2_{3,4}$ and $\text{buffer}^3_{3,4}$. When non-deterministically initializing the buffer, we need to take into account all possible scenarios of frame occupation within it. When T3 writes into the buffer, it needs to find the first free position (say xth); thus the writing call becomes write_by_T3():

$$\text{buffer}^{\text{x}}_{3,4} := \text{highest}_{3,4} := \text{frame\_in\_3}.$$

Similarly, for T4 to read from the buffer, we call the adapted function read_by_T4():

$$\text{frame\_in\_4} := \text{buffer}^1_{3,4}$$

$$\forall x \in \{1, 2\}\ \text{buffer}^{\text{x}}_{3,4} := \text{buffer}^{\text{x+1}}_{3,4}$$

$$\text{buffer}^3_{3,4} := 0$$

Note that the buffer status is also updated by the reading operation such that the first slot always contains the oldest frame.

Then, we can apply parametric analysis on the model for $n = 3$ using IMITATOR. A projection of all possible values to parameter E2E gives the following result:

$$\text{E2E} \in [63, 225.016].$$

Again, this matches our empirical estimation. Finally, we conclude results for Challenge 1A in Table I.

Buffer34 size	min E2E	max E2E
$n = 1$	63 ms	63 ms
$n = 3$	145.008 ms	225.016 ms

TABLE I: E2E latency results for Challenge 1A

VI. SOLUTION TO CHALLENGE 1B

This challenge is the most difficult to solve with a formal model. The first problem we encountered is that the analysis must keep track of many frames to measure the distance between two frame losses, and the number of frames to analyze can be very large.

In the long term the average rate of dropped frames depends on the rate between the period of the producer task (T1) and the period of the consumer task (T4). In particular, the average frame loss rate can be computed as $(n_1 - n_4)/\text{lcm}(\text{P}_2^{\min}, \text{P}_4^{\max}) \approxeq 0.005$, where n_1 and n_4 are the number of instances of T1 and T4 in the hyperperiod between T1 and T4, respectively (and lcm the least common multiple). By computing the formula, the average distance between 2 frame losses is $\approxeq$ 200 sec., *i.e.,* one loss every 5000.5 frames. Therefore the minimum distance between two frame loss is $\leq$ 200 seconds.

Unfortunately, computing the actual minimum distance is a very difficult task. First of all, due to clock drift, periods are

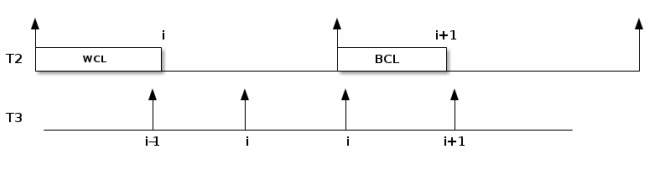


Fig. 2: An example sequence of frame repetitions.

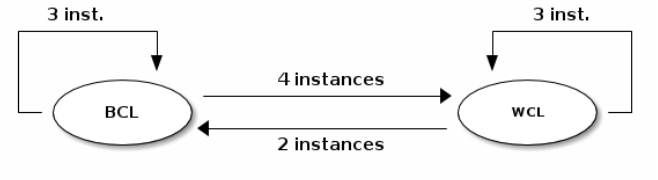


Fig. 3: The Sequence Automaton.

fixed but they can vary in a small interval, so the hyperperiod can become very large. Also, Task T3 contributes to add a substantial amount of complexity: in fact, the period P3 is approximately one third of P2. Therefore, T3 will read the same frame more than once. However, the exact number of times that a frame is read (which we call *frame repetition*) depends on the periods and on the relative initial offset of T2 and T3. Also it depends on the actual latency of T2. Consider the situation depicted in Fig. 2. Here T2 writes the frame i in the register terminating at its worst-case latency, just a little bit after Task T3 has read frame $i-1$. Then T3 reads frame i twice. Finally the response time of the second instance of T3 is equal to its best case, so frame $i+1$ is written just before task T3 can read the register. This means that Frame i is only read twice. By inverting the sequence of response times of T2, it is possible to show that one frame can be read 4 times. Finally, obviously a frame can be read 3 times.

If T3 has a period that is exactly equal to the period of T1 divided by 3, it is possible to construct periodic sequences of repetitions. We denote these periodic sequences as $< f1, f2, \ldots, f_{j-1} >$, where f_1 denoted the repetitions of frames $0, j, 2j, \ldots$, f_2 denotes the repetitions of frames $1, j+1, 2j+1, \ldots$, and so on.

Of course, repetitions can obey any pattern, even non-periodic ones. Assume that $\text{P}_3 = \text{P}_1/3$ and the offsets are as shown in Fig. 2. Then every possible sequence can be generated by the automaton in Fig. 3, that we call *Sequence Automaton*. Depending on whether the response time of T2 is equal to the best case or to the worst-case (non deterministic choice), we generate the next repetition.

Therefore, our strategy for reducing the complexity of the model to something analysable is the following:

- We fix periods $\text{P}_1 = \text{P}_2 = \text{P}_1^{\min}$, $\text{P}_3 = \text{P}_2/3$, $\text{P}_4 = \text{P}_4^{\max}$;
- We model the sequence of frames generated by T2 with the Sequence Automaton;
- The frame index is an integer number that varies in

$\{0, 1, 2\}$. In fact, it is easy to see that it is impossible to have 2 consecutive frame losses, and in the case of Buffer34 having only one position, there is no risk of conflicting indexes.

- Buffer34 is modeled by a discrete variable last and a counter with the number of frames in the buffer. Variable last maintains the index of the last frame successfully written in the buffer.
- Task T2 is modeled by a timed automaton that periodically tries to write the current frame in the buffer. The write is successful if the buffer is not full and if the frame index is different from last, in this case both variables are updated accordingly. Otherwise the frame is dropped. Every frame is repeated the number of times specified by the Sequence Automaton. After that, the frame index is increased (modulo 3) and a new repetition is generated. If the old frame index is different from last, then the frame was not successfully written, and we signal the occurrence of a frame drop.
- The distance between two frame drops is measured by an automaton *Observer* that uses an appropriate clock from measuring this distance.

We do not report here the full model for space constraints: the interested reader can find it at [XP].

Even fixing the periods, the model is still non-deterministic due to the Sequence Automaton, and unfortunately after 18 hours of running time IMITATOR had explored only 10 seconds of executions, generating a large number of states and never encountering two frame losses. Therefore we decided to further simplify the model by substituting the Sequence Automaton with a deterministic automaton that generates only periodic sequences. In this way the model becomes fully deterministic and can be analyzed in a relatively small time. Unfortunately we could not try all possible periodic sequences, nor we could try different values of the periods, therefore the computed values may not correspond to the minimal distance. The results are shown in Table II.

Buffer	Sequence	limit	runtime	Dist.	Frames
full	$< 2343 >$	60,000	660	66.7133	1668
full	$< 24 >$	100,000	2,005	66.7133	1668
empty	$< 2343 >$	60,000	639.54	-	-
empty	$< 24 >$	150,000	26,808.23	199,980	5000

TABLE II: Results of deterministic model for problem 1B

In the first column we report the initial state of Buffer34 and in the second column the periodic sequence. Since the model does not converge (i.e. it cannot explore all possible states in a reasonable time), we stop the model after a certain number of steps, reported in the third column. In the fourth column we report the running time of the tool in seconds. Finally, the last two column show the minimum observed distance, in seconds and in frames.

In the experiments relative to the first two rows in the table, we force an initial full buffer. However, we do not know if this specific initial state can actually happen in the model. So we cannot conclude that 1668 frames is an upper bound to the minimum loss distance. On the contrary, the last two lines report the results for an empty initial buffer (which is more realistic), and hence we conclude that 5000 is a realistic upper bound to the minimum loss distance. Observe that this value is very close to the average frame loss value computed at the beginning of this section.

We did not run any experiment for the case of a Buffer34 with 3 position, but everything leads to think that the results produced by a deterministic model would be very similar.

VII. SOLUTION TO CHALLENGE 2

A. Schedulability analysis

For Challenge 2A, when it goes to maximum end-to-end latency, we could employ the critical instant property [LL73], [LSAF14] to compute it. That is, the maximum latency happens such that

- T5, T6 and T7 always execute by their worst-case.
- T6 starts to execute coincidently with a release of T2.

We assume the period of T2 is exactly equal to $\mathrm{P}_2 = 40$ms; Then, we obtain the maximum latency 74ms for $J6 = 0$ and $74 + 20 = 94$ms for $J6 = 20$ms. Notice that they are both inferior to P_6, so there is no possibility that a new instance of T6 starts before the last instance of T7 completes. Possible variations in the period of T2 do not have any impact on the estimated end-to-end latency.

As for the minimum latency, a first intuition is that T5, T6 and T7 should execute by their best-case, so we compute $4 + 4 + 9 + 4 + 11 = 32$ms.

- If the initial offset between T2 and T6 is larger than 32ms, then the minimum latency is indeed 32ms.
- Otherwise, T2 preempts the execution of T5-T7 at least once, and the minimum latency would be $32+17 = 49$ms.

Therefore, we conclude that the latency of the chain T5-T7 is within $[49, 94]$.

In Challenge 2B, a mutually exclusive resource is shared between T5 and T2. In this case the minimum and maximum latencies do not change, whereas task T2 could suffer for a delay of 2 in accessing the shared resource, so its response time will vary in $[17, 19]$.

If we have the freedom to manipulate priorities we can further reduce the end-to-end latency. T2 is part of the chain for video frame processing, and to avoid interfering the verification result in Challenge 1, we still assume that T2 has higher priority than T5, T6 and T7. We can re-arrange the priorities among T5, T6 and T7 as T5 $>$ T7 $>$ T6, and the worst-case latency becomes 69ms for $J6 = 0$ and 89ms for $J6 = 20$ms. It is not difficult to see that this is the optimum priority assignment by enumeration.

B. PTA model for Challenge 2

We modeled the system following the scheduling framework for PTA proposed in [LSAF14]. We do not report here the full model for lack of space, it can be found at [XP]. We just summarize the main points:

- The scheduler is modeled as a separate automaton, generated automatically from the priority ordering, and it interacts with the task automata using synchronization.
- Variable execution times are modeled by non-deterministic choices.
- Also, initial task offsets are modeled as non-deterministic choices: the period of T6 is $p_6 \in [0, \mathrm{P}_6 - 32]$, whereas the period of T2 is $p_2 \in [0, \mathrm{P}_2 - 17]$.
- An observer automaton measures the end-to-end latency with a clock. Maximum and minimum latencies are modeled as parameters, exactly in the same way as the model developed for Challenge 1A.
- Due to lack of time, we modeled only Challenge 2A, however an extension of the model to Challenge 2B is straightforward.

The model converges after 2 seconds to the same values as the ones computed by the schedulability analysis.This confirms the results and confirms the fact that this second challenge is much easier to solve than Challenge 1.

One potential benefit of using IMITATOR is that we can easily investigate the robustness of the solution with respect to variations in task worst-case execution times or periods by choosing them as parameters in the PTA model.

In the end, we report the results for Challenge 2 in Table III. As we discussed, the existence of a shared resource between T5 and T6 does not affect the latency for the chain T5-T7.

	min latency	max latency
$J6 = 0$ ms	49 ms	74 ms
$J6 = 20$ ms	49 ms	94 ms

(a) The min/max latency

T2>T5>T7>T6	
	max latency
$J6 = 0$ ms	69 ms
$J6 = 20$ ms	89 ms

(b) The optimal priority assignment

TABLE III: Results for Challenge 2

VIII. Conclusion

We proposed solutions to the FMTV challenge both using empirical analysis and formal methods based on PTA. Thanks to their expressiveness, PTA are especially convenient for modeling systems that contain some unknown but constant configurations. Challenges 1A and 1B were the most difficult, so we only provided upper bounds on results for them. The high complexity is mainly due to the large hyperperiods, therefore a complete model is intractable. We spent about one week for studying the problem and provide a model that could converge in a decent time.

Conversely, Challenge 2 is a scheduling problem, whose solution can be obtained quickly using schedulability analysis methods. The solution has been validated by a formal model based on PTA that took half a day to be built using the framework developed in [LSAF14], and only a few seconds to converge.

An advantage of using parametric model checking for challenge 1A is that we do not get only a min/max value for the end-to-end latency, but a list of all possible values according to our model. Furthermore, we also get the exact values of the periods that lead to the smallest and largest end-to-end latency, which can be of interest to better understand the model.

IMITATOR is the software tool we rely on for performing parametric analysis. IMITATOR turned out to be particularly well-suited for the analysis of such systems with period constants but with some (unknown) imprecision. We also note that we used the current version of IMITATOR, with no dedicated improvement for this challenge (with the exception of a small Python script to parse the result). Hence, this leaves space for improvements dedicated to these case studies, *i.e.,* specific heuristics or simulation relations for ad-hoc reductions of the state space.

References

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AFKS12] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer, 2012.

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *STOC*, pages 592–601. ACM, 1993.

[AS13] Étienne André and Romain Soulat. *The Inverse Method*. FOCUS Series in Computer Engineering and Information Technology. ISTE Ltd and John Wiley & Sons Inc., 2013.

[BBLS15] Nikola Beneš, Peter Bezděk, Kim G. Larsen, and Jiří Srba. Language emptiness of continuous-time parametric timed automata. In *ICALP*, July 2015. To appear.

[BL09] Laura Bozzelli and Salvatore La Torre. Decision problems for lower/upper bound parametric timed automata. *Formal Methods in System Design*, 35(2):121–151, 2009.

[BO14] Daniel Bundala and Joël Ouaknine. Advances in parametric real-time reasoning. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *MFCS*, volume 8634 of *Lecture Notes in Computer Science*, pages 123–134. Springer, 2014.

[HRSV02] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53:183–220, 2002.

[JLR15] Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. Integer parameter synthesis for timed automata. *IEEE Transactions on Software Engineering*, 2015. To appear.

[LL73] C. L. Liu and James Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[LSAF14] Giuseppe Lipari, Youcheng Sun, Étienne André, and Laurent Fribourg. Toward parametric timed interfaces for real-time components. In *SynCoP*, volume 145 of *Electronic Proceedings in Theoretical Computer Science*, pages 49–64, 2014.

[SLDP09] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. PAT: Towards flexible verification under fairness. In *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer, 2009.

[XP] Experiments. http://www.imitator.fr/static/FMTV15.