Efficient End-to-End Latency Distribution Analysis for Probabilistic Time-Triggered Systems

M. Westmijze, M.J.G. Bekooij and G.J.M. Smit
University of Twente, Department of EEMCS
P.O. Box 217, 7500 AE Enschede, The Netherlands
{m.westmijze, m.j.g.bekooij, g.j.m.smit}@utwente.nl

***Abstract*—Medical image processing systems are typically implemented with a number of independent subsystems that have time-triggered interfaces. A critical design parameter for these systems is the latency between the instant that an image is captured and the instant that the enhanced image is displayed to the physician. Computation of the end-to-end latency distribution with existing techniques is often impractical due to the extremely large number of states that need to be considered.**

In this paper we introduce the probabilistic time-triggered system (PTTS) model. With this model the end-to-end latency distribution can be computed efficiently. Furthermore, we present a bound on the time-complexity of our analysis algorithm and a technique that reduces the state-space at the cost of accuracy.

We demonstrate the applicability of the presented analysis technique by showing that several system configurations of an X-Ray application can be quickly explored. This exploration reveals the parameters which have a significant effect on the end-to-end latency distribution.

I. Introduction

With an interventional X-Ray system, a physician makes use of images captured with an X-Ray imaging device to perform delicate medical procedures with a catheter inside a patient. The only visual feedback is provided by the images captured by the X-Ray device. It is therefore desirable that the latency between capturing and displaying an image is low enough, i.e. < 200 ms, to provide sufficient eye-hand coordination. Furthermore, the variation of the latency, which is called jitter, must be sufficiently low such that the physician experiences a constant delay, which improves the eye-hand coordination and prevents fatigue.

Traditional hard real-time techniques can guarantee that such latency constraints are never violated. However, the systems that we consider do allow rare violations of the specified latency requirement as long as the probability of such a violation is low and known at design time. This probability cannot be computed with hard real-time analysis techniques. Furthermore, treating these systems as hard real-time systems would require the use of special purpose hardware instead of off-the-shelf cache coherent multiprocessor systems, which could increase the design cost significantly. Therefore, it can be desirable to use off-the-shelf hardware in combination with a probabilistic model and corresponding analysis techniques.

Medical image processing systems such as interventional X-Ray systems are nowadays often composed of a number of independent off-the-shelf subsystems that have time-triggered interfaces. Examples of these subsystems are the image sensor, the image enhancement general purpose computer, a graphical processor unit (GPU) for image composition, and the video wall for the combination of several displays. The latency introduced by the individual subsystems can vary significantly and is measured. Also the end-to-end latency distribution of the chain of subsystems is measured. Such a measurement based approach cannot guarantee that rare-events are detected and that thus a complete latency distribution is observed.

Besides measuring, the end-to-end latency distribution of a chain of subsystems can be computed using the timed probabilistic labeled transition system (TPLTS) model [6]. However, the execution time of an analysis algorithm that computes a complete end-to-end latency distribution can become prohibitively high as a result of the large number of states that need to be considered. Therefore, a simulation based approach has been proposed [9] which has, equivalent to the measurement based approach, as disadvantage that rare-events might go undetected and that an incomplete latency distribution is observed.

In this paper we introduce the probabilistic time-triggered system (PTTS) model together with an analysis algorithm for the efficient derivation of the complete end-to-end latency distribution of time-triggered systems that are composed of time-triggered subsystems. The PTTS model is suitable for systems that are composed of a chain of time-triggered subsystems that are connected by buffers with one location. These buffers behave like registers. Writes in these buffers are destructive and reads non-destructive. Furthermore, we will derive an expression for the time-complexity of our analysis algorithm. This expression indicates in which case the execution time of our analysis algorithm will become unacceptably high. For these cases we propose a technique that reduces the state-space and the execution time at the cost of accuracy by adapting parameters in the PTTS model.

The organization of this paper is as follows. In Section II we discuss alternative analysis approaches for time-triggered systems. In Section III we define the PTTS model. In the subsequent section, we present a didactic example that provides the intuition behind our technique. A detailed description of our analysis algorithm is given in Section V. A technique to reduce the execution time of our analysis algorithm is discussed in Section VI. We demonstrate the applicability of the PTTS model for an interventional X-Ray system in Section VII. Finally we state the conclusion in Section VIII.

978-1-4799-3770-7/14/$31.00 ©2014 IEEE

II. Related Work

Time-triggered systems, in which tasks are scheduled strictly periodically, have been studied extensively by Kopetz [3], [4]. In his approach the schedule of the tasks is computed at design-time using the worst case execution times (WCETs) of tasks. The computed schedule should respect the precedence constraints between the tasks, as well as the throughput and end-to-end latency constraint of the task graph. Satisfaction of the precedence constraints guarantees functional deterministic behavior of the task graph, i.e. the results computed by the tasks are independent of the execution time of the tasks as long as the execution times are not larger than the WCETs assumed at design time. The most important difference with the work presented in this paper is that we do not make use of WCETs but make use of execution-time profiles (ETPs) [1] to characterize the execution time of the tasks. The ETPs are probabilistic characterizations of the execution times by means of probability mass functions (PMFs). As a result we do not want to compute a periodic schedule at design-time, because it will be overly pessimistic. Another consequence is that the execution of the systems that we consider in this paper is not functionally deterministic, because we allow that data is overwritten before it is read, depending on the execution times.

A deterministic time triggered model is proposed by Henzinger [2], which relies on a global clock and WCETs in order to compute permissible schedules. Our model does not use WCETs and we derive the distribution of the end-to-end latency and the probability of a data-race instead of giving a guarantee that the system is functionally deterministic and adheres to hard real-time constraints.

The TPLTS [6] model is suitable for the analysis of probabilistic time-triggered systems. This model can be automatically derived from a description of a system in the POOSL language [10]. A TPLTS model can be converted into a Markov chain for which exact analysis techniques exist [5], [7]. It should be noted that the obtained results are in general only valid for infinitely long intervals of time. A well known problem is that the number of states in these Markov chains is often so large that exact analysis is impractical. Therefore, approximation techniques based on simulation have been proposed that do not consider all states [9]. In these proposals the TPLTS model is simulated until estimators indicate that it is likely that a sufficiently large part of the state space has been considered and sufficiently accurate results are obtained. However, usually no indication is provided whether the obtained results are an over-approximation or an under-approximation of the throughput and the latency nor, is there a bound provided on the accuracy of the obtained analysis results. An important difference is that our PTTS model is only suitable for the analysis of a subset of the systems that can be analyzed with the TPLTS model, i.e. systems that can be modeled as a chain of time-triggered subsystems. This restriction enables the calculation of the arrival times of the data for a subsystem without considering the interaction with all the other subsystems. As a result, the state-space is reduced and the time-complexity of the analysis algorithm is decreased. Furthermore, we show that for the considered class of time-triggered systems the probability of a data-race can be defined for a bounded interval of time.

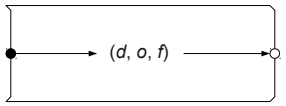

Fig. 1: A PTTS component

Another model is scenario aware dataflow (SADF) [8] which extends synchronous dataflow (SDF) with stochastic scenarios in order to model the dynamic behavior and variable execution times of applications. However, SADF lacks the expressiveness to model the time-triggered interfaces. The production and consumption rates of actors in the SADF model should be consistent. In our model the production rates of different subsystems may differ and our analysis algorithm is able to quantify the amount of data duplication or destruction.

It should be noted that the mentioned models (TPLTS, SADF and SDF use graphs to model the topology of the application. The PTTS model is restricted to a chain topology where the application consists of a chain of time-triggered subsystems. Although, for the purposes of a latency analysis, it is possible to extract such a chain from a graph topology, it falls outside of the scope of this paper and is therefore not elaborated.

III. PTTS Model Definition

In this section we define the PTTS model. The PTTS model consists of components with one input port and one output port. Such a component is depicted in Figure 1.

The input port of a component is a buffer with one location, which is read out time-triggered. More precisely the instant at which the component reads a container with data is determined by a timer and not by the arrival time of data containers at the input port. The buffer at the input port has non-destructive read and destructive write semantics. Therefore, if no new container has arrived on the input port at the sampling moment, the previously arrived container is read again from the input port. If multiple containers arrive at the input port between two sampling instants, then the container is processed that arrived last.

Each component is characterized by three parameters d, o, and f. Here d defines the period at which the time-triggered input port triggers, o is the initial offset when the input port is triggered for the first time, and f is the ETP. The ETP is a PMF that characterizes the latency of a component, i.e. the interval between the triggering of the input port of the component and that data is produced at its output port. The PMF is defined at discrete points. The use of a PMF instead of a probability density function (PDF) simplifies the explanation of our end-to-end latency distribution calculation algorithm, whereas the algorithm would remain conceptually similar if PDFs would be used instead. Multiple containers can be under processing at the same point in time, in other words, the model allows auto-concurrency. That a later triggering of the component can

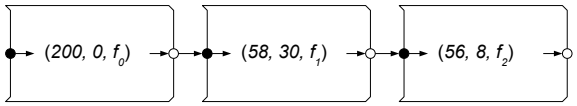

Fig. 2: A chain of PTTS components

result in an earlier production than an earlier triggering, i.e. out of order production, is not problematic because we only derive the temporal behavior and not the functional behavior with the PTTS model. The ETP can be obtained by measurement of the latency of a component in isolation.

Components in the PTTS model can correspond, for example, with a CCD in a video system that is triggered at 24 frames per second (FPS), or a GPU that transfers a new image from video memory to a monitor at 60 FPS. It is also possible that a component corresponds to a task graph executing on multiple processors, but the PTTS model abstracts from this.

PTTS components are connected in a chain. The output of all but the last component is connected to its successor, e.g. C_0 is connected to C_1. A chain of N components is denoted as $\langle C_0, C_1, \ldots, C_{N-1}\rangle$. An PTTS chain is described as a list of tuples, where each tuple (d_i, o_i, f_i) specifies the parameters of the component C_i. An example of a PTTS chain is shown in Figure 2.

The analysis algorithm presented in Section V derives the probability distribution that characterizes the end-to-end latency of a chain of components. With the same algorithm we determine the probability that a container is overwritten before it is read, i.e. the probability that data loss occurs. The end-to-end latency is defined as the difference between the production moment of a container with data by component C_{N-1} and the arrival moment of the corresponding container at the input of component C_0. This corresponding container resulted in the production of the container by C_{N-1}.

IV. Basic Idea

In this section we present the basic idea behind our end-to-end latency distribution analysis algorithm using an example. This analysis algorithm will be presented in detail in Section V.

As an example we use the PTTS chain shown in Figure 2. The model in Figure 2 can be described with the chain $\langle(200\,\text{ms}, 0\,\text{ms}, f_0), (58\,\text{ms}, 30\,\text{ms}, f_1), (56\,\text{ms}, 8\,\text{ms}, f_2)\rangle$. In this example the ETPs of the latency parameters f_0, f_1, and f_2, in which $t \in \mathbb{N}_0$, are given in Eq. (1), Eq. (2) and Eq. (3) respectively. These functions define a discrete homogeneous PMF over the given intervals and describe the ETP.

$$f_0(t) = \begin{cases} \frac{1}{100}, & t \in [50, 149] \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

$$f_1(t) = \begin{cases} \frac{1}{30}, & t \in [25, 54] \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

$$f_2(t) = \begin{cases} 1, & t = 5 \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

A system described by the PTTS model has a hyper-period which is defined as the time after which the same pattern of triggerings will occur. Each triggering of a component in a hyper-period has a unique set of offsets to the triggerings of other components. For example, the offset between the first triggering of C_0 and the subsequent triggering of C_1 is 30 ms, but the offset between the second triggering of C_0 at $t = 200$ ms and the subsequent triggering of C_1 at $t = 204$ ms is 4 ms. The same triggering pattern is repeated in every subsequent hyper-period. Given the periods of the components we can derive the hyper-period of the system by calculating the least common multiple (LCM) of all the periods of the components with Eq. (4).

$$h = \operatorname*{lcm}_{0 \le i < N}(d_i) \quad (4)$$

The hyper-period for the PTTSs model in Figure 2 is $\text{LCM}(200\,\text{ms}, 58\,\text{ms}, 56\,\text{ms}) = 40\,600$ ms. As a consequence, if there is a triggering of a component at time $t = 0$ ms then there are also triggerings with the same end-to-end latency distribution of the same component at time $t = o_0 + k \cdot 40\,600$ ms with $k \in \mathbb{N}_0$ and where o_0 is the initial offset of component C_0. As a consequence it is sufficient to consider the interval of a single hyper-period during analysis. A part of the hyper-period of this example is depicted in Figure 3.

One way to derive the end-to-end latency is to follow a container from the input of the PTTS model to the output of the PTTS model. We follow a container in the state space by stepwise expanding the state where the container is produced and follow it as it passes through the PTTS chain. For example, assume that the input port of C_0 is triggered at 0 ms. From the ETP f_0 we conclude that the latency introduced by component C_0 is between 50 ms and 149 ms. Suppose now that for this particular triggering the latency introduced by C_0 is 120 ms, then a container is produced by component C_0 at $t = 120$ ms. This container will stay in the input buffer of component C_1 till the input port of this component is triggered. Component C_1 has a period of 58 ms and the first triggering of its input port happens at $t = 30$ ms because its offset is 30 ms. Therefore, after C_0 has produced a container at 120 ms, the third triggering of C_1 will consume the container because $\min\{x \in \mathbb{N}_0 | 30 + x \cdot 58 \ge 120\text{ms}\} = 2$. In a similar manner we can find when C_2 consumes the containers produced by C_1 and at which moments in time C_2 produces its output containers. This way we can simulate what the end-to-end latency is for the triggering of C_0 that originates at 0 ms and for one particular latency per component.

In order to obtain an impression of the end-to-end latency distribution by means of simulation, we would have to simulate all possible triggerings of all components for many possible latencies of the individual components. A problematic aspect of such an approach is that such a simulation takes long, because many cases needs to be considered. Furthermore, no bounds are defined on how accurate the end-to-end latency distribution obtained by simulation approximates the latency distribution. Therefore, it can be preferable to derive the end-to-end latency distribution by means of an analytical approach instead of simulation. Such an analytical approach can exploit the fact that not all possible latencies of the components need to be considered but that an interval of possible latencies results in the same end-to-end latency.

The sampling moments of each component C_i is defined by its period and offset, d_i and o_i respectively. Given these sampling moments and the ETP of a component we can derive the intervals in which containers are produced by a component. These intervals are then further divided in subintervals based on the sampling moments of a consuming component. For each of these subintervals we can derive the probability that a container is produced. From this we can derive for combinations of triggering moments of two subsequent components in the PTTS model the probability that a specific triggering of the component that produces the container is sampled by a specific triggering of the component that consumes the container.

In Figure 3 the triggerings of the components are indicated by arrows ($\rightarrow$). The same figure shows with rectangles in which interval of time a container is produced by a specific triggering. From these rectangles we can derive which triggering of a subsequent component in the chain is potentially consuming the container. With each fraction of a rectangle, a probability is associated that defines the probability that a container is produced in the corresponding interval of time. Given these probabilities, we can derive the probability that a container produced by a triggering of a component, is consumed by a specific triggering of a consecutive component.

More precisely, the probabilities are determined for our PTTS example as follows. When C_0 activates at $0\,\text{ms}$ we know from its ETP that the latency of this component lies between $50\,\text{ms}$ and $149\,\text{ms}$. Given this information we can determine which triggerings of C_1 might sample the output of the first triggering of C_0. From the trace in Figure 3 we can conclude that the triggerings of C_1 at $88\,\text{ms}$, $146\,\text{ms}$, and $204\,\text{ms}$ might sample the output of the first triggering of C_0. Furthermore, we can also calculate for each of these triggerings the probability that a particular triggering of C_1 will sample the output. This can be achieved by computing the sum of probabilities of C_0 for the interval that a specific triggering a C_1 consumes a container and that is produced by a specific triggering of C_0. For example: the triggering of C_1 at $146\,\text{ms}$ samples the output that is produced by C_0, triggered at 0 ms and which produces it output between $89\,\text{ms}$ and $146\,\text{ms}$. The probability that this triggering of C_1 consumes the container that is produced by C_0 is therefore: $\sum_{t=89}^{146} f_0(t) = \frac{58}{100}$.

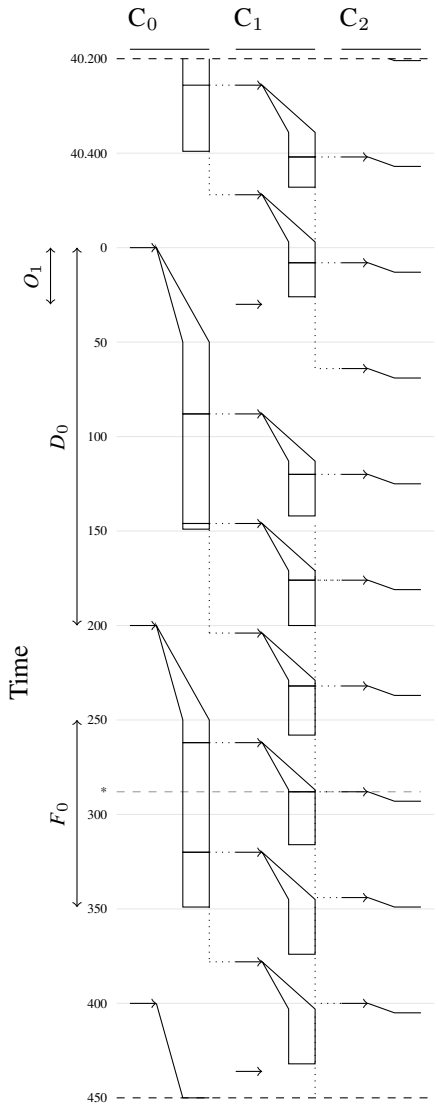

Fig. 3: Part of the trace in one hyper-period of the PTTS in fig. 2

When we have calculated the probabilities for this component we have to do the same for the subsequent component in the chain. In this example we therefore perform three probability calculations by means of summation, one for each of the triggerings at $88\,\text{ms}$, $146\,\text{ms}$ and $204\,\text{ms}$. This analysis is repeated for each component in the chain till we reach the end of the chain. Furthermore, these steps are repeated for all triggering of C_0 in the hyper-period.

The information in the trace of Figure 3 together with information about the probabilities is stored in a so-called latency tree which is shown in Figure 4. Each node in the tree represents a triggering instant in one hyper-period of a component in the PTTS. The numbers in the node represent the index of the component in the chain and the triggering instant respectively. The value on the edges represent the probability that the two triggerings of different components are related, i.e.

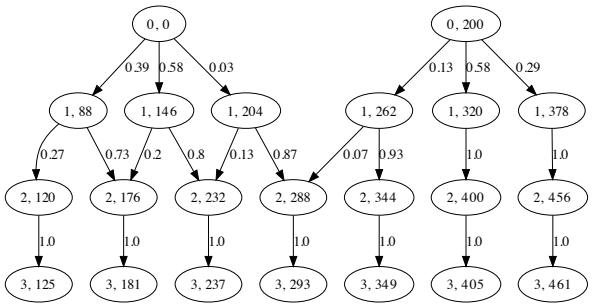

Fig. 4: Latency tree that results from the first two triggerings of C_0

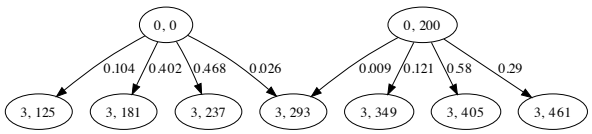

Fig. 5: Latency tree of Figure 4 after being flattened

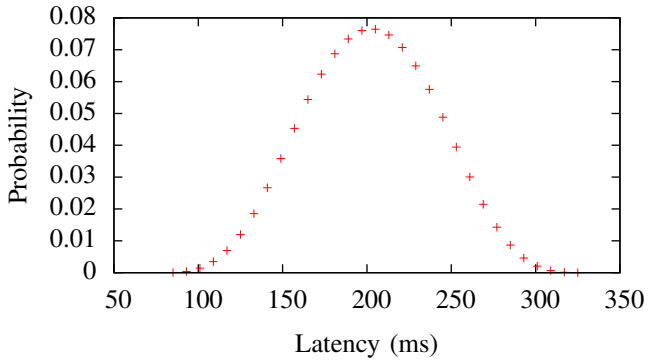

Fig. 6: End-to-end latency distribution of the PTTS model in Figure 2 using f_0

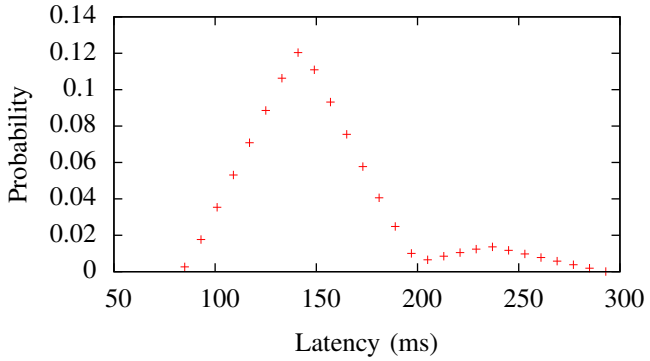

Fig. 7: End-to-end latency distribution of the PTTS model in Figure 2 using $f_0^{'}$

the triggering of a component that consumes the container that is produced as a result of the triggering of the other components. Lastly, the bottom leaves are a dummy sink component that represent the end of processing of the last component. For example, the latency trees in Figure 4 are the results of the latency analysis of triggerings of C_0 at 0 ms and 200 ms.

The end-to-end latency distribution is derived from the latency tree. The end-to-end latency distribution is found by following all paths from the root till the leaves of the tree and multiplying the probabilities along those path. The probabilities obtained after multiplication are put into a flattened version of the latency tree. Flattening will be elaborated in Section V. For example, the flattened latency trees for the first two triggerings of C_0 are shown in Figure 5. Optionally, we could aggregate all paths in the tree in order to obtain the average end-to-end latency distribution of the complete system. Figure 6 shows the end-to-end latency distribution for the PTTS model in Figure 2.

The end-to-end latency distribution of the example, see Figure 6, has a Gaussian shape. That the end-to-end latency distribution can also have another shape is demonstrated by replacing the distribution ETP f_0 by the distribution ETP $f_0^{'}$ which is defined in Eq. (5). The resulting end-to-end latency distribution is shown in Figure 7.

$$f_0^{'}(t) = \begin{cases} \frac{1}{10}, & t \in [50, 58] \\ \frac{1}{10}, & t = 149 \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

A. Container Loss

The latency trees can also be used to derive the probability that a container is overwritten before it is read. This corresponds in an X-Ray system with an image that is captured by the X-Ray image sensor but which is not displayed. Because this can result in hiccups in the video sequence, which is undesirable and may only occur rarely.

That a container is overwritten before it is read can also occur in our running example. When we examine the latency trees for the triggerings at $t = 0$ and $t = 200$, we notice that they share two nodes with each other. The shared nodes represent the triggering of C_2 at $t = 288$ and the triggering of the last output node at $t = 293$. This means that when the triggering at $t = 0$ is slow and generates the path in the latency tree $\langle 0, 204, 288, 293\rangle$ and the subsequent triggering at $t = 200$ is fast and generates the path $\langle 200, 262, 288, 293\rangle$ that then the container that is a result of the triggering at $t = 0$ is overwritten before it is read by C_2, i.e. the data in the container is lost. This behavior can also be concluded from the trace in Figure 3 where the data loss might occur at the dashed gray line at $t = 288$.

For the latency tree we can also calculate the probability that a container is overwritten before it is read. This probability in our running example is equal to the probability that the

in the previous paragraph mentioned slow and fast trace happen after each other divided by the number of triggerings of C_0 in the hyper-period. This probability is equal to $\frac{(0.03 \cdot 0.87) \cdot (0.13 \cdot 0.07)}{203} = 1.17 \cdot 10^{-6}$. Because in this PTTS example the triggerings at $t = 0$ and $t = 200$ are the only triggerings that result in a potentially container loss, we conclude that $1.17 \cdot 10^{-6}$ is the probability that the data in a container is lost during a hyper-period of the system.

V. The Analysis Algorithm

In this section we present the analysis algorithm for the derivation of the end-to-end latency distribution based on the principles discussed in the previous section. First we will define an algorithm that constructs a latency tree, and then flattens this latency tree to retrieve the distribution of the latency between the first and the last component in the chain.

The latency tree is defined as follows. The tuple (i, t, S) denotes a node in the latency tree, where C_i represents the i^{th} component in the chain, t represents the instant at which component i is triggered and S the set of tuples that represents the children of this node. The tuple $s \in S$ that represents a child of a latency tree node is defined by the tuple (p, y), where p is the probability that component C_{i+1} consumes the output of C_i, and y is a recursive tuple $(i + 1, t', S')$ that represents the triggering of the subsequent component C_{i+1} that consumes a container from C_i at t'.

Before we present the derivation of the latency tree with recursive functions, we first define some helper functions.

Let the function $A(i)$ define the set of points in time that the i^{th} component in the PTTS model is triggered:

$$A(i) = \{k \mid n \in \mathbb{N}_0 \wedge k = n \cdot d_i + o_i\} \tag{6}$$

Let $u(i, t_a, t_b)$ define the function that calculates the probability that the triggering at t_a of component C_{i-1} produces the token that is consumed by C_i, triggered at t_b:

$$u(i, t_a, t_b) = \sum_{t=t_b-d_i+1}^{t_b} f_{i-1}(t - t_a) \tag{7}$$

It is sufficient to sum the discrete probabilities from $t_b - d_i + 1$ because at $t_b - d_i$ the previous triggering of C_i takes place which consumes containers from C_{i-1}.

The function Q defined in Eq. (8) computes a set of tuples, where each tuple is defined as (p, y) with y being the recursive tuple $(i + 1, t, S)$. Each tuple corresponds to a triggering of component C_{i+1} for which there is a probability larger than 0 that it samples a container produced by component C_i. This function accepts two parameters, i and t where i is the index of the component from which we want to derive the latency tree. The other parameter t represents the instant at which C_i is triggered. The function Q uses the function Z, which is defined in Eq. (9), that will recursively generate the whole latency tree.

$$Q(i, t) = \{(u(i + 1, t, a), (i + 1, a, Z(i + 1, a)) \mid a \in A(i + 1) \wedge u(i + 1, t, a) > 0\} \tag{8}$$

The function Z is defined as follows:

$$Z(i, t) = \begin{cases} W(i, t) & \text{if } i + 1 = N \\ Q(i, t) & \text{if } i + 1 < N \end{cases} \tag{9}$$

The function Z calls the function Q, that derives the time-triggered parts of the end-to-end latency distribution till the end of the chain is reached and then the function W is called to add the part of the latency distribution that is not consumed by a consecutive timed-triggered component. The function W defined in Eq. (10) returns a set of tuples $(p, (i, t, S))$ for the last component in the chain of the PTTS model and adds the latency introduced by this component. It accepts the parameters i and s that represent the index of the component and the instant the component is triggered. The function W is defined as follows:

$$W(i, t) = \{(f_i(n), (i, t + n, \varnothing)) \mid n \in \mathbb{N}_0 \wedge f_i(n) > 0\} \tag{10}$$

Given the hyper-period h, the latency tree (or more precisely the forest of latency trees) is calculated with:

$$F = \{(0, a, Z(0, a)) \mid a \in A(0) \wedge a < h\} \tag{11}$$

The end-to-end latency distribution can be derived from a latency tree by flattening. The flattening step removes all the inner nodes in a latency tree and directly connects the root of a tree with its leaves. The probabilities on the path from root to the leaves are multiplied in order to derive the end-to-end probability. We do not directly derive the flattened tree because we need the intermediate nodes to detect data loss when data is overwritten before it is consumed.

In some cases there will be several leafs in a latency tree that represent the same triggering of the last component. This might happen because two components might have an ETP that is larger than the interval between triggerings of the subsequent component. In our example we see that there can be duplicate leafs in Figure 4 when the tree is flattened. The leafs with $t = 181$ and $t = 237$ will have duplicates when the tree is flattened and should be aggregated. For each tree that has multiple leafs that correspond with the same triggering we aggregate the leafs and compute its probability by summing the probabilities of the original leafs that share the same triggering instant.

A. Time Complexity

In this subsection we derive an expression for the time-complexity of our analysis algorithm. This expression will be used in Section VI to estimate the effect of a modification of the periods on the execution time of our analysis algorithm.

The time-complexity of our analysis algorithm is based on the number of latency trees that have to be derived and the time it takes to derive a latency tree. The number of latency trees that have to be derived is equal to the hyper-period divided by the period of the first component in the PTTS model. The time it takes to derive a latency tree is proportional to the depth, i.e. the number of components, and the branching factor in the latency tree, i.e. how many siblings each node has in the tree.

The branching factor depends on the width of the ETPs, i.e. the interval between the best-case and worst-case latency, and the period of the subsequent component that reads the container that is produced after those ETPs. For example, the width of the ETP that is associated with component C_0, from the example in Section IV, is $149\,\text{ms} - 50\,\text{ms} + 1\,\text{ms} = 100\,\text{ms}$. Component C_1 consumes containers from C_0 and has a period of $58\,\text{ms}$. This implies that each node from C_0 has between $\lfloor \frac{100}{58} \rfloor = 2$ and $\lceil \frac{100}{58} \rceil = 3$ siblings. In Figure 4 we can see that the first two triggerings of C_0 each have three siblings, and the branching factor for those two triggerings is therefore three. We therefore conclude that the time-complexity of our analysis algorithm can be expressed in the terms of the maximum number of required operations with Eq. (14).

$$\check{\tau}(i) = \min\{t | f_i(t) > 0\} \tag{12}$$

$$\hat{\tau}(i) = \max\{t | f_i(t) > 0\} \tag{13}$$

$$\frac{1}{d_0} \cdot \operatorname*{lcm}_{0 \le i < N}(d_i) \cdot \prod_{i=0}^{N-2} \left\lceil \frac{\hat{\tau}(i) - \check{\tau}(i) + 1}{d_{i+1}} \right\rceil \tag{14}$$

Configuration	Components		
	C_0	C_1	C_2
a	(66.0, 0, f_0)	(16.0, 0, f_1)	(16.0, 0, f_2)
b	(66.6, 0, f_0)	(16.6, 0, f_1)	(16.7, 0, f_2)
c	(66.66, 0, f_0)	(16.66, 0, f_1)	(16.67, 0, f_2)
d	(66.666, 0, f_0)	(16.666, 0, f_1)	(16.667, 0, f_2)

TABLE I: Configurations used to demonstrate the effects of a reduction of the hyper-period

Configuration	Latency trees	Analysis time (ms)
a	136	36
b	13861	3041
c	1388611	297620
d	$138 \cdot 10^6$	33795549

TABLE II: Execution time of the analysis algorithm for the configurations in Table I

VI. Analysis Time Reduction

In this section we describe a technique to reduce the execution time of the analysis algorithm at the cost of accuracy. The technique is intended for the cases that the hyper-period is too long, which result in impractically long execution times.

A. Hyper-period Reduction

From Eq. (14) it follows that we can reduce the execution time of our analysis algorithm by reducing the hyper-period. In this section we examine how modifying the periods of the components affects the execution time and accuracy of our analysis algorithm.

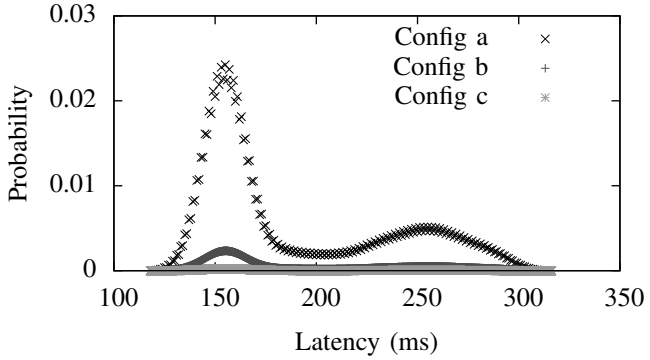

Fig. 8: End-to-end latency distribution for the configurations in Table I

The hyper-period of a PTTS can be reduced by adapting the periods of the components. The effect of such a modification on the accuracy of the analysis result is hard to predict in general. However for a number of realistic systems we have observed that the shape of the latency distribution remains intact. For example: consider a small system with three components where the parameters for the components are defined as in Table I. The configurations *a*, *b*, *c* and *d* have an increasing hyper-period, and as a result, an increasing number of latency trees has to be derived. We have run our analysis algorithm on these four configurations. In Table II the number of derived latency trees and the execution time of the analysis algorithm can be found. From this table we conclude that the execution time of configuration *d* is impractically long. Furthermore, in Figure 8 we have depicted the end-to-end latency distribution for configuration *a*, *b* and *c*. The shape of the end-to-end latency distribution is similar, but due to the decreased resolution of the PMF the amplitude of the configuration with the reduced hyper-period is higher. We also notice that, due to the dramatically decreasing hyper-period, the analysis time reduces drastically. In Figure 9 the cumulative end-to-end latency distribution is shown. From this figure it can be concluded that the cumulative latency distribution is almost the same for configuration *a*, *b* and *c*.

VII. Case Study

In this section we study the influence of parameter changes on the latency distribution of an image processing chain using the analysis techniques presented in the previous sections. Furthermore, we compare the results obtained by simulation with the exact results obtained by analysis.

The image processing chain we consider in this section consists of an X-Ray sensor (CCD), a network, an image processing subsystem, a GPU and a media wall, which also incorporates another GPU. The chain contains three time-triggered components namely the X-Ray sensor and the two

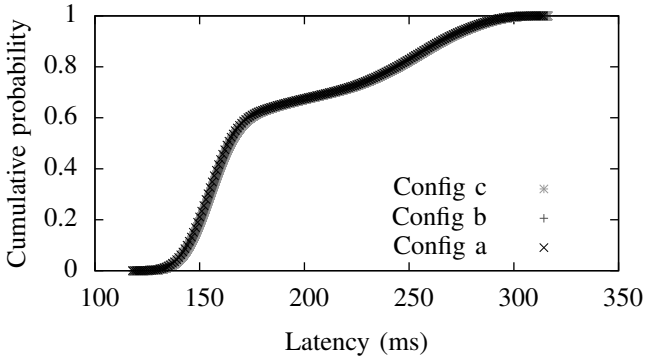

Fig. 9: End-to-end cumulative latency distribution for the configurations in Table I

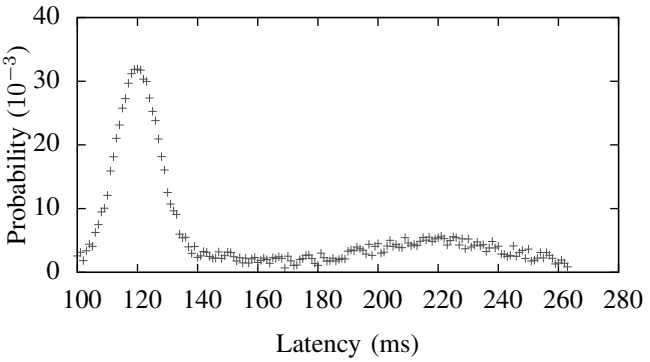

Fig. 10: Definition of the ETP for f_0 that is used in the case study

GPUs. The PTTS model is defined by the chain $\langle\ C_0, C_1, C_2\ \rangle$.

To study the effect of the parameters on the end-to-end latency distribution we consider the three system configurations of Table III. The applied ETPs can be found in Figure 10, Eq. (15) and Eq. (16).

$$f_1(t) = \begin{cases} \frac{9}{10}, & t = 11 \\ \frac{1}{10}, & t = 12 \\ 0, & t \notin [11, 12] \end{cases} \tag{15}$$

$$f_2(t) = \begin{cases} 1, & t = 8 \\ 0, & t \neq 8 \end{cases} \tag{16}$$

	Components		
Configuration	C_0	C_1	C_2
d	(66.6, 0, f_0)	(16.6, 0, f_1)	(16.7, 0, f_2)
e	(66.6, 0, f_0)	(16.6, 0, f_1)	(16.6, 0, f_2)
f	(66.4, 0, f_0)	(16.6, 0, f_1)	(16.7, 0, f_2)

TABLE III: Configurations for the PTTS model defined in Figure 2

Analysis tool	Time (ms)
Analytical	30
Simulation (100 iterations)	98
Simulation (1000 iterations)	749
Simulation (10000 iterations)	11372

TABLE IV: Analysis time

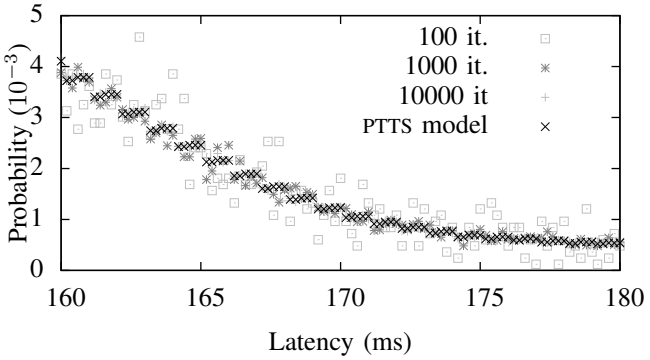

Fig. 11: Comparison of the end-to-end latency distributions for configuration *e*

We compare the latency distribution for configuration *e*, which is obtained by simulation, with the exact results. We have measured the end-to-end latency distribution by simulation for 100, 1000 and 10000 iterations. A comparison with the exact distribution is shown in Figure 11. It can be observed that the simulation results approximate the exact results accurately when a large number of iterations is simulated. The number of iterations that results in a sufficiently accurate approximation is however not known in advance. In Table IV we measured the time that our analysis algorithm took and the time that the simulations took. The table shows that the analytical approach is much faster than the simulation, even when the simulation only does 100 iterations. A comparison of the end-to-end latency distributions of the configurations *d*, *e* and *f* is shown in Figure 12. Similar end-to-end latency distributions have been obtained with the POOSL simulator SHEsim. A comparison with the execution time of an exact analysis algorithm for a POOSL model was not possible because an implementation is not yet available [6].

Also the offsets can have a significant affect on the end-to-end latency distribution. We demonstrate this for configuration *e* from Table III in which we select different offsets for C_2. The obtained analytical results are shown in Figure 13. From this figure we can conclude that the offsets have a significant influence on the end-to-end latency. Another important observation that we can make from this figure is that the end-to-end latency distribution does not increase monotonically with regard to the offset. For example, the lowest latency as well as the highest latency can occur when the offset is 80. However, when we examine the cumulative probability at 160 ms we

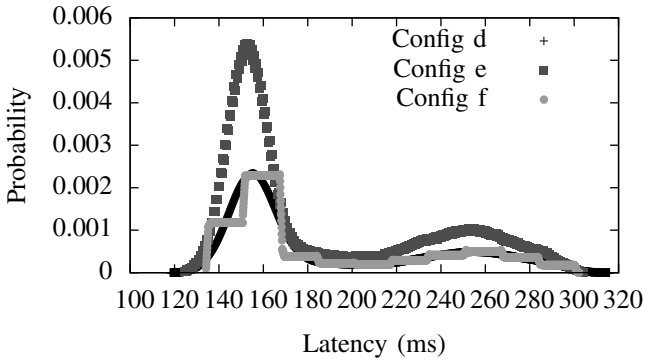

Fig. 12: End-to-end latency distribution for different configurations of the X-Ray application in the case study

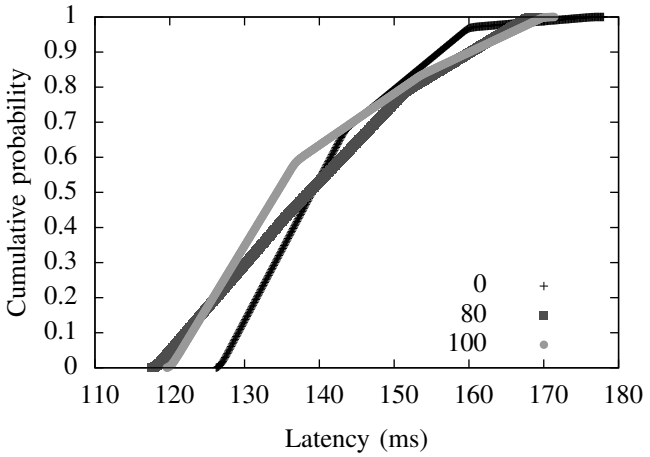

Fig. 13: Cumulative end-to-end latency distribution for configuration e with varied offset for C_1

see that the cumulative probability of the distribution with offset 80 is higher than for the other two offsets. This indicates that although the end-to-end latency can be higher when the offset is 80, also the probability that it is below 160 ms is also the highest for the three considered offsets. Due to the non-monotonic behavior it can be required that for many offsets the latency distribution needs to be computed before an offset is found that results in a latency distribution which adheres to the temporal requirements of the system.

VIII. Conclusion

In this paper we introduced the PTTS model for analyzing the latency distribution of real-time systems that consist of a chain of subsystems with time-triggered interfaces and a probabilistic delay. This type of systems is found in, for example, the medical image processing domain where the end-to-end latency distribution is an important design parameter. Furthermore, we presented an analysis algorithm for the computation of the end-to-end latency distribution. The algorithm allows the derivation of a complete end-to-end latency distribution, i.e. without overestimation or underestimation of the distribution. This guarantees that rare events such as data-races will be detected. Such a data-race can result in an image being overwritten in a buffer before it is displayed. We have shown that for the considered type of time-triggered systems we can compute the probability of a data-race in a bounded interval of time.

A useful feature of the PTTS model is that the size of the state-space can be computed, which enables the derivation of an expression for the time-complexity of our analysis algorithm. This expression indicates in which cases the execution time of our analysis algorithm will become unacceptably high. For these cases we propose a technique that reduces the state-space and the execution time of our analysis algorithm at the cost of accuracy by adapting parameters in the PTTS model.

We have compared the accuracy and execution time of our analysis approach for an X-Ray system with results obtained by simulation. The results indicate that the execution time of our analysis algorithm is lower while the computed latency distribution is exact. Furthermore, we studied the effects of parameter changes on the end-to-end latency distribution. The results show that parameter changes can have non-monotonic effects on the latency distribution and that these effects can be substantial. The presented analysis algorithm can be an attractive option for a fast exploration of different system configurations. With this exploration, system parameters can be found for which the latency distribution adheres to the temporal requirements.

References

[1] G. Bernat, A. Colin, and S. Petters. Wcet Analysis of Probabilistic Hard Real-time Systems. In *Real-Time Systems Symposium (RTSS)*, pages 279–288, 2002.

[2] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Embedded Control Systems Development with Giotto. In *Proc. Int'l Conf. on Languages, Compilers, Tools and Theory for Embedded Systems (LCTES)*, pages 64–72, 2001.

[3] H. Kopetz. Event-triggered versus Time-triggered Real-time Systems. In A. Karshmer and J. Nehmer, editors, *Operating Systems of the 90s and Beyond*, volume 563 of *Lecture Notes in Computer Science*, pages 86–101. Springer Berlin Heidelberg, 1991.

[4] H. Kopetz. The Systematic Design of Large Real-time Systems or Interface Simplicity. In *Hardware and Software Architectures for Fault Tolerance*, volume 774 of *Lecture Notes in Computer Science*, pages 250–262. Springer Berlin Heidelberg, 1994.

[5] Y. Pribadi, J. P. M. Voeten, and B. D. Theelen. Reducing Markov Chains for Performance Evaluation. In *Proc. of workshop on Embedded Systems*, pages 173–179, 2001.

[6] B. Theelen et al. Software/hardware engineering with the parallel object-oriented specification language. In *Proc. Int'l Conf. on Formal Methods and Models for System Design (MEMOCODE)*, pages 139–148, 2007.

[7] B. Theelen, J. Voeten, and R. Kramer. Performance modelling of a network processor using POOSL. *Computer Networks*, 41(5):667 – 684, 2003.

[8] B. D. Theelen et al. A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In *Proc. Int'l Conf. on Formal Methods and Models for System Design (MEMOCODE)*, pages 185–194, 2006.

[9] L. J. van Bokhoven. *Constructive Tool Design for Formal Languages: From Semantics to Executing Models*. PhD thesis, Eindhoven University of Technology, 2002.

[10] P. van der Putten and J. Voeten. *Specification of Reactive Hardware/Software Systems*. PhD thesis, Eindhoven University of Technology, 1997.