Int J Softw Tools Technol Transfer (2013) 15:211–228
DOI 10.1007/s10009-012-0245-y

EUROSYS

Timed-automata based schedulability analysis for distributed firm real-time systems: a case study

Thi Thieu Hoa Le · Luigi Palopoli · Roberto Passerone · Yusi Ramadian

Published online: 6 July 2012

Abstract The growing level of complexity of modern embedded systems, which are increasingly distributed and heterogeneous, calls for new design approaches able to reconcile mathematical rigour with flexibility, user-friendliness and scalability. In the last few years, Timed Automata (TA) have emerged as a very promising formalism for the availability of very effective verification tools. However, their adoption in the industrial community is still slow. The reluctance of industrial practitioners is partly motivated by persistent concerns on the ease of use of this formalism, on the scalability of the verification process and on the quality of the feedback that the designer receives. In this paper, we discuss these issues by showing the application of the TA formalism on a case study of industrial complexity. We expose the generality of the approach, the efficiency of state of the art tools, but also the limitations in the semantics and in dealing with free design parameters.

Keywords Embedded systems · Timed automata · Real-time scheduling

1 Introduction

In recent years, the complexity of embedded systems (ES) has risen to unexpected levels. The once elementary and strictly dedicated machines have now evolved into flexible and networked systems, made of a large set of heterogeneous components. In this context, the adjective "heterogeneous" spans a wide range of possible meanings. Modern embedded systems are heterogeneous in the way they process data and react to events (synchronously or asynchronously), in the different network protocols that they use, in the different scheduling policies they share resources by and in the different degrees of criticality of their timing requirements (soft, hard, firm). The different combinations of these features create a variety of design issues that are currently addressed by a variety of different methods. While no silver bullet yet exists to manage the complexity of ES design, researchers and industrial practitioners aim for approaches that can claim a sufficient level of generality to be applicable in most contexts of practical relevance.

The real-time scheduling analysis (RTSA) [1] has gained an undisputed popularity in the past few years for the efficiency and the intuitiveness of the techniques it proposes. However, it suffers from important limitations. First, it is strongly focused on particular families of scheduling algorithms (priority based) and of computation models (periodic activations). Second, it only addresses temporal properties of the tasks, which are simply modelled as activities requiring computation or communication time. Unfortunately, the correctness of ES encompasses both functional and non-functional aspects, and the two are not always easy to separate out. Third, the RTSA is necessarily based on worst case scenarios that are not necessarily likely to occur. This often leads to over-provisioning resource allocations, hardly an acceptable choice in many cases.

Very different in nature are the approaches based on timed automata [2]. By using these abstractions, designers can create descriptions of the different components at varying levels of detail. What is more, they can investigate properties of different kind (functional, non-functional and combinations thereof) using such advanced verification tools as UPPAAL [3]. As promising as they may seem, the approaches based on timed automata have so far encountered a lukewarm reception among designers. Some of their most important

T. T. H. Le · L. Palopoli (✉) · R. Passerone · Y. Ramadian
DISI, University of Trento, Trento, Italy
e-mail: palopoli@disi.unitn.it

concerns are on the ability of the TA formalism to express a complex system, on their user friendliness, on the scalability of verification and synthesis methods, and on the quality of the feedback that the designer receives. The first concern includes two different points: (1) is the TA semantics powerful enough to capture systems of industrial complexity? (2) is the TA formalism offered by modelling tools user friendly enough to be used by non specialists? The second concern comes from the bad reputation of formal methods when they are applied to large systems (the curse of dimensionality). The third one is on the richness of the information that are received from the tool across the different design iterations. An ES is often associated with different parameters whose value is unknown (or partially known) upfront. Some of them correspond to design choices and are under the control of the designer (e.g., the bit-rate of a channel, or the complexity of an algorithm), some others model an uncertain knowledge of the system and of its environment (e.g., the inter-arrival time between two user requests). The information of the range of parameters that correspond to feasible design choices (where "feasible" means "complying with some required properties") is in this setting a very valuable one for the designer. Indeed, it can guide him/her in the selection of the design parameters that attain the desired project trade-offs and/or to evaluate the robustness of a candidate design.

In this paper, we explore these issues taking inspiration from an industrial case-study. The complexity of the case-study is sufficient for it to qualify as a realistic design scenario. The system is distributed, and the different activities (tasks) share resources of different kind. The scheduling solutions adopted for the different resources are different (FIFO for some, priority based for others). The properties the system is required to respect regard both its correct functionality (fill level of buffers) and its temporal behaviour (violation of deadlines). Interestingly, the temporal properties are linked with the functional behaviour of the system. The temporal properties refer to a global notion of time, whilst their enforcement depends on a local clock. So a synchronisation procedure has to be executed by the system, and the compliance with the real-time constraints depends on its correct execution. Finally, the correct execution of the systems crucially depends on some design parameters. These features could make our system a very hard match for standard real-time analysis. The purpose of this paper is to show how the system can instead be modelled using timed automata and analysed with a state of the art tool for TA verification (UPPAAL). We aim at exposing the difficulty of the process, the pitfalls of the TA semantics, the scalability and the quality of the feedback to the designer.

The paper is organised as follows. In Sect. 2 we offer a description of the problem, highlighting the properties of interest and the design parameters. In Sect. 3, we discuss how to model the system, using a high level variant of the TA formalism. The model is cast into the TA formalism used by the UPPAAL verification tool in Sect. 4. In Sect. 5, we show the execution of different verification tasks, recording their timing performance and discussing the role of design parameters. In Sect. 6 we discuss the related work and in Sect. 7, we state our conclusions and discuss future work directions.

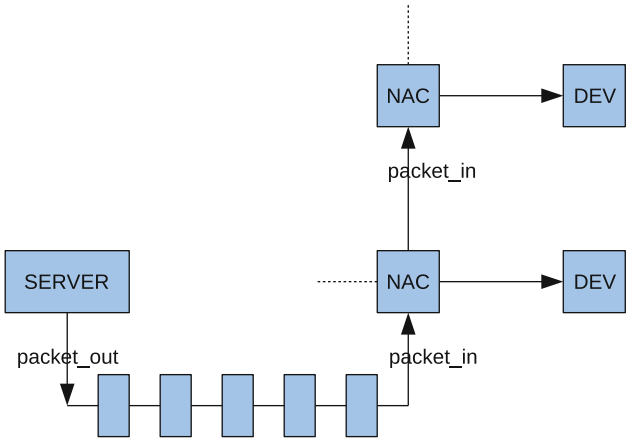


Fig. 1 Heterogeneous communication system (HCS)

2 Problem description

The Heterogeneous Communication System (HCS) is used in commercial aircrafts to distribute music and audio announcements to the main cabin. This is done by streaming a flow of audio packets from a server to a set of terminal devices, each one serving a passenger. Therefore, the HCS consists of a server, a network and a number of devices. The architecture of the HCS is described in detail in a technical report [4]. For the purposes of this paper, it is useful to provide a quick overview. The scheme of the architecture is depicted in Fig. 1. The different components (which are in the order of hundreds) are interconnected through a set of homogeneous networks and exchange a large number of packets for communication, control and monitoring. The gateway and routing functionalities are implemented by special purpose components, defined Network Access Controllers (NACs), which are arranged in a daisy chain topology.

2.1 Applications running in the HCS

The primary purpose of the HCS is to stream audio packets from the server to the devices. The audio streaming application requires a periodic packet delivery every P_a time units (tus) from the server to the devices. Every packet has to be played back at a time instant t_{play}, which is recorded in the packet upon its creation. An excessive delay accumulated along the path from server to device results in the impossibility to comply with this constraint (as detailed below). Besides, the devices are required to reproduce the packets

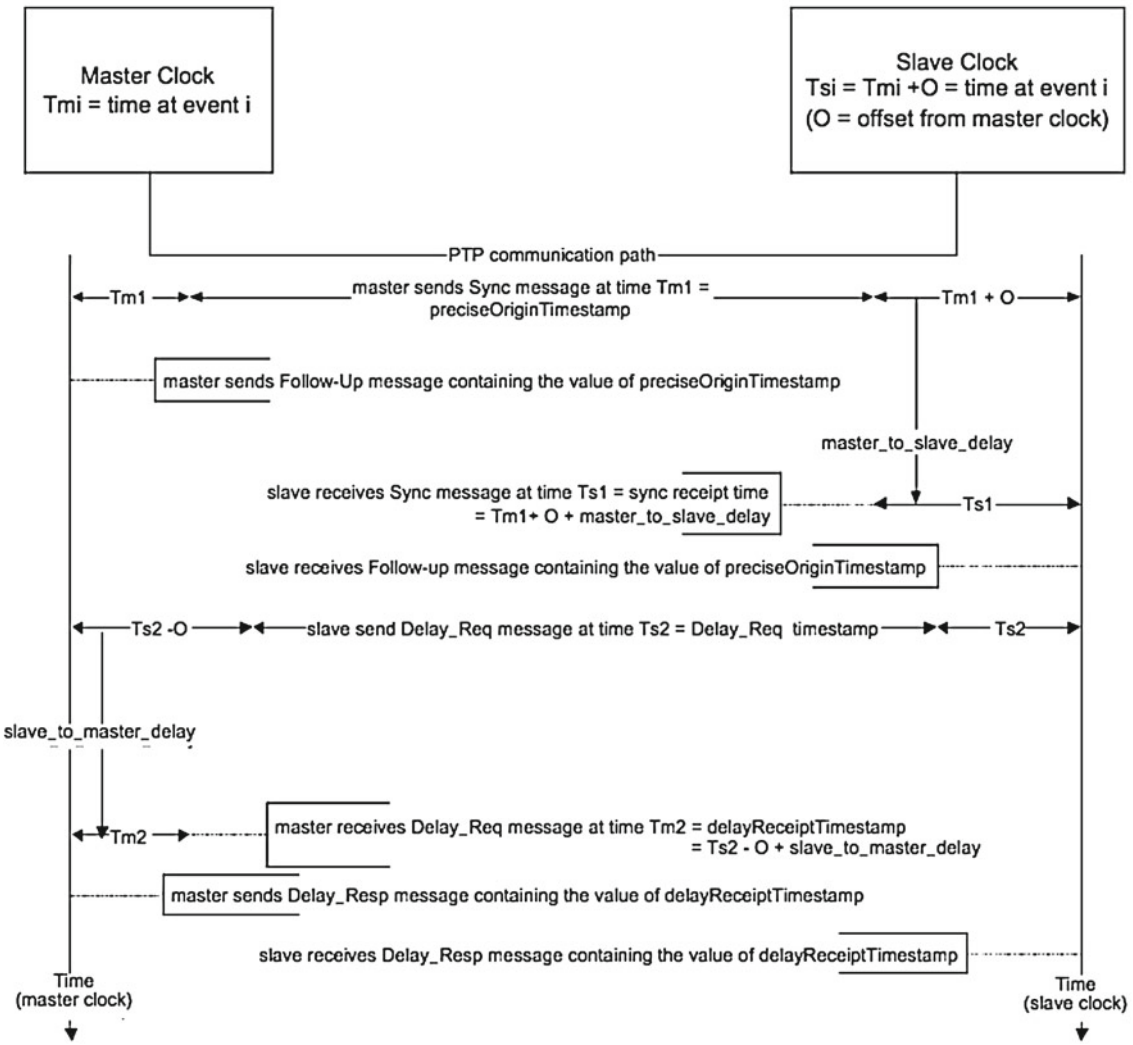


Fig. 2 PTP messages exchanged between master and slave clock to achieve time synchronisation (Courtesy IEEE Standard 1588)

at the same instant, to avoid echoing effects along the cabin. For these reasons an accurate synchronisation between server and devices, and between each device is an imperative requirement.

This is obtained by applying the Precision Time Protocol (PTP) [5]. The task implementing the PTP protocol is itself periodic with period P_s. Figure 2 depicts the message sequence required to synchronise the device (slave) clock with the server (master) clock using PTP. We will illustrate the idea of PTP under the assumption that the relation between the clocks of Master and slaves are given by $T_{\text{Slave}} = T_{\text{Master}} + O$, where O is an unknown offset. The synchronisation procedure is executed iteratively. At the start of each iteration, the Master sends a `Sync` packet and records the exact time T_{m1} when the packet is transmitted. This information is recorded and sent to the slave in a subsequent `Follow_Up` packet. The slave receives this information at time

$$T_{s1} = T_{m1} + O + D, \tag{1}$$

where D is the transmission delay. The slave replies by sending a `Delay_Req` packet at time T_{s2}. The master receives this packet at time

$$T_{m2} = T_{s2} - O + D, \tag{2}$$

which is recorded and sent back to the slave in a `Delay_Resp` packet. The slave may therefore compute D and O by the simple formula:

$$D = \frac{T_{m2} - T_{m1} + T_{s1} - T_{s2}}{2} \tag{3}$$

$$O = T_{s1} - T_{m1} - D = \frac{T_{s1} - T_{m1} - T_{m2} + T_{s2}}{2} \tag{4}$$

This iteration has to be repeated to control the drift between the clocks, which can change in time due to changes in the environment. This description applies to the case of Master and Slave being connected to the same network. In this case, the delay D relates only to the "physical" transmission and can easily be assumed to be the same in both directions.

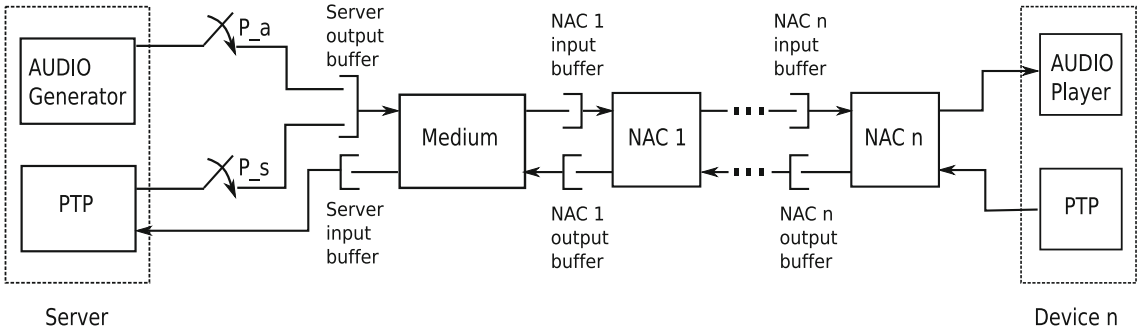


Fig. 3 Block diagram of the end-to-end information flow between server and device in the HCS

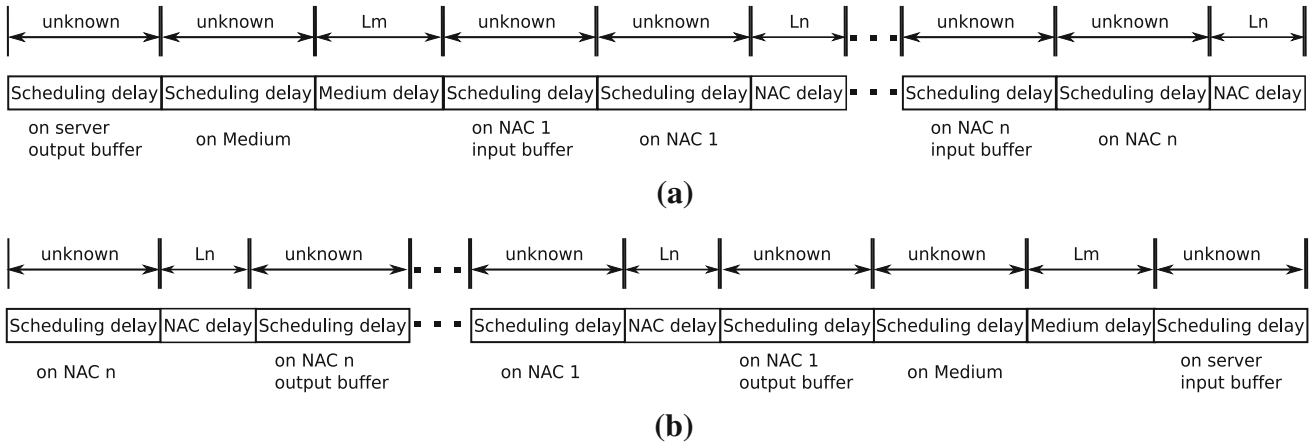


Fig. 4 Time-line of the end-to-end delays incurred by a packet. **a** Server to device communication, **b** device to server communication

In our topology (see Fig. 1), there are sub-networks connected through the NACs. The scheduling delays incurred by a packet along its travel make the delays accumulated in the two directions very different. This problem has to be managed at the scheduling level, as discussed next.

2.2 Scheduling

A more detailed scheme of the HCS representing the end-to-end information flow between the server and one device is reported in Fig. 3.

Consider a packet sent by the sever to a device. The time-line reporting the different delays incurred along the way is reported in the top half of Fig. 4. After the send request, the packet is inserted into an output buffer, and it waits until it is scheduled for transmission. The position in the buffer is related to the packet priority. When the packet reaches the top of the buffer, it can be scheduled for transmission on the medium, which is itself managed by a scheduler. The scheduling discipline used for the medium is priority based and non-preemptive: on-going packet transmissions are not interrupted regardless of the priority of the packet being transmitted. Therefore, the scheduling delay accumulated by the packet awaiting transmission is related to the presence of higher priority transmission requests from other devices, and to possible ongoing transmissions. When the packet wins the competition for the medium, it can be transmitted and it incurs a transmission delay L_{m}. For the sake of simplicity we will consider this transmission time to be the same for all the packets. After the transmission, the packet is placed into the input buffer of the NAC, which is managed using a FIFO policy. When the packet reaches the top position of the buffer, it can be processed by the NAC. This phase introduces an additional scheduling delay due to the fact that the NAC manages several input and output buffers. Also in this case, the policy is priority based and non-preemptive. After the packet gains control of the NAC, the processing time required is L_{n} (assumed equal for all packets). This sequence of actions is repeated as many times as required to traverse the chain until the packet reaches the NAC connected to the device. Similar considerations apply to the inverse direction (bottom half of the Fig. 4).

As mentioned earlier, the network is used for two different packet streams: the PTP and the audio stream. Both streams are triggered by periodic activations (with period P_{s} and P_{a} respectively). As far as the scheduling priority is concerned, the packets used by the PTP have, in our setting, a higher priority than the ones used for the audio stream. The reasons for this choice are twofold. First, the presence of a drift between the clocks can, at least to a first approximation, be

neglected if the difference between T_{m1}, T_{m2}, T_{s1} and T_{s2} remains moderate. This translates into a requirement that the transmissions of the different packet in the PTP execution remain close to each other. The second and more important reason is that the use of high priority packets for the PTP allows us to keep in check the scheduling delays. This way the transmission delays in the two directions (Eqs. (1) and (2)) are likely to be very similar to each other.

To summarise, the transmission priorities of PTP packets including `Sync` (**S**), `Follow_Up` (**F**), `Delay_Req` (**DQ**) and `Delay_Resp` (**DP**) are assumed to be higher than that of audio packets (**A**) as follows:

$$\mathbf{PR_S} > \mathbf{PR_F} > \mathbf{PR_{DP}} > \mathbf{PR_{DQ}} > \mathbf{PR_A},$$

where $\mathbf{PR_X}$ denotes the data priority of packet **X**.

2.3 Temporal properties

A prominent role in the HCS is played by the buffers, which are located in different parts of the system, as shown in Fig. 3. For example, the server has an output buffer where outgoing packets await transmission while the medium is busy. The NACs have two buffers to contain incoming or outgoing packets. All these buffers have finite length. Therefore, a first property of interest is *whether or not any of such buffers overruns during the operation of the HCS*.

To simplify the model, we have assumed that the server responds in a negligible time to a `Delay_Req` packet. Therefore, we will not consider the overrun property for the input buffer of the server. The property will be verified for the buffers located on the NAC and for the output buffer of the server.

The second property that we will verify is on the correct delivery of the audio packet. If the k^{th} audio packet is generated at time $t^{(k)}$, it has to be played back at time $t^{(k)}_{\text{play}}$. The end-to-end response time $R = t^{(k)}_{\text{play}} - t^{(k)}$ is fixed at design time. This requirement corresponds to setting a deadline for the chain of computation initiated by the creation of a new audio sample: in order for the packet to be processed and the sound sample to be played back, the sum of all the delay component detailed in Sect. 2.2 has to be smaller than R. If this temporal constraint is violated the sample is dropped. The existence of a scheduling solution that makes for the satisfaction a deadline is said *schedulability*. We will consider two possible flavours of the schedulability property: hard-real time schedulability and firm real-time schedulability. A system is said hard real-time schedulable if all deadlines are always met. In contrast a firm real-time system is one for which deadlines can occasionally be missed. In particular, we will apply the notion of (m, k)-firm constraints [6]: the task is permitted to miss m deadlines in every group of k. Considering firm real-time constraints allows us to reduce the degree of conservativeness of the analysis: we trade occasional and controlled anomalies for a better utilisation of the system resource, a very frequent choice in multimedia systems.

In the systems outlined above, two quantities can be considered as design parameters: the latencies L_m and L_n. Such quantities are related to the packet size and to the channel bit-rate. Hence two types of system verification are possible:

- Verification with ground parameters,
- parametric verification.

By verification with ground parameters, we mean that, for a given choice of parameters L_m and L_n, we want to know if the system respects the schedulability properties (either hard or firm) and if the buffers do not overrun. By parametric verification, we mean that we look for all possible choices of L_m and L_n such that the above properties are satisfied.

3 System model

We present in this section our complete model following the modelling conventions given below.

3.1 Modelling conventions

In the following, we will make use of the notation of timed automata [2]. Similarly to a regular automaton, a timed automaton has a finite set of states or locations L linked by transitions or edges. The automaton also includes a set of *clocks* X_c and a set of state variables X_s. Transitions are characterised by an action label, a guard and an update expression, while states are characterised by an invariant. A transition can be taken in response to the presence of a synchronising label of the same name in another automaton of the system, when the guard evaluates to true, and, in that case, the value of the variables are updated according to the update expression. $\nu!$ is used to denote a sending action and $\nu?$ a receiving action. Together $(\nu!, \nu?)$ is a *synchronisation* between two automata. An edge is usually referred to as a *synchronising* transition if there is a synchronising label on it or an *internal* transition (denoted by τ) if there is no such label. Urgent locations are marked with a **U** letter, meaning that time elapse is not allowed when the automaton is staying in one of these locations. A transition must be taken if the invariant of the state becomes false. The semantics of a timed automaton is regulated by a global notion of continuous time. While the automaton resides in a state, the value of the clock variables grows linearly with time, and can be reset only upon the execution of an update expression on a transition. The value of state variables can instead be changed only as a result of an update expression when a transition is taken. The updated

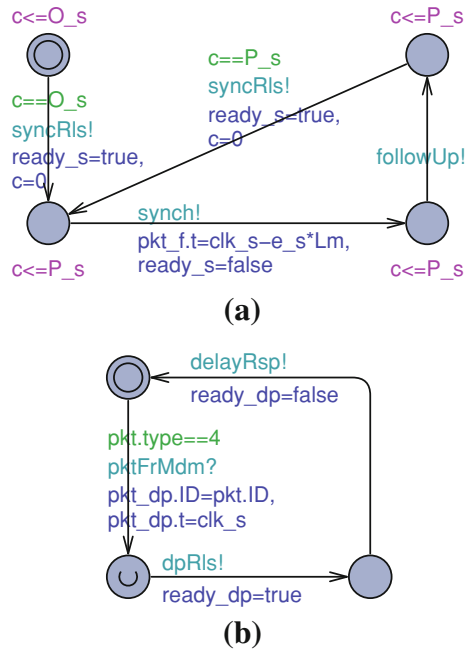


Fig. 5 PTP implementation on server

values for clocks and variables are effective after the transition is taken.

A run (or trace) of a timed automaton is a sequence of states and transitions compatible with the structure of the automaton. When taking the composition of two automata, their transitions must synchronise on the labels and the variables according to rules that depend on the specific timed automata model. Synchronisations are classified into regular and broadcast synchronisations and are discussed in more details in Sect. 3.6.

Our model is composed of 12 timed automata described below.

3.2 Server

The server runs various applications including the PTP implementation and audio generation. The PTP is performed periodically on the server by broadcasting **S**- and **F**-messages to end devices every P_s with a possible non-zero offset O_s (shown in Fig. 5a), and by replying to a **DQ** from a device with a **DP** (shown in Fig. 5b). Variables c and clk_s are periodic clocks (reset after every P_s) and server clock (never reset) whose drifting rate w.r.t. the real time is e_s (i.e., $c = clk_s = e_s \times c_{real}$). In the sequel sections, c will always denote a clock.

The readiness of a packet **X** is indicated by setting a boolean variable $ready_x$ to *true* upon an **X**-release or *false* upon its delivery to the medium. Similarly, a composite variable pkt_x denotes the content of a packet **X**. Such a variable has three members: (i) $type = 1/2/3/4/5$ to indicate that

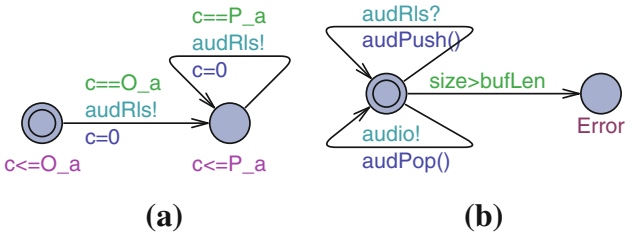


Fig. 6 Periodic audio generation and buffering. **a** Audio generation, **b** audio buffering

X = **S**/**F**/**DP**/**DQ**/**A** respectively, (ii) t to denote the time at which **A** must be played by devices (i.e., $pkt_a.t$), or **S** is sent (i.e., $pkt_f.t$),[1] or **DQ** is received (i.e., $pkt_{dp}.t$), (iii) ID to record the device identity in case the packet is a **DQ** or **DP**. Only some assignments of these members are shown in the figure, others are omitted for the sake of brevity (e.g., $pkt_f.type = 2$).

The periodic audio release and buffering[2] are shown respectively in Fig. 6a and b where $audPush()$ and $audPop()$ represent respectively the buffer push and pop operations of audio messages. The audio stream is characterised by the offset O_a for the first release (transition from initial state to the second state), and is then periodic afterwards (self transition on the second state). Each **A**-packet is time-stamped with the time it has to be played at devices and are placed in a waiting-for-transmission buffer. The interval between two consecutive audio times to play is assumed to be P_a.

The buffer push[3] and pop operations are shown in Procedure 1 and 2, respectively. When a buffer over-run occurs ($size > bufLen$), the buffer automaton goes to the Error state.

Procedure 1 : audPush()

1: buffer[tail] = t_{play}
2: t_{play} = t_{play} + P_a
3: tail = next(tail)
4: size = size+1
5: $pkt_a.t$ = buffer[head]
6: $ready_a \stackrel{def}{=} (size > 0)$

3.3 Devices

To model the PTP implementation on a device, two automata are devoted to observing the PTP stream and recording the sending/arriving time of **S** in $t1/t2$ (Fig. 7a) and **DQ** in $t3/t4$

[1] The delivery of an **S**-packet is confirmed at the end of the transmission, hence the delivery time is $clk_s - e_s \times L_m$.

[2] We consider bounded First-In-First-Out buffers with finite storage memory and implement them as circular buffers.

[3] Initially, $t_{play} = O_a + D_{rel}$ where D_{rel} is the audio relative deadline.

Procedure 2 : audPop()

1: **if** size > 0 **then**
2: head = next(head)
3: size = size-1
4: $pkt_a.t$ = buffer[head]
5: **end if**
6: $ready_a \stackrel{\text{def}}{=} (\text{size} > 0)$

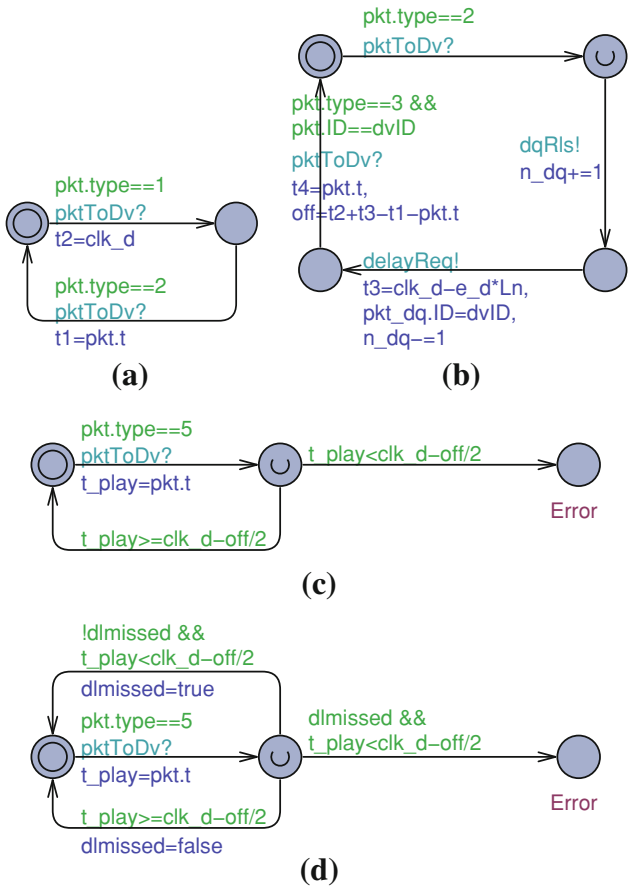


Fig. 7 PTP implementation and audio deadline verifier on end devices

(Fig. 7b) for the offset calculation between server and device clocks as shown in Procedure 3. The device clock clk_d is also a drifting clock with rate e_d and $t2, t3$ are measured w.r.t. this clock clk_d.

Every device has its own identity denoted by a positive dv_{ID} which is used to indicate the source of a **DQ**. As illustrated in Fig. 1, a NAC can be associated to many devices. Thus, there can be many **DQ**s available simultaneously and n_{dq}, which is a shared variable between the NAC and all of its associated devices, is used to record the presence of these **DQ**s. For simplicity, the NAC is assumed to resolve the contention by selecting randomly a **DQ**. The device that sends its **DQ** successfully to the NAC can then switch to waiting the corresponding **DP** from the server.

Procedure 3 : Offset calculation

1: $clk_d = clk_s + off_d$
2: $delay = (t_2 - off_d) - t_1$
3: $delay = t_4 - (t_3 - off_d)$
4: $2off_d = t_2 + t_3 - t_1 - t_4$
5: $off = t_2 + t_3 - t_1 - t_4$, where $off = 2off_d$
6:
7: $off_0 = 0$
8: $clk_0 = clk_d - off_0/2$
9:
10: $off_1' = (t_2 - off_0/2) + (t_3 - off_0/2) - t_1 - t_4$
11: $off_1 = off_0 + off_1' = t_2 + t_3 - t_1 - t_4$
12: $clk_1 = clk_0 - off_1'/2 = clk_d - off_1/2$
13: ...
14: $off_n = off_0 + \sum_1^n off_i' = t_2 + t_3 - t_1 - t_4$
15: $clk_n = clk_{n-1} - off_n'/2 = clk_d - off_n/2$

Figure 7c describes the activities of an audio-hard-deadline checker implemented on a device. Upon receiving an audio packet, its time-to-play t_{play} parameter is retrieved and checked with the current time of the device clock. If the clock has already passed this t_{play}, the automaton goes to the Error state. For soft-deadline constraints, the automaton can be modified to tolerate some deadline misses. For example, in case two consecutive audio deadline misses are allowed, an additional boolean variable $dlmissed$ can be used to indicate whether the previous deadline was missed as in Fig. 7d.

3.4 Network medium

The network medium is responsible for data transmission into subnets which must take place whenever any data is available, unless the medium is currently busy. In this heterogeneous system, the data priority (**PR**) is assumed as follows:

$$\mathbf{PR_S} > \mathbf{PR_F} > \mathbf{PR_{DP}} > \mathbf{PR_{DQ}} > \mathbf{PR_A}$$

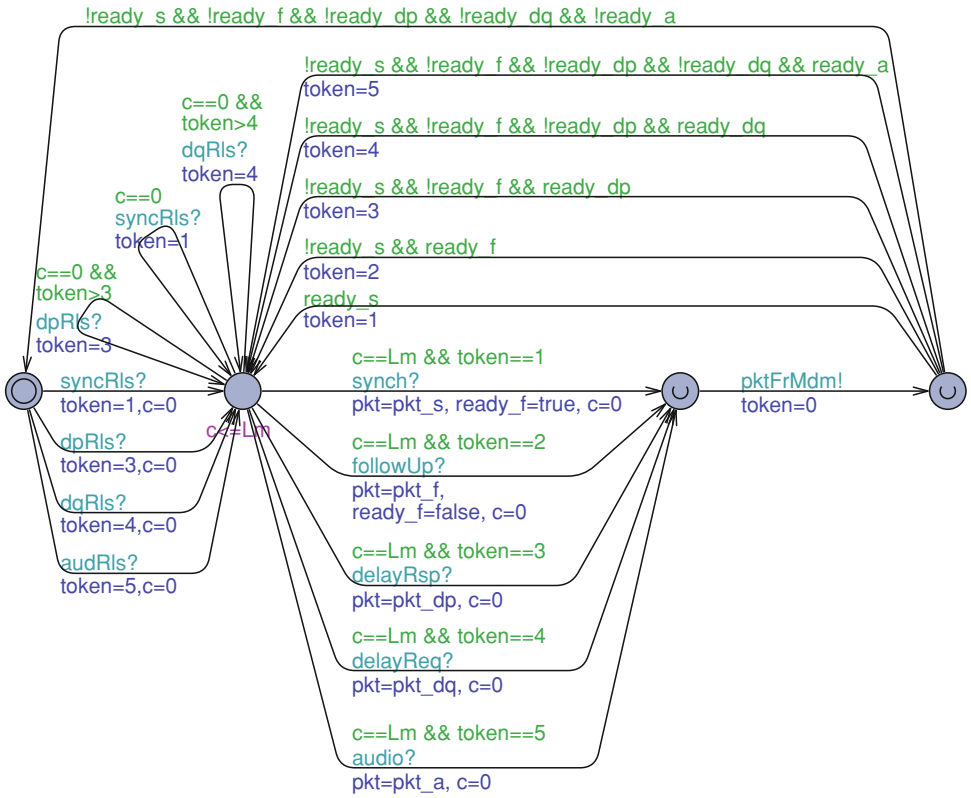
The data of highest priority (denoted by $token$ in Fig. 8) is transmitted first and others have to back-off and wait for their turn. The network transmission is done after L_m (tus) and the data is forwarded to the NACs which then pass it onto subnets.

Variables pkt_x and $ready_x$ ($\mathbf{X} \neq \mathbf{DQ}$) are shared between the medium and server components of the model. Particularly, pkt_a and $ready_a$, denoting the head and positive size of the audio buffer, are updated in Procedures 1 and 2. Likewise, variables pkt_{dq} and $ready_{dq}$, shared between the medium and associated **DQ**-sources, denote the winning **DQ** and positive number of **DQ**s waiting for a transmission (i.e., $ready_{dq} \stackrel{\text{def}}{=} (n_{dq} > 0)$).

3.5 Network access controllers (NACs)

NACs are responsible for data routing from the server into subnets and vice versa as shown in Fig. 9c. In addition,

Fig. 8 Network medium



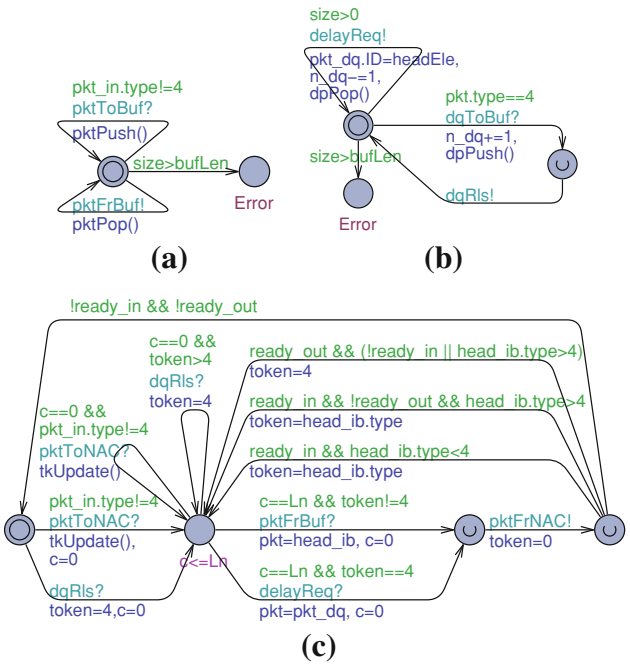


Fig. 9 Input buffer, output buffer and NAC

NACs can perform data encoding/decoding on the data passing through it which takes L_{n} (tus). Because only one packet can be processed at a time, NACs must be capable of buffering data. In this model, NACs are assumed to have two types of buffer: input buffer for data from the server to devices (i.e., **S**, **F**, **DP**, **A**) and output buffers for **DQ** from devices to the server. These buffers are depicted in Fig. 9a and b. The definitions of the push and pop operations for these buffers are similar to those given for the audio buffer in Sect. 3.2.

As the network medium (or NAC) can be associated to many NACs, there are also many NAC output buffers contending for a **DQ**-transmission. The contention is resolved in a manner similar to that presented in Sect. 3.3. Here, $headEle$ stands for the head element of the output buffer and pkt denotes the packet currently processed by its associated NAC.

Also similar to the transmission mechanism of the medium, a NAC always processes the data of highest priority. Deciding which data to process is based on the availability of data from the server and **DQ**s from devices. Such information is available from variables $head_{\mathrm{ib}}$ and $ready_{\mathrm{in}}$, which denote the head and positive size of its input buffer and variables pkt_{dq}, $ready_{\mathrm{out}}$ (shared between a NAC and its associated **DQ**-sources) which denote the winning **DQ** and positive number of **DQ**s still queuing for their turn (i.e., $ready_{\mathrm{out}} \stackrel{\mathrm{def}}{=} (n_{\mathrm{dq}} > 0)$).

3.6 Synchronisation and transmission modelling mechanism

A synchronisation can be either regular or broadcast. Regular synchronisations require both sending and receiving

actions to happen simultaneously while the sending action can always happen (provided that the guard is satisfied) in a broadcast synchronisation, no matter if any receiving actions are enabled. The receiving actions that are enabled will synchronise.

The transmission mechanism is modelled by shared variables and broadcast synchronisations. The transmission arbiters implemented inside the transmitters (e.g., network medium, NACs) must have the ability to select the data of highest priority among available data for transmission at two points of time as follows.

- When a transmitter is idle (i.e., staying in the initial location and no data is available as $ready_i = false$ where $i \in \{s, f, dp, dq, a, in, out\}$), it waits for any broadcast sending actions ($syncRls!$, $dpRls!$, $dqRls!$, $audRls!$) to happen and synchronises with those enabled at the same time in a random *sequential* order with *no time elapse* in between. For example, let $O_s = O_a = 0$ so initially two broadcast sending actions are enabled: $syncRls!$, $audRls!$ and two packets **S** and **A** waiting to be transmitted through the network medium. The synchronising order can be either: $audRls$, $syncRls$ or $syncRls$, $audRls$. No matter what order is chosen, the arbiter guarantees to transmit the highest priority data first, which in this example is **S**. This is because the value of $token$ is set to 1 in either synchronising orders.
- When a transmitter has just finished its current transmission (i.e., after $pktFrMdm?$ or $pktFrNAC?$ is taken), it waits for the arbiter decision on the next transmission and takes an internal transition accordingly. The transmission decision is based on the information given by variables $ready_i$ and the data priority in $.type$. All broadcast sending actions to the transmitter happening before an internal transition is taken can change variables $ready_i$ and $.type$, thereby possibly affecting the arbiter transmission decision. Otherwise, the transmitter will synchronise with them in a random sequential order as described above.

It is possible to model the arbiter transmission decision by using *interleaving semantics*, which requires the number of internal transitions and synchronisations executed in parallel to be at most 1. So, transitions which are enabled at the same time are taken sequentially without time elapse in between. In the modelling environment where interleaving semantics is not supported, the model requires minor modification. That is, all broadcast synchronisations are made regular by adding dummy receiving actions to ensure that the transmitter can always synchronise with a broadcast sending source in whatever state. As the transmitter cannot do multiple synchronisations simultaneously, they will have to be taken in a random sequential order as desired. Figure 10 shows an example of adding dummy receiving actions to the medium automaton in Fig. 8 to make $syncRls$ a regular synchronisation.

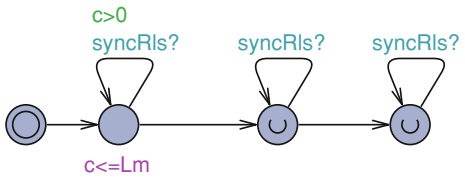


Fig. 10 Add dummy transitions to make $syncRls$ a regular synchronisation

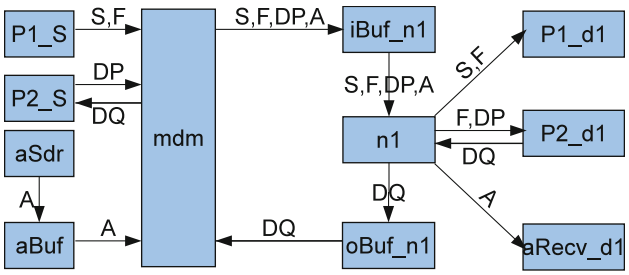


Fig. 11 Packet streams in simple HCS

Table 1 System instantiation for a simple example

Name	Template
$P1_S$	5a
$P2_S$	5b
$aSdr$	6a
$aBuf$	6b
$P1_{d1}$	7a
$P2_{d1}$	7b
$aRecv_{d1}$	7c
mdm	8
$iBuf_{n1}$	9a
$oBuf_{n1}$	9b
$n1$	9c

3.7 Example

Consider a simple system that consists of a server, a network medium, a NAC, a device and whose flows of PTP and audio messages are depicted in Fig. 11. To represent such a system, 11 automata whose templates are described in Figs. 5, 6, 7, 8, 9c are needed. For the sake of convenience in explanation and reasoning, these automata are named as shown in Table 1.

Table 2 summarises all the synchronisations (regular and broadcast) between the automata. An action is denoted by $Automaton$.Label, for example $P1_s$.synch! indicates the **S**-sending action of the automaton in Fig. 5a. Table 3 provides a summary of all the variables shared between the automata

Table 2 Synchronisations between automata

Type	Name	Sending action	Receiving action(s)
Regular	synch	$P1_s$.synch!	mdm.synch?
Regular	followUp	$P1_s$.followUp!	mdm.followUp?
Regular	delayRsp	$P2_s$.delayRsp!	mdm.delayRsp?
Regular	dqToMdm	$oBuf_{n1}$.delayReq!	mdm.delayReq?
Regular	dqToN1	$P2_{d1}$.delayReq!	$n1$.delayReq?
Regular	audio	$aBuf$.audio!	mdm.audio?
Regular	pktFrIBufN1	$iBuf_{n1}$.pktFrBuf!	$n1$.pktFrBuf?
Broadcast	syncRls	$P1_s$.syncRls!	mdm.syncRls?
Broadcast	dpRls	$P2_s$.dpRls!	mdm.dpRls?
Broadcast	dqRlsToN1	$P2_{d1}$.dqRls!	$n1$.dqRls?
Broadcast	dqRlsToMdm	$oBuf_{n1}$.dqRls!	mdm.dqRls?
Broadcast	audRls	$aSdr$.audRls!	$aBuf$.audRls?
			mdm.audRls?
Broadcast	pktFrMdm	mdm.pktFrMdm!	$P2_s$.pktFrMdm?
			$iBuf_{n1}$.pktToBuf?
			$n1$.pktToNAC?
Broadcast	pktFrN1	$n1$.pktFrNAC!	$P1_{d1}$.pktToDv?
			$P2_{d1}$.pktToDv?
			$aRecv_{d1}$.pktToDv?
			$oBuf_{n1}$.dqToBuf?

Table 3 Shared variables between automata

Global name	Corresponding local names
$t1_{d1}$	$P1_{d1}.t1$, $P2_{d1}.t1$
$t2_{d1}$	$P1_{d1}.t2$, $P2_{d1}.t2$
L_n	$P2_{d1}.L_n$, $n1.L_n$
L_m	$P1_s.L_m$, $mdm.L_m$
n_dq_{n1}	$P2_{d1}.n_{dq}$, $n1.n_{dq}$
n_dq_m	$oBuf_{n1}.n_{dq}$, $mdm.n_{dq}$
off_{d1}	$P2_{d1}.off$, $aRecv_{d1}.off$
pkt_s	$P1_s.pkt_s$, $mdm.pkt_s$
pkt_a	$aBuf.pkt_a$, $mdm.pkt_a$
pkt_m	$mdm.pkt$, $P2_s.pkt$
	$iBuf_{n1}.pkt\_in$, $n1.pkt\_in$
pkt_{n1}	$n1.pkt$, $P1_{d1}.pkt$, $P2_{d1}.pkt$
	$aRecv_{d1}.pkt$, $oBuf_{n1}.pkt$
pkt_{dp}	$P2_s.pkt_{dp}$, $mdm.pkt_{dp}$
pkt_dq_m	$oBuf_{n1}.pkt_{dq}$, $mdm.pkt_{dq}$
pkt_dq_{n1}	$P2_{d1}.pkt_{dq}$, $n1.pkt_{dq}$
$head\_ib_{n1}$	$iBuf_{n1}.head\_ib$, $n1.head\_ib$
$ready_s$	$P1_s.ready_s$, $mdm.ready_s$
$ready_a$	$aBuf.ready_a$, $mdm.ready_a$
$ready_{dp}$	$P2_s.ready_{dp}$, $mdm.ready_{dp}$
$ready_{dq}$	$oBuf_{n1}.ready_{dq}$, $mdm.ready_{dq}$
$ready\_in_{n1}$	$iBuf_{n1}.ready\_in$, $n1.ready\_in$
$ready\_out_{n1}$	$P2_{d1}.ready\_out$, $n1.ready\_out$

Table 4 Packets sent with synchronisations

Global name	Synchronisations
pkt_s	$P1_s$.synch!,mdm.synch?
pkt_a	$aBuf$.audio!, mdm.audio?
pkt_m	mdm.pktFrMdm!, $P2_s$.pktFrMdm?
	$iBuf_{n1}$.pktToBuf?, $n1$.pktToNAC?
pkt_{n1}	$n1$.pktFrNAC!, $P1_{d1}$.pktToDv?, $P2_{d1}$.pktToDv?
	$aRecv_{d1}$.pktToDv?, $oBuf_{n1}$.dqToBuf
pkt_{dp}	$P2_s$.delayRsp!, mdm.delayRsp?
pkt_dq_m	$oBuf_{n1}$.delayReq!, mdm.delayReq?
pkt_dq_{n1}	$P2_{d1}$.delayReq!, $n1$.delayReq?
$head\_ib_{n1}$	$iBuf_{n1}$.pktFrBuf!, $n1$.pktFrBuf?

and Table 4 shows which packet is sent in which synchronisation. For each of the automata that share a variable, it has a copy and a local name for that variable. The semantics for a shared variable with local copies is that whenever a local copy is changed, it is immediately effective on the global variable, i.e., it is also changed.

Since we accept the convention that updating clocks and variables are only effective after the edge is taken, there seems to exist a semantic problem in retrieving the correct information of a **DQ** sent from $oBuf_{n1}$ to mdm (or from $P2_{d1}$ to $n1$). Because $oBuf_{n1}.pkt\_dq$ (or $P2_{d1}.pkt\_dq$), which represents that **DQ**, is updated only after the synchronisation $\{oBuf_{n1}$.delayReq!, mdm.delayReq? $\}$ (or $\{P2_{d1}$.delayReq!, $n1$.delayReq?$\}$) happens, any read operation on the variable during or before the synchronisation will not get the expected value, e.g., reading and assigning this variable to $mdm.pkt$ in action mdm.delayReq? (or $n1.pkt$ in action $n1$.delayReq?) in the model. However, this turns out not to be so problematic when some particular modelling construct is exploited. For example, in UPPAAL, the update expressions in a sending action are executed before the update expressions on a corresponding receiving action, so the problem dissolves. In another modelling languages like NuSMV [7] which does not adopt such convention, variables are updated by specifying their next value with the keyword **next**. So the problem is also solved by specifying:

$$\textbf{next}(mdm.pkt) = \textbf{next}(mdm.pkt\_dq)$$

Consider an example where $P_\mathrm{s} = 8$, $P_\mathrm{a} = 2$, $O_\mathrm{s} = O_\mathrm{a} = 0$, $L_\mathrm{m} = L_\mathrm{n} = 1$, $e_\mathrm{s} = e_\mathrm{d1} = 1$ (clocks do not drift) and audio relative deadline $D_\mathrm{rel} = 2$. Then the deadline of the first **A**, which is 2, is violated as shown by the following transition sequence:

$$\begin{aligned}\alpha = {} & \mathsf{syncRls}, \mathsf{audRls}, 1,\\ & \mathsf{synch}, \mathsf{pktFrMdm}, mdm.\tau, 1,\\ & \mathsf{audRls}, \mathsf{followUp}, \mathsf{pktFrMdm}, mdm.\tau,\end{aligned}$$

pktFrIBufN1, pktFrN1, $n1.\tau$, 1
audio, pktFrMdm, $mdm.\tau$,
pktFrIBufN1, pktFrN1, $n1.\tau$, dqRlsToN1, 1
audRls, audio, pktFrMdm, $mdm.\tau$,
dqToN1, pktFrN1, $n1.\tau$, dqRlsToMdm, 1
dqToMdm, pktFrMdm, $mdm.\tau$,
pktFrIBufN1, pktFrN1, $aRecv.\tau$

3.8 Modelling requirements for HCS

To model a real-time system with clock synchronisation like HCS, we need a modelling language that can express different timed requirements and allow manipulating clocks. Other requirements including shared variables, broadcast channels and interleaving semantics certainly ease the modelling effort but are not mandatory because there are always other modelling alternatives for them.

UPPAAL, which is a mature modelling languages, can provide very efficient verifications and analyses on temporal properties of a system modelled in terms of timed automata. While it supports all the optional modelling requirements for HCS, it does not allow direct clock manipulations. We defer the discussion on this limitation of UPPAAL to the next section where we describe how to model HCS in UPPAAL. Therefore, with UPPAAL the mandatory modelling requirements for HCS stated earlier can only be satisfied partially.

4 UPPAAL model for HCS

Procedure 4 : Integer subtraction

Require: a time point (t_i, d_i),
an offset to be subtracted away off
Ensure: (t, d) is the subtraction result
1: $t = t_i - off$
2: $d = d_i$
3: **if** $t < 0$ **then**
4: $t = t + L_{\text{day}}$
5: $d = 1 - d$
6: **else if** $t > L_{\text{day}}$ **then**
7: $t = t - L_{\text{day}}$
8: $d = 1 - d$
9: **end if**

We have modelled HCS in UPPAAL [3] because the optimised real-time model checker integrated in the UPPAAL modelling environment allows the temporal properties of the system to be efficiently verified and analysed. The pseudo model presented in Sect. 3 is translated to UPPAAL with some modifications on modelling and capturing specific instants of time. Such modifications are not trivial as UPPAAL generally does not support all expression involving

Procedure 5 : Delay Calculation

Require: two time points (t_1, d_1) and (t_2, d_2)
Ensure: $delay$ is the difference between two time points
1: **if** $d_1 == d_2$ **then**
2: $delay = t_2 - t_1$
3: **else**
4: $delay = t_2 + L_{\text{day}} - t_1$
5: **end if**

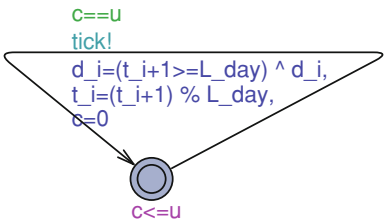


Fig. 12 Integer clock

clocks. For instance, reading a clock and assigning its value to a variable are impossible in UPPAAL since only integer variables are supported there. Access to clocks, however, is essential to the PTP implementation on both the server and devices.

One approach is to try to use integer clocks and introduce the variables t_i whose values are the integer parts of the clock values. The idea is to compute an upper bound u and increase t_i whenever a clock reaches u. Specifically, given $c = e \times c_{\text{real}}$ then $u = \lfloor 1/e \rfloor$ is the clock bound at which c must be reset to 0 and t_i is increased by 1. In addition, u can be measured more finely by taking up to n digits after the decimal point of $1/e$. For example, $e = 0.95$ so $u = \lfloor 1/0.95 \rfloor = \lfloor 1.052 \rfloor = 1$ or better $u = \lfloor 10^3/0.95 \rfloor = 1,052$.

As every integer variable will finally overflow if it keeps increasing, it is necessary to introduce the notion of *date*, which we split into an *odd* and an *even* date, so that t_i can be reset to 0 whenever it reaches the duration of day, denoted by L_{day}. An additional variable d_i, whose value is either 0 (even date) or 1 (odd date), is used to remember the current date of the integer clock. The use of integer clocks gives rise to additional considerations and complications:

- An additional automaton is specified to model the integer clocks. Figure 12 shows a UPPAAL template of such automaton. Here, ^ and % denote the bitwise XOR and modulo operator respectively. Moreover, the transition $tick!$ must always happen before any reading attempt on integer clocks, thus it must be modelled as a broadcast channel and be given a higher priority than other transitions.
- Invariants on a periodic clock c like $c \leqslant P_s$ must be multiplied by their upper bound u, e.g., $c \leqslant u \times P_s$.
- Consider a subtraction involving clock c like $(c - off)$. To turn this into an integer subtraction, t_i, d_i are introduced as described above and the integer subtrac-

tion stored in t, d as in Procedure 4. Similar expressions involving clocks can be found in Figs. 5a and 7, e.g., $pkt_f.t = clk_s - e_s \times L_m$.

– An essential part of the PTP is to record precisely the sending/arriving time of **S** (or **DQ**) in t_1/t_2 (or t_3/t_4) respectively. Because all clocks are now integer, also the sending/arriving date is recorded in d_1/d_2 (or d_3/d_4). Computing the difference between the sending and arriving time is not straightforward as there is no way to know how many days have elapsed between these two dates, especially when the server and device clocks have different drifting rates. For example:

$$e_s = 0.50 \rightarrow u_s = \lfloor 1/0.50 \rfloor = 2$$
$$e_d = 0.25 \rightarrow u_d = \lfloor 1/0.25 \rfloor = 4$$

Therefore when the device integer clock reaches its bound the second/fourth/eighth time, etc., the server clock hits its bound the fourth/eight/sixteenth time, etc. respectively and the date variables for both clocks are 0. If the drifting rates are constrained to be exactly the same and L_{day} to be greater than the worst delay of any packet in HCS, then the difference between (t_1, d_1) and (t_2, d_2) can be measured as in Procedure 5.

– The verification of audio deadlines encounters a similar problem because the deadlines are checked w.r.t. integer device clocks. When the drifting rates are the same and L_{day} is much greater than both the worst delay of any packet in the system and the device offset off_n calculated in Procedure 3, then audio deadlines can be verified according to Procedure 6. Let (t_p, d_p) denote an audio time-to-play or deadline and (t_i, d_i) denote the time at which a device receives the packet without considering the offset information. Then, (t_g, d_g) or the time at which the packet was generated by the server can be calculated by subtracting away the audio relative deadline D_{rel} from (t_p, d_p) (line 1). Likewise, the subtraction involving off_n (line 2) computes the actual arrival time of the packet. The deadline verification is based on comparing the receipt time without (t_i, d_i) and with (t_{arr}, d_{arr}) considering the offset information against the deadline (t_p, d_p). For example, when $(d_i == d_p) \wedge (t_i \leqslant t_p)$ (line 3), it means the actual dates are coincident and the deadline may be not be due yet because $D_{rel} \ll L_{day}$ and the worst delay is less than L_{day}. Then, a positive offset (i.e., $off_n \geqslant 0$) means the actual arrival time is even earlier and so the deadline is respected (lines 4–5). A negative offset may not violate the deadline as long as the actual arrival time is still earlier than the deadline (lines 4–5). Otherwise, the deadline is violated (lines 6–8).

Procedure 6 : Audio Deadline Verification

Require: offset off_n from Procedure 3,
deadline (t_p, d_p),
without-offset receipt time (t_i, d_i)
Ensure: check $\in$ {violated, satisfied}
1: $(t_g, d_g) = (t_p, d_p) - D_{rel}$
2: $(t_{arr}, d_{arr}) = (t_i, d_i) - off_n/2$
3: **if** $(d_i == d_p) \wedge (t_i \leqslant t_p)$ **then**
4: **if** $(off_n \geqslant 0) \vee (d_{arr} == d_p \wedge t_{arr} \leqslant t_p)$ **then**
5: check = satisfied
6: **else**
7: check = violated
8: **end if**
9: **else if** $(d_i == d_p) \wedge (t_i > t_p)$ **then**
10: **if** $(off_n \leqslant 0) \vee (d_{arr} == d_p \wedge t_{arr} > t_p)$ **then**
11: check = violated
12: **else**
13: check = satisfied
14: **end if**
15: **else if** $(d_i != d_p) \wedge (d_p == d_g)$ **then**
16: **if** $(off_n \leqslant 0) \vee (d_{arr} != d_p) \vee (t_{arr} > t_p)$ **then**
17: check = violated
18: **else**
19: check = satisfied
20: **end if**
21: **else if** $(d_i != d_p) \wedge (d_p != d_g)$ **then**
22: **if** $(off_n \geqslant 0) \vee (d_{arr} != d_p) \vee (t_{arr} \leqslant t_p)$ **then**
23: check = satisfied
24: **else**
25: check = violated
26: **end if**
27: **end if**

5 Analysis of ground parameters

The UPPAAL model for HCS is a network of 13 timed automata of which 11 automata are described in Figs. 5, 6, 7, 8, 9c with further modifications for capturing clock values as detailed in Sect. 4, and 2 additional automata to model the integer server/device clock.

We have performed verification and analyses on the correct functioning of HCS (i.e., audio deadlines are satisfied w.r.t. a hard or firm constraint and buffer overruns do not occur) under a diverse set of parameter settings. We present in this section the experimental results for a simple instantiation of HCS whose structure and connection are depicted in Fig. 11. We have chosen as free parameters the latency L_m and L_n. A point (L_m, L_n) is considered feasible if it does not drive the network of automata to any Error location. The values of the fixed parameters are as follows:

$$P_s = 40,\ P_a = 10,\ O_s = O_a = 0,\ D_{rel} = 10,$$
$$L_{day} = 1000,\ buf_{len} = 10,\ e_s = e_{d1} = 1$$

As discussed in Sect. 4, to ensure the correctness of the comparisons between integer clocks, it is necessary to enforce the following constraints:

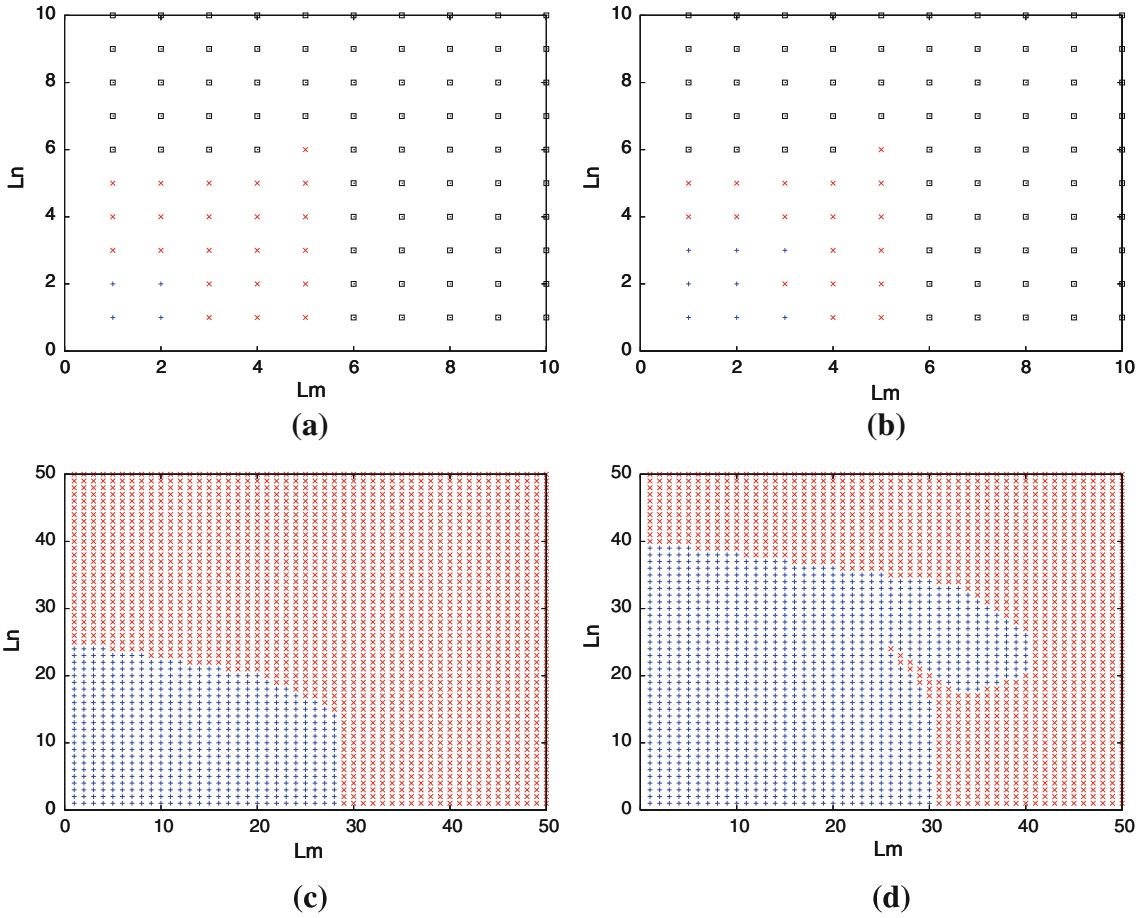


Fig. 13 Point-wise verification result on buffer-overrun (*box*), deadline violation (*times*) w.r.t hard constraint (**a**, **c**) and soft constraint (**b**, **d**), correct functioning (*plus*). **c** Zoomed region between 0 and 5 in **a**. **d** Zoomed region between 0 and 5 in **b**

- drifting rates are exactly the same
- $L_{\text{day}} > off > -L_{\text{day}}$
- $L_{\text{day}} > \text{delay}_{worst}$

The first constraint is respected because in our experiment $e_{\text{s}} = e_{\text{d1}} = 1$, meaning clocks do not drift. The satisfaction of the second constraint is confirmed by the UPPAAL checker. The last constraint can be verified by using the worst delay for an **A**-packet approximated in the following formula, since **A**-packets have the lowest priority:

$$\text{delay}^{A}_{worst} \simeq 10 \times P_{\text{a}} + 10 \times L_{\text{n}} + (10 \times P_{\text{a}}/P_{\text{s}} + 1) \times L_{\text{n}}.$$

where:

- $10 \times P_{\text{a}}$ is the time period after which the packet must be delivered, otherwise a buffer overrun occurs because the buffer can contain at most 10 packets ($buf_{\text{len}} = 10$).
- $10 \times Ln$ is the delay the packet might have to experience if it is inserted at the end of the NAC input queue and so has to wait for the other 9 packets to be processed.
- $10 \times P_{\text{a}}/P_{\text{s}} + 1$ is the number of **S**-packets released until the packet is delivered to the network and since there is only one device in this experiment, it is also the number of **DQ**-packets that are sent upon receiving **S**-packets.
- $(10 \times P_{\text{a}}/P_{\text{s}} + 1) \times L_{\text{n}}$ is the delay the packet may experience when the **DQ**s are processed while the packet is queuing for its turn.

Moreover, to prevent deadlock in Fig. 5a, we need to enforce a further constraint of $P_{\text{s}} \geq 3 \times L_{\text{m}}$. This is because the Sync period must complete sending a pair of **S** and **F** which takes $2L_{\text{m}}$ and the **S** may have to wait for the network medium to become free which takes $1L_{\text{m}}$. We have carried the verification in a point-wise manner. The parameters of interest (i.e., L_{m} and L_{n}) are grounded to different values between 0 and 10 (a value of 10 is chosen to respect the constraint on deadlock prevention). Figure 13a and b show the result of the verification.

In Fig. 13, points marked with □ result in buffer overruns while points marked with × do not, but instead violate a hard or soft deadline constraint. Feasible points which

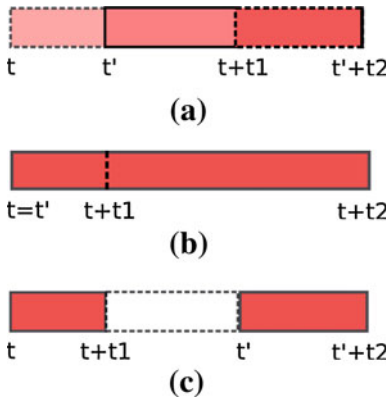


Fig. 14 Variation of latency

can ensure the correct functioning of the system are marked with +. A closer look at the region in Fig. 13a, b where only a hard/soft deadline constraint is violated or satisfied is shown in Fig. 13c/d where the square regions between 0 and 5 are zoomed in. This zooming operation essentially means performing a finer granularity on the free parameters L_m and L_n. Since UPPAAL supports only integer variables, finer granularities can be obtained by performing a z-multiplication. That is, fixed timing parameters (including P_s, P_a, O_s, O_a, D_{rel}, L_{day}) and the vertex coordinates of the square region are multiplied by 10^z where $z \geqslant 1$ denoting the number of digits after the decimal point of L_m and L_n. For example, the feasibility verification on point $L_m = 5$, $L_n = 5$ is enabled in Fig. 13c, d where the 1-multiplication is performed while it is not in Fig. 13a, b since L_m and L_n cannot take on real values like 0.5. The choice of z certainly depends on the largest integer value that UPPAAL can represent and therefore cannot grow arbitrarily large.

In particular, Fig. 13d shows the non-monotonic behaviour of HCS under a soft constraint of audio deadlines. For example, at $L_m = 3$, the system respects the constraint for $0.1 \leqslant L_n \leqslant 1.6$. When L_n grows larger, i.e., $1.7 \leqslant L_n$, the constraint is violated until L_n goes beyond 2.0 when it is satisfied again and remains so as long as $L_n \leqslant 3.4$.

Figures 15, 16, 17 explain graphically the reasons for such non-monotonic behaviour w.r.t. the variation of L_n in the intervals $[t_1, t_2]$ equal to [0.1, 1.6] or [1.7, 2] or [2.1, 3.4]. The variation of L_n from t_1 to t_2 causes the latency of a packet to vary as well, which in turn causes the variation in the latency of the packet transmitted right after it. We use a horizontal bar to denote a latency and depict the variation in latency by two such bars, each of which corresponds to L_n being at the starting/end point of the interval in which it varies. The two bars may partially overlap as shown in Fig. 14a, b) or be separate as in Fig. 14c. In Fig. 14, the starting/end points of the two bars are $t/t + t_1$ and $t'/t' + t_2$.

Figure 15a and b show the packet stream during the first two audio periods for $0.1 \leqslant L_n \leqslant 1.6$. **S**, **F** and **A** packets are delivered to the medium at time 0, 3 and 6 respectively. They are then forwarded to the NAC at time 3, 6 and 9 respectively. As L_n changes from 0.1 to 1.6, the release time of a **DQ**-packet changes from 6.1 to 7.6 upon receiving the **F**-packet and the starting time of the **DQ** transmission through the medium changes from 9 to 9.2. Note that the medium was busy finishing the first **A**-transmission before time 9. Upon receiving the **DQ**-packet, the server replies with a **DP** at time ranging from 12 to 12.2 w.r.t. the variation of L_n between 0.1 and 1.6.

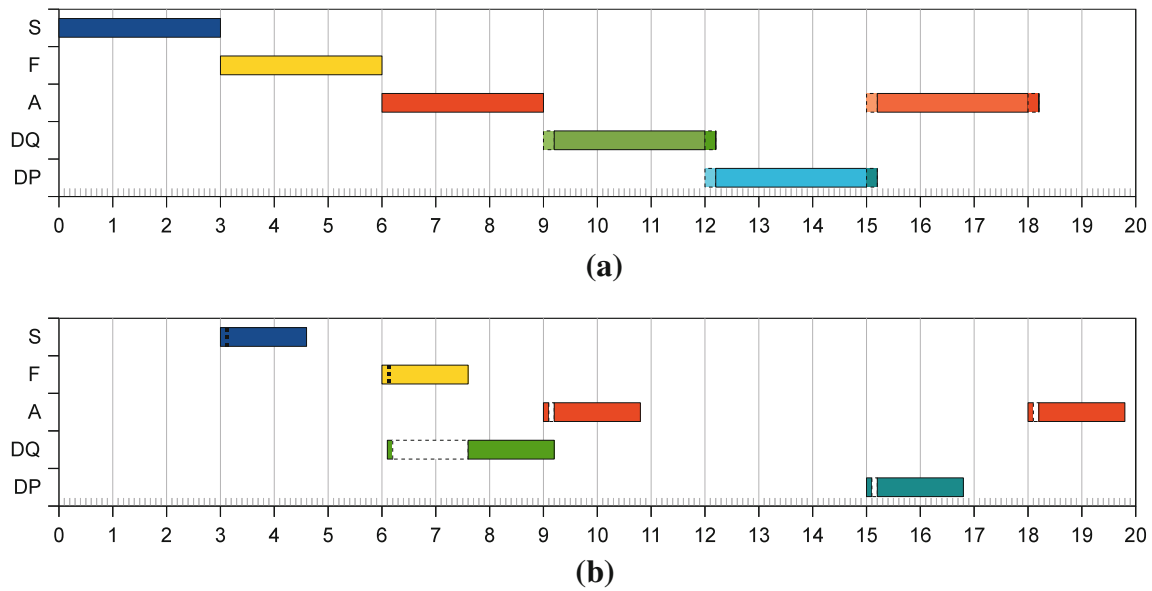


Fig. 15 Packet streams in HCS during the first two audio period. **a** Packet transmission scheme of the NAC for $L_m = 3$ and $0.1 \leqslant L_n \leqslant 1.6$. **b** Packet transmission scheme of the NAC for $L_m = 3$ and $0.1 \leqslant L_n \leqslant 1.6$

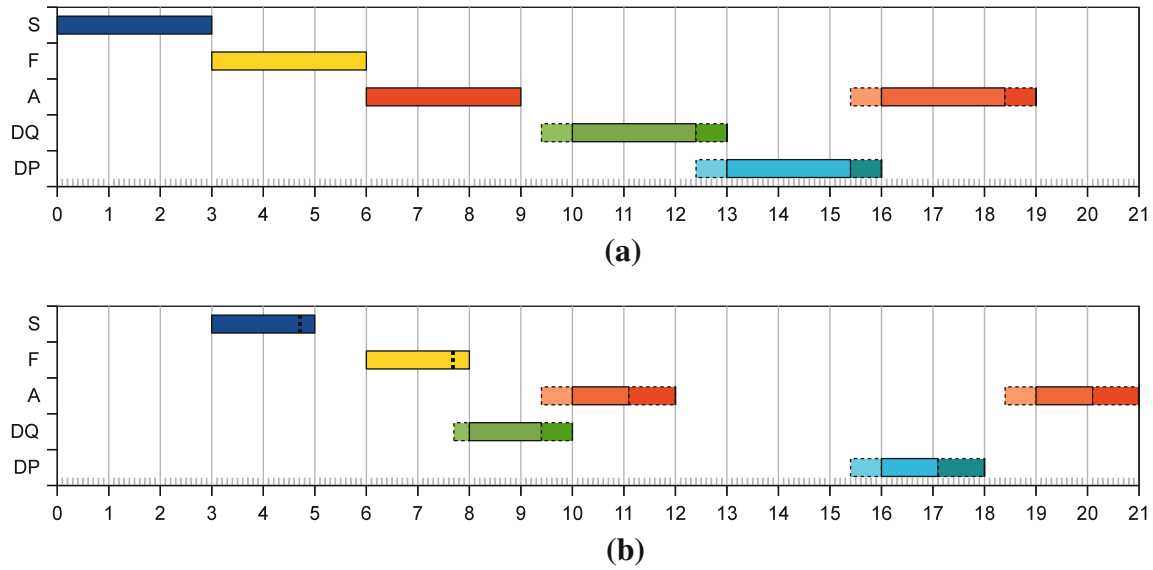


Fig. 16 Packet streams in HCS during the first two audio periods. **a** Packet transmission scheme of the medium for $L_m = 3$ and $1.7 \leqslant L_n \leqslant 2.0$. **b** Packet transmission scheme of the NAC for $L_m = 3$ and $1.7 \leqslant L_n \leqslant 2.0$

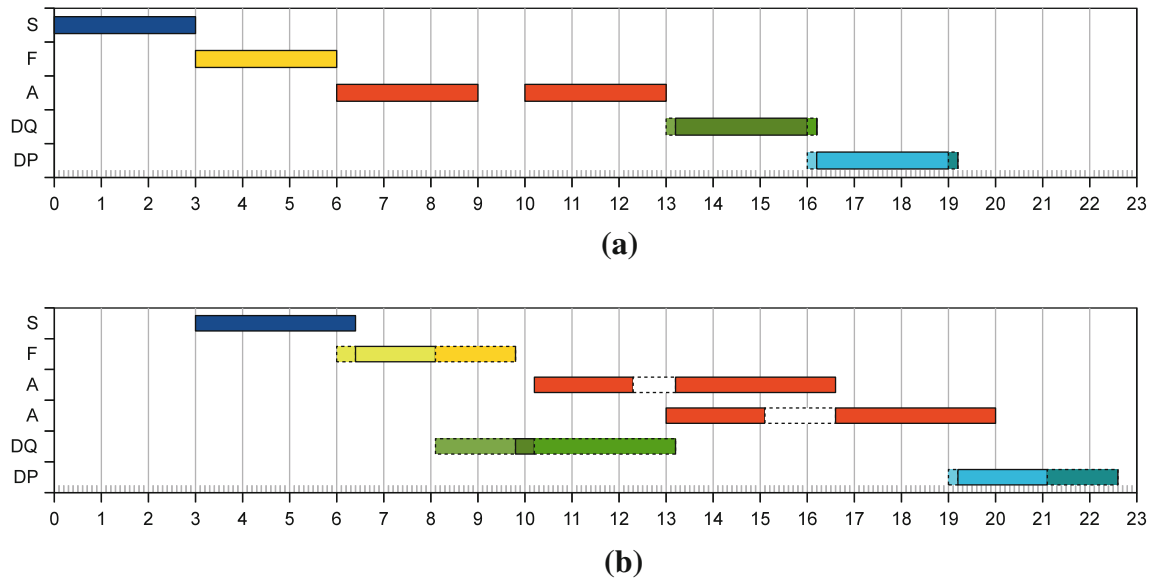


Fig. 17 Packet streams in HCS during the first two audio periods. **a** Packet transmission scheme of the medium for $L_m = 3$ and $2.1 \leqslant L_n \leqslant 3.4$. **b** Packet transmission scheme of the NLC for $L_m = 3$ and $2.1 \leqslant L_n \leqslant 3.4$

The arriving time of the first **A**-packet also varies with L_n, i.e., it goes from 9.1 to 10.6. Since $D_{rel} = 10$, it is possible that the first **A**-deadline is not respected. The second **A**-released at time 10 is transmitted through the medium at time ranging from 15 to 15.2 and so its arriving time ranges from 18 to 19.6 which is less than 20. As can be seen clearly from the figures, the delays of the **S** and **DQ** packets are the same, hence $off_n = 0$ and the second **A**-deadline is satisfied. From time 20–40, two other **A**-packets are generated and transmitted through the idle network without further delays from packets of higher priorities. Those **A**-deadlines are also satisfied and this transmission scheme is repeated at time 40, 80, etc.

For $1.7 \leqslant L_n \leqslant 2.0$, the transmission scheme looks similarly as in Fig. 16a and b. However, since the arriving time of the first **A** varies in the interval of [11.1,12] and the second **A** in [20.1,21], two consecutive deadline misses certainly happen and so the soft deadline constraint is violated.

When L_n grows larger than 2.0, i.e., $2.1 \leqslant L_n \leqslant 3.4$, the NAC processes the **DQ**-packet longer than previous cases. The **DQ** is forwarded to the medium at time ranging from 10.2 to 13.2 and so has to wait for the second **A**-packet, which was created at time 10 and took the transmission token, to complete its transmission. In this case, the second **A** gets transmitted before the **DQ** and arrives at the device before

Table 5 Analysis time in minutes (mins) under a hard/firm audio deadline constraint with a coarse/fine granularity

Constraint type	Granularity	Range	Figure	Verified points	Total analysis time (min)	Average analysis time (min)
Hard	Coarse	0–10	13a	100	445	4.5
Firm	Coarse	0–10	13b	100	440	4.4
Hard	Fine	0–5	13c	2,500	250	0.1
Firm	Fine	0–5	13d	2,500	260	0.1

the **DP** between time 15.1 and 20. Since the offset initially is zero and changed upon receiving the **DP**, the second **DQ** deadline is satisfied (Fig. 17a, b).

Finally, as soon as L_n goes beyond 3.5, i.e., $3.5 \leqslant L_n \leqslant 5.0$, the arriving time of the second **A** goes beyond 20 and causes the soft deadline constraint to be violated.

Table 5 shows the total analysis time taken to analyse HCS operating under a hard or soft constraint on audio deadline and with a coarse or fine granularity. The analysis is performed on $10 \times 10 = 100$ points in a coarse-grained parameter setting with the parameters ranging from 0 to 10 time units. For the fine-grained parameter setting, the grid is made of $50 \times 50 = 2{,}500$ points while the parameters vary in the smaller 0–5 time units range. The fine-grained analysis results in a much lower average time per point, and an overall lower total run-time. This is because the analysis is restricted to a region with smaller values of the latency, which result in a smaller time for UPPAAL to converge (find an error trace or establish the feasibility of the system). If we spanned the whole range with fine granularity, the computation time would easily become unaffordable. A coarse granularity should therefore be used to study large variations, while smaller regions can be analysed more finely.

Given the figures in Table 5, we can make the following concluding remarks. First, the system seems to have an acceptable performance when the verification is carried out with ground parameters (a few minutes to carry out the verification). Clearly, the performance worsens significantly when the complexity of the system grows (e.g., by inserting more NACs), but it remains within acceptable bounds for realistic sizes of the problem. Second, the exploration of the parameter space is cost affordable for a coarse granularity, but the non monotonic behaviour of the system could very easily lead to wrong conclusions. Third, a more exhaustive analysis (with a finer granularity) provides better guarantees and enables the designer to gain insight in the system behaviour. However, the cost of this analysis remains very high for a sufficiently large span of the parameters.

6 Related work

There are several methodologies that have been developed in the literature to address various issues in the analysis of real-time embedded systems. In our specific case, we are interested in determining timing properties of a complex system over a range of different parameters and, in particular, in computing an area of safe operation. This problem can be approached using formal analytical exhaustive methods or semi-formal simulation-based methods, and may take into account worst case or average case (under a certain statistics) behaviour.

One class of methodologies is based on the analytical representation of the timing characteristics of the system through a set of upper and lower bound functions on the rate of activity, or on the availability of computing resources. One example of this approach is Modular Performance Analysis (MPA) [8,9]. In MPA, tasks and input events are characterised in terms of the maximum and minimum activation or arrival rate over any interval length through a pair of functions. Computing platforms are represented in a similar way by modeling the least and largest amount of computation that can be supported in any time interval, and by selecting a policy for scheduling the input service requests. Analytical methods can be used through RTC-toolbox, a tool that implements the MPA methodology, to compute the arrival rate of output events, and the residual computation resources available under the chosen scheduling policy. These data can be used to determine the performance of the system, including end-to-end delays and buffer sizes [10]. Because of the degree of abstraction, this method is fast, but is limited by the relative expressive power of the model, which, for instance, is unable to support computation units whose behaviour depends on some state. To overcome this limitation, a recent evolution of MPA allows state-based scheduling policies to be modeled as timed automata, which are integrated into the system by translating the arrival rate information into generating (for the input) and observer (for the output) automata [11,12]. The use of this variant has, of course, an impact on the running time of the analysis. The method that we present in this work stands on the automata side of the spectrum, and offers higher accuracy in the representation (not just lower and upper bounds) and higher flexibility. One specific advantage of the direct use of automata is the ability to specify any behavioural property by employing appropriate error states, and is therefore more expressive than MPA. In our case study, for instance, it was important to faithfully

model the time synchronisation algorithm, since the clock drift directly affects the correctness of the system. In addition, the formal analysis conducted in UPPAAL is able to provide a witness error trace when a particular choice of the values of the parameters make the system unable to satisfy the required property, offering a valuable tool to qualitatively understand how to optimise the system. The downside is a much larger execution time.

Hierarchical event streams, used in the SymTA/S tool, are an alternative way of representing task activation patterns analytically [13]. The SymTA/S tool implements a variety of schedulability analysis techniques, based on event streams, and offers a number of scheduling algorithms to be selected in constructing an architecture platform for the system. The approach was successfully applied to analysing systems using binary search on the parameter space [14]. The analysis is fast, and the results comparable to what can be obtained using MPA. Likewise, the same arguments regarding flexibility apply, prompting us to use the direct automata-based specification.

Exhaustive and analytical methods are typically oriented towards considering the worst, or sometimes best, case behaviour of the system. An alternative approach is to consider the average case, under some statistics of the input patterns. A particularly interesting method is *statistical model checking*, which allows the designer to determine the probability that a certain property is satisfied. The model checking algorithm is computationally intensive, so that for large systems it is necessary to compute an abstraction. This has been achieved on an extended version of our case study using simulation-based methods [15] to analyse the behaviour of a large deployment of devices (in the order of a hundred) to provide a stochastic abstraction. In particular, the system has been modeled in the BIP framework [16] as a network of simplified timed atomic components, in a way similar to our present work (our model, in fact, derives in its simplest form from the BIP model). The method consists in selecting a particular pair of server and device, and simulating the entire system while concurrently measuring the timing behaviour of the data packets exchanged by the selected pair. These data are collected into a statistical model which is limited to the pair, but which accounts for the interaction with the other applications on the system and the effects of resource sharing (such as the network links). In other words, the statistical model depends on the context of use. Not surprisingly, the results are highly dependent on the particular device that is paired with the server. The statistical model checking technique is finally used on the context-dependent model to establish the probability that the desired properties are satisfied. While the approach is based on formal methods, the correctness of the results cannot be guaranteed because the model is derived by a finite number of simulations. However, techniques have been developed to estimate and bound the probability of making an error. In contrast to the statistical approach, the method presented in this work focuses on the worst-case behaviour. The advantage is that the results are guaranteed to be correct. To relax the requirements that all deadlines are met, we have developed alternative models, where the error state that identifies a missed deadline is not reached unless two packets in a row are lost. Similar models can be constructed to account for different average case behaviours, depending on the particular application under study.

An approach similar to ours, and from which we have taken inspiration, was proposed by Amnell et al. [17] with the tool TIMES. In TIMES, task activation are represented as finite automata, and appropriate schedulers can be selected to verify the schedulability of the system according to a scheduling policy and a set of worst case execution times and deadlines. The technique uses UPPAAL as an underlying engine. As opposed to TIMES, our approach is more generic since we are interested in analysing more than just schedulability in a potentially distributed settings. We therefore work directly at the level of the UPPAAL model to achieve the desired flexibility and expressiveness.

In our earlier work, we have shown how to analyse the parameter space of a similar system using exhaustive techniques based on bounded model checking [18]. While conceptually applicable to the extended model presented in this work, the computational complexity makes the approach impractical. Our current research is geared towards combining the method described in this paper with bounded model checking to achieve an efficient and exhaustive search of the continuous parameter space.

7 Discussion and conclusions

We have presented a case study, a modelling methodology and the analysis over a space of parameters of an industrial heterogeneous communication system. The system is characterised by complexity in terms of structure, and also in terms of functional and non-functional constraints, which must be carefully verified to assess the robustness of an implementation. To this end, we have studied the behaviour of the system over changing values of the parameters, to explore the regions of feasibility and their extension. Our approach is semi-formal. While we exhaustively analyse the property of the system for every point in a grid of the parameters, we do not study the behaviour of the system between these points. One could take a binary search-like approach to find the boundaries where the system changes from feasible to unfeasible. This may or may not work, depending on the system. In our case, for instance, the observed non-monotonic behaviour breaks a binary search algorithm and may lead to conservative or, in the worst case, optimistic answers where unfeasible points are classified as feasible. We have

also discussed alternative approaches, which can be used to trade off faster exploration with model expressiveness.

A fully parametric approach based on formal methods is desirable. Formal methods guarantee completeness (when the problem is decidable). By fully parametric we mean a method that accounts for parameters natively in their continuous space, and that understands their semantics. In our previous work, we have proposed a technique based on a symbolic representation of the timed automata and their parameters using NuSMV and tools for solving Satisfiability Modulo Theory (SMT) problems [18,19]. Instead of exploring points one by one, the method would symbolically search for an unfeasible point, and then generalise this point to a region by taking advantage of the boolean structure of the unfeasible trace that leads to the error state. The successive application of this technique eventually covers the unfeasible region. This method is not affected by non-monotonicity, and is decidable for certain classes of systems. The downside of the approach is a considerable computational complexity that makes the technique impractical in all but the simplest cases. The problem lies primarily in the search of a new unfeasible point and the trace that leads to the error state, while the process of generalisation of the trace is typically rather efficient. For this reason, one approach that we are experimenting with is to randomly search for unsatisfiable points using UPPAAL, which is relatively fast. Once one is found, the point is symbolically generalised to a region bounded by a polyhedron (linear constraints) and is added to the unfeasible set. The combination of the explicit method in UPPAAL, and the implicit method with NuSMV proves very effective, and is able to drastically reduce the run-time. The combined use of different tools has, however, some disadvantages. In our case, one has to ensure that the semantics of the model is preserved when moving from one tool to another. This is not always simple, as in UPPAAL clock values can only be stored in integer variables, while they can be stored in real variables in NuSMV. Synchronisations must also be dealt with carefully to make sure that the semantics of composition matches. The results that we have obtained with a simplified version of the model presented in this paper are encouraging. A fully automated flow is under investigation and development to properly manage models of higher complexity. Our research is particularly oriented towards the use of higher level languages, such as Hydi [20], to derive an equivalent representation in the different tools that matches the original semantics.

References

1. Liu, J.W.S.W.: Real-Time Systems, 1st edn. Prentice Hall, PTR, Upper Saddle River (2000)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci **126**(2), 183–235 (1994)
3. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell Int. J. Softw. Tools Technol. Transf. **1**, 134–152 (1997)
4. EADS Innovation Works: Case study on distributed heterogeneous communication systems (hcs), Germany. http://www.combest.eu/home/?link=Application1 (2009)
5. IEEE Standard 1588–2002: A precision clock synchronization protocol for networked measurement and control systems. November (2002)
6. Ramanathan, P.: Overload management in real-time control applications using (m, k)-firm guarantee. Parallel Distrib. Syst. IEEE Trans. **10**(6), 549–559 (1999)
7. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: Nusmv: a new symbolic model verifier. pp. 495–499. Springer, Berlin (1999)
8. Wandeler, E., Thiele, L., Verhoef, M., Lieverse, P.: System architecture evaluation using modular performance analysis: a case study. STTT **8**(6), 649–667 (2006)
9. Thiele, L., Chakraborty, S., Naedele, M.: Real-time calculus for scheduling hard real-time systems. Proc. Intl. Symposium Circ. Syst. **4**, 101–104 (2000)
10. Suppiger, U., Perathoner, S., Lampka, K., Thiele, L.: Modular performance analysis of large-scale distributed embedded systems: an industrial case study. Computer Engineering and Networks Laboratory, ETH Zurich, TIK Report 330, November (2010)
11. Lampka, K., Perathoner, S., Thiele, L.: Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In: EMSOFT'09: Proceedings of the 7th ACM international conference on embedded software, pp. 107–116. ACM, New York, NY, USA (2009)
12. Lampka, K., Perathoner, S., Thiele, L.: Analytic real-time analysis and timed automata: A hybrid methodology for the performance analysis of embedded real-time systems. Des. Autom. Embed. Syst. **14**(3), 193–227 (2010)
13. Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., Ernst, R.: System level performance analysis-the SymTA/S approach. IEEE Proc Comput. Digit. Tech. **152**(2), 148–166 (2005)
14. Racu, R., Jersak. M., Ernst, R.: Applying sensitivity analysis in real-time distributed systems. In: RTAS '05: Proceedings of the 11th IEEE real time on embedded technology and applications symposium, pp. 160–169. IEEE Computer Society, Washington, DC, USA (2005)
15. Basu, A., Bensalem, S., Bozga, M., Caillaud, B., Delahaye, B., Legay, A.: Statistical abstraction and model-checking of large heterogeneous systems. In: Hatcliff, J., Zucca, E. (eds.) Formal techniques for distributed systems, Joint 12th IFIP WG 6.1 international conference, FMOODS 2010 and 30th IFIP WG 6.1 international conference, FORTE 2010, Amsterdam, The Netherlands, June 7–9, 2010. Proceedings, ser. Lecture notes in computer science, vol. 6117, pp. 32–46. Springer, Berlin (2010)
16. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time systems in BIP. In: SEFM06, pp. 3–12. Pune, India, September (2006)
17. Amnell, T., Fersman, E., Mokrushin, L., Pettersson, P., Yi, W.: TIMES: a tool for schedulability analysis and code generation of real-time systems. In: FORMATS, pp. 60–72 (2003)
18. Le, T.T.H., Palopoli, L., Passerone, R., Ramadian, Y., Cimatti, A.: Parametric analysis of distributed firm real-time systems: a case study. In: Proceedings of the 15th IEEE international conference on emerging technologies and factory automation (ETFA10), pp. 1–8. Bilbao, Spain, September 13–16 (2010)
19. Cimatti, A., Palopoli, L., Ramadian, Y.: Symbolic computation of schedulability regions using parametric timed automata. In: Real-Time Systems Symposium, pp. 80–89. Dec 3 (2008)
20. Cimatti, A., Mover, S., Tonetta, S.: Hydi: A language for symbolic hybrid systems with discrete interaction. In: EUROMICRO-SEAA, pp. 275–278 (2011)