Re-verification of a Lip Synchronization Algorithm using robust reachability

Piotr Kordy
Twente University of Twente,
Drienerlolaan 5,
7522 NB Enschede,
The Netherlands
kordy@cs.utwente.nl

Rom Langerak
Twente University of Twente,
Drienerlolaan 5,
7522 NB Enschede,
The Netherlands
langerak@cs.utwente.nl

Jan Willem Polderman
Twente University of Twente,
Drienerlolaan 5,
7522 NB Enschede,
The Netherlands
j.w.polderman@math.utwente.nl

The timed automata formalism is an important model for specifying and analysing real-time systems. Robustness is the correctness of the model in the presence of small drifts on clocks or imprecision in testing guards. A symbolic algorithm for the analysis of the robustness of timed automata has been implemented. In this paper we re-analyse an industrial case lip synchronization protocol using the new robust reachability algorithm. This lip synchronization protocol is an interesting case because timing aspect are crucial for the correctness of the protocol. Several versions of the model are considered, with an ideal video stream, with anchored jitter, and with non-anchored jitter.

1 Introduction

Timed automata [2] is a popular and successful formalism to analyse real-time systems. Timed automata are automata extended by clock variables that can be tested and reset. Numerous real-time systems have been specified and analysed by the tool UPPAAL [5, 17] and the approach can be said to be mature and industrially applicable.

However if we want to implement a system robustness becomes an issue. We need to know if the system is resilient with respect to small perturbations. Timed automata in its original form are a bit "too precise". Several works exists suggesting alternative semantics of timed automata, that takes into account perturbations. In particular, skewed clocks automata from [1] can have arbitrary rates for clocks, "tube languages" from [9, 11] deal with open sets of trajectories, "perturbed" timed automata from [14] are subjected to an infinitesimal noise, implementable timed automata from [19, 20] should be implementable using discrete clocks etc.

However successful semantics should be applicable in practise. In this paper we are interested in the work that was initiated by Puri [13]. He considered drifting clocks and showed that timed automata models are not robust with respect to safety properties, meaning that a model proven to be safe under the standard ideal semantics might not be safe even if clocks drift by an arbitrarily small amount. The region based algorithm is proposed to calculate set of states that are reachable for *any* clock drift. Puri's approach was extended by introduction of stable zone [7], which made it possible to implement much faster algorithm that can be used in practice.

To check the new algorithm performance the best way is to apply it to industrial case study. On the UPPAAL homepage [17] there are number of case studies in which UPPAAL tool has been applied. We investigated several of them and we chose the case study where lip synchronisation algorithm is analysed. This algorithm is used to synchronize multiple information streams sent over a communication network, in this case, audio and video streams of a multimedia application. We chose this case study mainly because timing is an important aspect of synchronisation and such algorithm can be sensitive to the small disturbances on the clock drift.

Submitted to:
FMA 2009

Structure of the paper The rest of the paper is organised as follows. Section 2 introduces the lip synchronisation problem. Section 3 introduces the modelling formalism and tool used in the analysis of lip synchronisation protocol. Section 4 provides a description of a model of a lip synchronisation algorithm. Section 5 presents the verification results and Section 6 gives a concluding discussion.

2 Lip Synchronisation Problem

The problem of lip synchronisation has been present in the literature[16, 4]. Here we present a brief description, for more detailed description look in [6]. Other descriptions in line with our description can be found in [16] and [15]. In this paper we consider the problem of synchronising of audio and video streams. We consider a scenario when audio and video are transmitted as separate streams that needs to be synchronised at the sink.

The overview of the basic configuration can be seen in Figure 1. There are two stream sources: on for sound and one for video. Streams arrive at a presentation device. We need to ensure that both streams play synchronised within certain level of tolerance. This is the problem that lip synchronisation algorithm solves.

The algorithm is implemented using several components: *sound* and *video* managers, and a *controller*. Communication between components is done using signals. When presentation device receives sound packet it sends a *savail* signal to the *Sound Manager*. At an appropriate moment the Sound Manager sends *spresent* to the Presentation Device to indicate that the packet should be played. The *Video Manager* has similar behaviour and uses signals *vavail* and *vpresent*. The *Controller* contains the main body of the algorithm. It receives signals *sready* and *vready* from the managers. The signals indicate that sound and/or video packets can be presented. The Controller decides if it is the correct time to play the packet. Confirmation is done using *sok* and *vok* signals, respectively. If it is not possible to synchronise, the Controller signals an error and enters an error state.

The requirements for acceptable synchronisation between the two streams are the following:

- The time granularity is 1 millisecond.
- A sound packet is presented every 30 milliseconds and no jitter is allowed.
- Optimally, a video packet should be presented every 40 milliseconds. However we allow some margin of error:
  - video frames may precede sound by up to 15 milliseconds and may lag up to 150 milliseconds.
  - we allow a 5ms jitter, that is, a video package may be no less than 35 ms late and no more than 45 ms away from ideal presentation time (Anchored Jitter) or from previous packet (Non-anchored Jitter)

3 Modelling Formalism

3.1 Timed Automata

For our purposes we use the existing timed automata model from [6]. The modelling formalism is based on a network of timed automata. The network of timed automata consist of the parallel composition

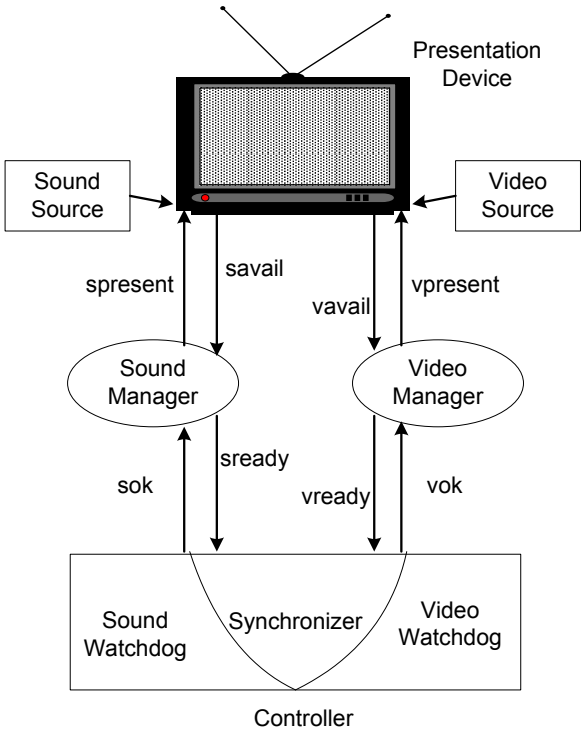


Figure 1: Overview of the structure of the lip synchronisation system

of a number of timed automata, and a configuration. A timed automaton is an automaton consisting of locations and edges that is extended with real valued variables called clocks.

Edges and locations are labelled. The labels of the edge may consists of several optional components: a guard, a synchronisation label, a set of clocks to reset, and assignments to integer variables. The guard on clocks and/or on data variables expresses under which conditions we are allowed to take a transition. If there is no guard, the condition is interpreted as true. Because later we consider a so called robust semantics, we limit guards on clocks to the form : $x \leq c$ or $x \geq c$ where x is a real valued clock and c is natural number. We also allow a formula that is a combination of the above terms using logical *and*. When we take a transition we may perform a synchronisation. The synchronisation label must be synchronised with its counter part. The synchronisation rules are similar as in CSS [12]. When there is no synchronisation, a label is interpreted as an internal action (similar to τ-actions).

The labels of locations consist of the name of a location, an optional invariant, and can be marked as *committed* or *urgent*. An invariant is a constraint on clocks, indicating how long we can stay in a location. It is similar to guard but only upper bound constraints are allowed. When a location is marked as committed, we have to leave this location without any delay or any interleaving actions. This is useful to ensure atomicity of sequence of transitions. When a location is urgent time is not allowed to pass in that location.

The configuration consists of the names of timed automata composing the system, global variables, and channels. Synchronisation happens through channels and synchronisation labels are names of channels. Channels can be *urgent*. When a channel is urgent, we have to take that transition as soon as possible that is without delay. There can be no guards on edge with urgent channel.

The state of timed automaton is of the form $(\bar{q}, v)$ where $\bar{q}$ is a control vector and v is the clock

valuation. The control vector shows the current location for each timed automata in the network and the clock valuation indicates value of each clock and integer variables. The initial state consists of initial locations for each timed automaton and all clocks and variables equal to value 0. From a state it is possible to take two types of transitions: *delay* and *edge* transition. When we take a delay transition, all clocks are progressing at the same speed within the values allowed by the location invariants. An edge transition can be internal or a synchronisation. An internal transition can occur when the network is at the location in which it can take an edge with no synchronisation label. The guard must be satisfied by the current clock valuations. A synchronisation transition occurs when two edges can synchronise over complementary synchronisation labels. Guards of both edges must be satisfied.

3.2 Model Checking

The continuous time leads to infinitely many states. Fortunately as noted in [2] similar states can be grouped into *regions*. However region automaton is not the most efficient representation of the state space of a timed automaton. It suffers from a combinatorial state explosion which is dependent on the size of the constants used. *Zones* are used as a more efficient representation of the state-space [8][10][21] as they represent the state space in a more aggregated way. In the tool UPPAAL [5] more effective zone based algorithm is used.

The UPPAAL tool is able to check for reachability properties. Those properties are of the form:

$$\varphi ::= \forall[]\beta \mid \exists <> \beta \qquad\qquad \beta ::= a \mid \beta_1 \wedge \beta_2 \mid \neg\beta \mid \beta_1 \Rightarrow \beta_2$$

where a is an atomic formula being either an atomic clock (or data) constraint or a component location $(A_i \text{ at } l)$. Atomic clock (data) constraints are integer bounds on individual clock (data) variables (e.g. $1 \leq x \leq 3$).

Intuitively for $\forall[]\beta$ to be satisfied all reachable states must satisfy β. For $\exists <> \beta$ to be satisfied some reachable state must satisfy β.

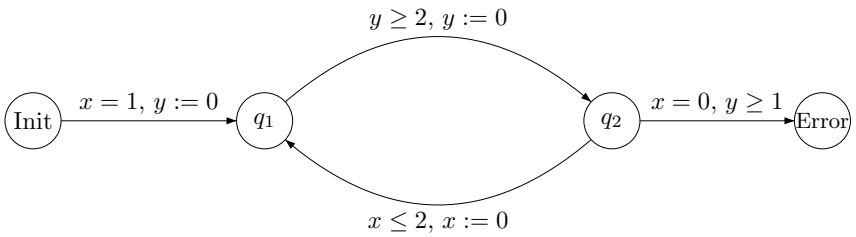
3.3 Robustness Problem

Clocks in a timed automata network are synchronous. Puri [14] has shown that this assumption is not robust to even infinitely small clock drifts. In short, it means that we can reach states that are not reachable in normal semantics for *any* value of the clock drift. He proposed a new reachability semantics for timed automata and we will call it the *robust semantics*. The idea is to have a parametrised reachability, where a parameter is allowed clock drift. In normal reachability, when time progresses by t time units, the new clock valuation is $v+t$. When we allow clock drift parametrised by ε, the new clock valuation can ε small differences between clocks. Formally

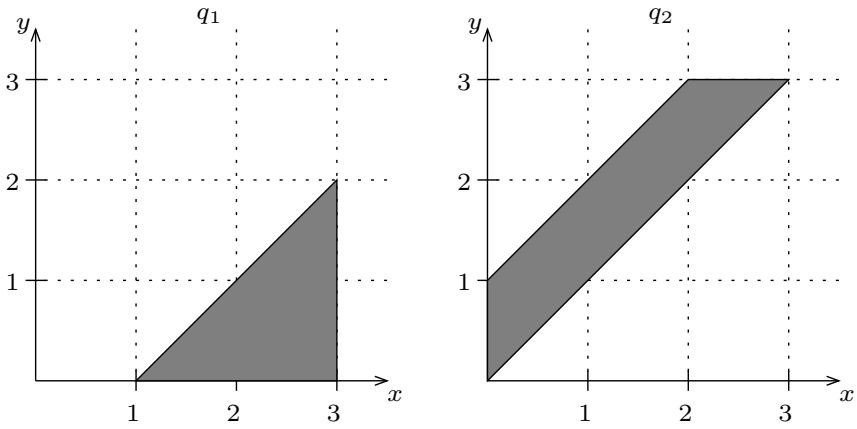
$$v'(x_i) - v(x_i) \in [(1-\varepsilon)t, (1+\varepsilon)t] \text{ for } i = 1, \ldots, n$$

where n is number of clocks, v' is a valuation after the time transition and v is a clock valuation before the time transition. Let $\mathsf{Reach}_\varepsilon(s_0)$ denote the reachable set of states from the initial state s_0 in the robust semantics. Unfortunately parametric model checking of timed automata with three clocks and only one parameter is known to be undecidable [3][18]. What Puri proposes is the reachability when the drift ε is infinitely small. Formally

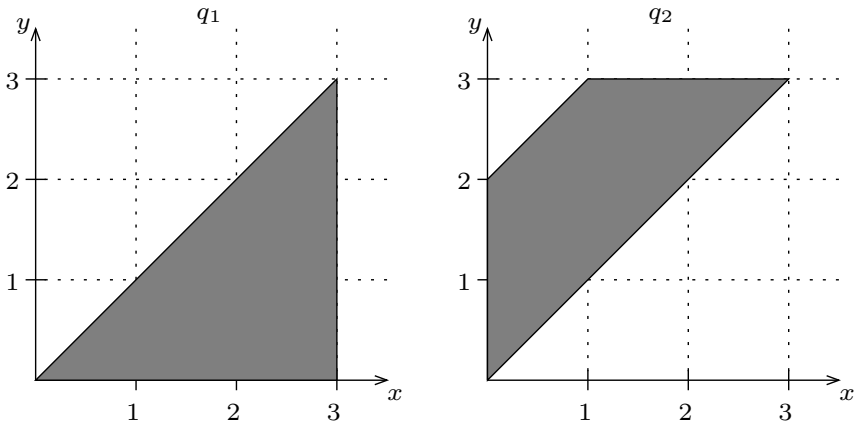
$$R_{\varepsilon\to 0}(s_0) = \bigcap_{\varepsilon>0} \mathsf{Reach}_\varepsilon(s_0)$$



(a) A timed automaton



(b) Reachable set of states in the normal semantics



(c) Reachable set of states in the robust semantics

Figure 2: Three subfigures.

Puri shows that calculating $R_{\varepsilon\to 0}(s_0)$ is decidable. He proposes an region based algorithm.

It may seem that $R_{\varepsilon\to 0}(s_0)$ and $\mathsf{Reach}(s_0)$ are the same since ε is small, but this is not the case. Consider the following example shown in Figure 2(a). This timed automaton has two clocks x and y and four locations. From location Init we can only go to location q_1 and the value of the clocks will be $x = 1$ and $y = 0$. In the precise semantics, following the cycle between locations q_1 and q_2, we will get the reachable set of states $\mathsf{Reach}(\mathscr{S})s_0$ depicted in Figure 2(b). We want to avoid Err location. The Err location is not reachable for both $\alpha = 2$ and $\alpha = 3$.

Now we consider the case for the robust semantics. Let e_1 be edge from location q_1 to q_2 and e_2 edge from location q_2 to q_1. Notice that for any $0 \le \beta \le 1$ we the following sequence of transitions is possible: $(q_1; x = \beta, y = 0) \overset{2-\beta}{\to}_{\varepsilon} \overset{e_1}{\to}_{\varepsilon} (q_2; x = 0, y = 2-\beta+\varepsilon) \overset{\beta-\varepsilon}{\to}_{\varepsilon} \overset{e_2}{\to}_{\varepsilon} (q_1; x = \beta-\varepsilon, y = 0)$. Hence if we cycle sufficient number of times for any $\varepsilon > 0$ we can reach state $(q_1; x = 0, y = 0)$ which is not reachable in the normal semantics $\mathscr{S}$. Thus the Err location is robustly reachable for $\alpha = 2$. This shows that the normal semantics is not robust with respect to small clock perturbations. Even small changes in the clock drift may lead to a dramatical change in the behaviour of a system. We avoid the Err location only in the normal semantics but not in the robust semantics. We say that such safety property is non-robustly satisfied. If a system has non-robustly satisfied property it is not implementable, because its correctness depends on the mathematical idealization of the normal semantics.

3.4 Verification Tool

Puri proposed an algorithm to calculate $R_{\varepsilon\to 0}(s_0)$ that is based on regions. Basically the algorithm finds regions that are on the cycle in the region graph. Such regions have the property that we can drift form any point to any other point in that region in robust semantics. When a part of such a region is encountered in the search, the whole region is added to the reachable set of states. In [7] the notion of a *stable zone* has been introduced. Basically the stable zone has the same property that we can drift from any point in the stable to any other point in that stable zone. Thus this is a good starting point for the zone based algorithm to calculate robust reachability. Together with the Aalborg university the prototype tool is being developed based on UPPAAL 4.1.1 and it is used in this paper.

4 The Model

In this section UPPAAL model is presented. It follows the specification given in [6] which in turn was derived from the specification given in LOTOS [15]. The model represents the specification of the video and sound managers and the synchroniser from Figure 1

The model is shown in Figure 3. Sound and Video Managers are modelled by automata *VideoMgr*, *SoundMgr*, *VideoWdg*, *SoundWdg* and *UrgMon*. The Synchronizer consists of *Synch*, *VideoSynch*, *SoundSynch* and *SoundClock*. The external environment that is the sound and video streams are modelled by *VideoStr* and *SoundStr*. We briefly discuss the components.

The stream managers Both stream managers are quite simple. After receiving a signal *vavail* or *savail* that the video or sound packet is available, they forward the signal immediately to the synchroniser using *vready* and *sready*. The immediacy is ensured by marking the locations *vm2* and *sm2* as committed. Next the manager waits for a confirmation from the controller (the confirmation signal comes from the

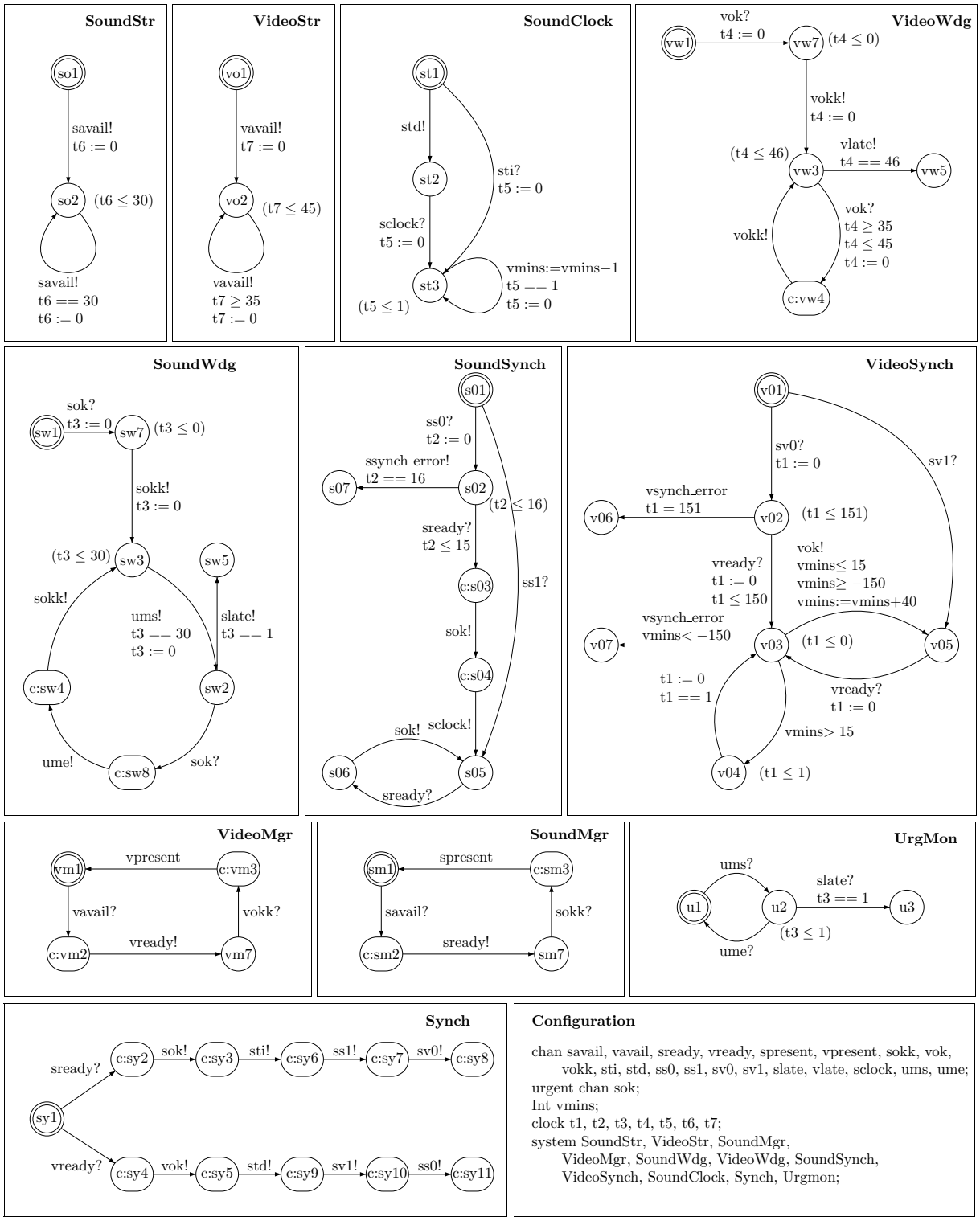


Figure 3: A model of Lip Synchronisation Protocol

watchdogs) that the packet can be played. This is done using signals *vokk* and *sokk*. The confirmation is immediately forwarded to the presentation device using signals *vpresent* and *spresent*. Since the presentation device is not modelled, those actions are internal.

The watchdog timers The role of a watchdog timers is to ensure that the time between presentations of subsequent media packets is within certain time bounds. We discuss the video watchdog. The timing requirement is that consecutive video packets are played between 35 ms and 45 ms. Initially the watchdog waits for the first packet to arrive (signal *vok*) and sends immediately the confirmation to the video manager using signal *vokk*. We ensure that no time passes between *vok* and *vokk* signals. The signals *vok* and *vokk* are in a way a complex signal that allows synchronisation between *VideoSynch*, *VideoMgr* and *VideoWdg*. After presenting the first packet the time is measured until the next packet arrives. In order to ensure proper timing of the presentation of the packet, the transition leaving location vw3 is guarded by $35 \leq t4 \leq 45$. If *vok* does not occur before 45 ms passes, *vlate* error is given.

The *SoundWdg* is slightly more complicated because we must ensure that sound packets are played exactly every 30 ms. Similarly to the video watchdog it waits for the confirmation (signal *sok)* that the first packet should be presented and relays the signal to the *SoundMgr* using the signal *sokk* without time delay. After that the clock t3 is used to measure the time between consecutive presentations of the sound packets. To ensure that exactly 30 ms passes between sound packets, *UrgMon* is used and signal *sok* is marked as urgent. If a sound packet is not available in 30 ms the *slate* error is generated.

The synchroniser The role of a *Synch* is to initialise the other automata. Depending on whether a sound or a video packet arrives first, automata can be initialised in two ways. If signal *vready* or *sready* arrives then we confirm that the packet can be presented (signal *vok* or *sok*) and initialise *SoundClock* (signal *std* or *sti*) then initialise *VideoSych* (signal *sv1* or *ss1*) and at last we initialise *SoundSynch* (signals *sv0* or *ss0*). Note that all locations except sy1 are committed to ensure that initialisation is done immediately.

The sound clock The *SoundClock* is a discrete clock that ticks every millisecond. It is started at the moment the first sound packet arrives. It can be initialised by signal *sti* if the sound packer is first or by combination of *std* and *sclock* signals if video packet arrives first.

The clock is used to compute the skew between sound and video streams. The skew is stored in *vmins* variable. Every time the clock ticks it is decreased by one.

The sound synchroniser The *SoundSynch* can be initialised it two ways. If a sound packet is first it receives *ss1* signal and starts the repeating behaviour immediately. If a video packet is first then it checks if there is a synchronisation error - it can happen only when the first sound packet does not arrive within 15 ms after the first video packet. This is the requirement of the lip synchronisation. After the fist sound packet arrives it initialises *SoundClock* through the *sclock* signal and starts the repeating behaviour

The repeating behaviour is very simple. If it receives signal *sready* that the sound packet has arrived, it send a signal *sok* that it can be presented.

The video synchroniser The *VideoSynch* is quite complex. Similarly as *SoundSynch* it can be initialised in two ways. If the video packet arrives first it goes immediately to the repeating behaviour through signal *sv1*. If sound packet arrives first, it checks if a video packet is received within 150 milliseconds but not earlier than 15 milliseconds from the sound packet. If more than 150 milliseconds has passed then *vsynch_error* is generated.

In the repeating behaviour *VideoSynch* checks if there is too much skew between sound and video packets. After receiving *vready* signal, it checks the lip synchronisation requirement immediately (the t1$\leq$ 0 invariant). Now we have three possibilities:

- The video presentation is more than 150 milliseconds later than the sound presentation. This is true when *vmins* is less than -150. In such case *vsynch_error* is generated.
- The video is more than 15 ms too early with respect to the sound packet. This is the case if *vmins*$>$ 15. In such case the presentation of the video frames are postponed. We enter a state where we are forced to wait one millisecond. After that we check the synchronisation requirement again.
- The video and the sound packets are sufficiently synchronised. In such case we send a signal *vok* to present a video packet and we update the *vmins* variable.

The media streams The informal specification of the protocol does not make any assumptions about the streams. Several possible streams are modelled and are further described in Section 5.

5 Verification

5.1 Verified properties

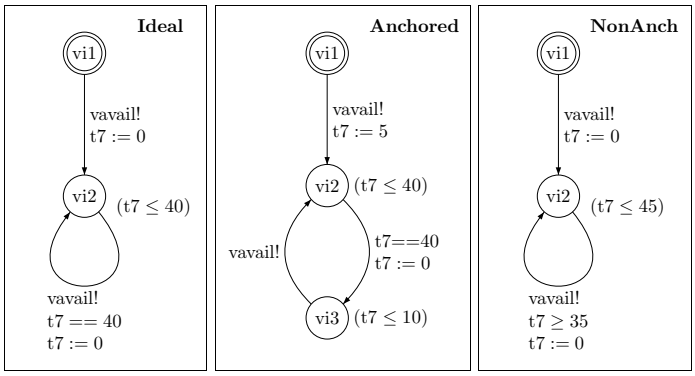


Figure 4: Three variations of video stream

We have followed [6] and we assumed that the sound stream is ideal and arrives every 30 ms. The perturbations may affect the video stream. There are three possible video streams that are investigated:

- An *ideal* video stream that delivers a video frame every 40 ms.
- A video stream with *anchored jitter* that have a rate of 40 ms and a variation of $\pm$5 ms.
- A video stream with *non-anchored jitter* where the variability between each two consequent frames is minimally 35 ms and maximally 45 ms.

Figure 4 shows automata representing different variations of the video stream.

Another variation that was investigated is the initial delay of video and sound streams. The first option is that the starting time of streams is left unspecified, and the other possibility is that both streams start at the same time.

The verification is done using error location reachability. Each error location reachability was done using normal and the robust semantics. The reachability properties are all of the form:

$$E <> A.l \ \ and \ \ not\,(B_1.l_1 \ \ or \ \ldots \ B_n.l_n)$$

The answer to such a query will be positive if there exists a path in timed automata network which will eventually reach location l in A but all locations l_i in B_i will be avoided. The location l will be the error location we are checking. The second part is to ensure that timed automata network did not reach another error location as this might have caused another error location to be reachable. The following error location have been modelled and checked for reachability:

- Initial sound synchronisation error in the *SoundSynch* (location s07)
- Initial video synchronisation error in the *VideoSynch* (location v06)
- Video synchronisation error in the *VideoSynch* (location v07)
- Video late error in the *VideoWdg* (location vw5)
- Sound late error in the *SoundWdg* (location sw5)

5.2 Results

We have implemented the algorithm for the robust semantics in a prototype tool based on UPPAAL 4.1.1. The normal semantics reachability analysis is done using UPPAAL 4.1.1. Our implementation at best can be as good as a depth first search for the normal reachability. Thus all the results presented here are run using depth first search. Experiments were performed on a PC with an AMD 1.2 GHz processor with 768MB of RAM. In [6] the state space was reduced by marking the error locations as committed. We have not done this optimisation.

Error location	Ideal		Anchored		Non-anchored	
Init Sound Synch(s07)	T	0.5	T	0.1	T	0.2
Init Video Synch(v06)	T	1.5	T	56.7	T	55.7
Video Synch (v07)	F	3.3	T	57.9	T	26.5
Video Late (vw5)	F	3.3	T	0.2	F	55.5
Sound Late (sw5)	F	3.2	F	61.8	F	57.2
Init Sound Synch(s07*)	T	0.1	T	0.2	T	0.2
Init Video Synch(v06*)	T	7.4	T	4613.3	T	1260.1
Video Synch (v07*)	T	3378.2	T	3168.4	T	674.9
Video Late (vw5*)	F	5636.4	T	7.5	F	5834.5
Sound Late (sw5*)	F	5378.2	F	5591.2	F	5724.4

Table 1: Verification results for streams with possible initial delay for both normal and robust semantics (marked with *)

Table 1 gives the results of the verification of the lip-synchronisation protocol for the various reachability properties. In the leftmost column we have a type of error that can occur. In the case of the error location being not reachable we mark it with F and if the error location is reachable we put T. In the second column we have a verification time given in seconds.

We can see that for all kinds of video streams, the initial sound and video synchronisation errors can occur. This can be explained by the fact that video or sound stream can postpone sending packet. This allows for the gap between sound and video packet to be arbitrarily long and reaching error location.

The anchored and non-anchored video streams can encounter video synchronisation error and can reach location vw07. In both cases it is enough to wait as long as possible but avoiding initial video synchronisation error and then the gap between sound and video packet can be enlarged due to allowed jitter.

Only in the case of the video stream with anchored jitter video frames can be late. In anchored jitter maximal gap between two consecutive packets is 50 ms. This is 5 ms more

than allowed gap thus video frames can be late.

Error location	Ideal		Anchored		Non-anchored	
Init Sound Synch(s07)	F	0.2	F	0.4	F	86.1
Init Video Synch(v06)	F	0.2	F	0.5	F	85.1
Video Synch (v07)	F	0.2	F	0.5	T	36.8
Video Late (vw5)	F	0.2	T	0.2	F	83.1
Sound Late (sw5)	F	0.2	F	0.5	F	81.8
Init Sound Synch(s07*)	F	81.8	F	153.2	F	178.4
Init Video Synch(v06*)	F	391.1	F	397.2	F	357.9
Video Synch (v07*)	T	275.4	T	293.7	T	244.2
Video Late (vw5*)	F	394.4	T	9.2	F	385.4
Sound Late (sw5*)	F	391.5	F	385.3	F	401.2

Table 2: Verification results for streams without initial delay for both normal and robust semantics (marked with *)

The results where both streams are forced to start at the same time are shown in Table 2. The presentation format is the same as in Table 2. In such a situation the initial synchronisation errors are not reachable. In the case of an ideal video stream we do not encounter any errors. In the case of an anchored video stream again the video can be late. The reason is the same as previously. The non-anchored video stream can lead to out of video synchronisation error. This is because the gap can accumulate over time.

The second part of Table 1 and Table 2 shows the result for the robust semantics. They are marked with * next to the location name. It is worth mentioning here that if some location is reachable in the normal semantics then it is reachable in the robust semantics. The main difference between normal and robust semantics is that a video synchronisation error is always possible for all kinds of video stream, with or without allowed initial delay.

We try to explain this for the case of ideal video stream without initial delay, as all other cases are less restrictive. The variable vmins is decreased every millisecond. The timing is provided by the clock t5. In the case of an ideal video stream, every 40 ms (ensured by clock t7) a video packet is sent. If vmins is small enough, so we do not have to enter location v04 and vmins is increased by 40. Thus over time vmins oscillates around the same base value. Now assume that clocks t5 and t7 desynchronise by ε every millisecond. For sufficiently many cycles the base value over which vmins oscillates can be changed up or down. If vmins is too large it will be remedied by visiting location v04, but no such mechanism exists when vmins is getting smaller. Thus finally we will reach v07 and video synchronisation error will occur. Other types of errors use the clocks that cannot accumulate the drift because their reset time is

synchronised with the signals. So the verification results from normal and robust semantics are the same.

The conclusion is that if we play video long enough we are not able to guarantee that the protocol will not desynchronise, no matter how precise clocks we have. We are only able to guarantee proper lip synchronisation for a playback with limited time.

Deadlocks In [6] in addition the deadlock detection is done. We do not do deadlock detection as current status of the theory does not allow the tool to detect deadlocks robustly. The main limiting factor is the fact that we do not allow the guards to be strict. To do the deadlock detection we need to detect when we reach state that cannot leave through any transition. For that we need to complement guards on the edges and that introduces strict inequalities. The brief manual analysis of the specification reveals that new deadlocks can be reached in robust semantics. For example non-anchored video can send video packet at 45 ms but because of slight desynchronisation clock t4 have value $45 + \varepsilon$ thus edge leading to location vw4 is not enabled any more.

As a side note let us mention that in [6] authors report one deadlock and they expected the other deadlock that should be detected by the UPPAAL, but full state search did not revealed it. It appears that the reason for not detecting the deadlock must have been the early imperfection of the UPPAAL tool, as the current version 4.1.1 detect both deadlocks.

6 Conclusions

We have re-verified a lip synchronisation algorithm using robust semantics. The original algorithm has been previously model checked using UPPAAL [6] and has been presented in a number of different formalisms [16, 15, 4].

The verification results using robust semantics are slightly different from normal semantics. The choice of case study was done to maximise probability of different results so this was something anticipated. The robust reachability analysis allowed us to identify the problem with the lip synchronisation algorithm. For a continuous playback we are not able to make clock precise enough to ensure that video and sound do not become desynchronised. The sound and video can stay synchronised only for a limited time, and this time is depending on the precision of the clocks.

The verification of lip synchronisation algorithm gave us also the possibility to evaluate the performance of the robust reachability algorithm. The verification of a lip synchronisation using robust semantics takes significantly more time than verification using normal semantics. The main reason is that the model uses variable vmins as a kind of discrete clock which divides state space into small pieces. This forces our algorithm to add many stable zones, which is expensive.

The limitation of the algorithm is inability to detect deadlocks. In the case of industrial case study this is important feature. We believe that ability to detect deadlocks would identify more problems - mostly connected to the way time-out is modelled. Another limitation is that it is not possible to use strict guards. In the context of robustness where we allow small clock drifts, we believe that differentiating between strict and non-strict guards is not essential feature. Unfortunately this feature is needed for deadlock detection. This will be a main interest of our future work. At the moment only reachability properties can be analysed. Thus the future research can be directed to extending the algorithm with the possibility to check liveness properties.

References

[1] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger & Pei-Hsin Ho (1993): *Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems*. In: *Hybrid Systems*, *LNCS* 736. Springer, pp. 209–229.

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

[3] Rajeev Alur, Thomas A. Henzinger & Moshe Y. Vardi (1993): *Parametric real-time reasoning*. In: *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. ACM, New York, NY, USA, pp. 592–601.

[4] Ahmet F. Ates, Murat Bilgic, Senro Saito & Behçet Sarikaya (1996): *Using Timed CSP for Specification Verification and Simulation of Multimedia Synchronization*. *IEEE Journal on Selected Areas in Communications* 14(1), pp. 126–137.

[5] Johan Bengtsson & Wang Yi (2003): *Timed Automata: Semantics, Algorithms and Tools*. In: *Lectures on Concurrency and Petri Nets*. Springer, pp. 87–124. Available at `http://springerlink.metapress.com/openurl.asp?genre=article%&issn=0302-9743&volume=3098&spage=87`.

[6] H. Bowman, G. Faconti, J.-P. Katoen, D. Latella & M. Massink (1998): *Automatic Verification of a Lip-Synchronisation Protocol Using Uppaal*. *Formal Aspects of Computing* 10(5-6), pp. 550–575. Available at `http://www.springerlink.com/content/21366fhg6r7xt15b/`.

[7] Conrado Daws & Piotr Kordy (2006): *Symbolic Robustness Analysis of Timed Automata*. In: *Formal Modeling and Analysis of Timed Systems*, *Lecture Notes in Computer Science* 4202. Springer Berlin / Heidelberg, pp. 143–155. Available at `http://www.springerlink.com/content/l8026102m7jv3j32/`.

[8] David L. Dill (1990): *Timing assumptions and verification of finite-state concurrent systems*. In: *Automatic Verification Methods for Finite State Systems*, *Lecture Notes in Computer Science* 407. Springer Berlin / Heidelberg, pp. 197–212. Available at `http://www.springerlink.com/content/y053502404521143/`.

[9] Vineet Gupta, Thomas A Henzinger & Radha Jagadeesan (1997): *Robust Timed Automata*. In: *Hybrid and Real-Time Systems*, *LNCS* 1201. Springer-Verlag, pp. 331–345.

[10] T. A. Henzinger, X. Nicollin, J. Sifakis & S. Yovine (1994): *Symbolic Model Checking for Real-Time Systems*. *Information and Computation* 111(2), pp. 193 – 244. Available at `http://www.sciencedirect.com/science/article/B6WGK-45NJVYS-W/2/24e16e68e4c74a19199c87c5330433bf`.

[11] Thomas A. Henzinger & Jean-François Raskin (2000): *Robust Undecidability of Timed and Hybrid Systems*. In: *HSCC'00*, *LNCS* 1790. Springer-Verlag, pp. 145–159.

[12] Robin Milner (1995): *Communication and concurrency*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK.

[13] Anuj Puri (1998): *Dynamical Properties of Timed Automata*. In: *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 210–227. Available at `http://www.springerlink.com/content/c432527n7g271qk1/`.

[14] Anuj Puri (2000): *Dynamical Properties of Timed Automata*. *Discrete Event Dynamic Systems* 10(1-2), pp. 87–113. Available at `http://www.springerlink.com/content/n2q726540t250p41/`.

[15] Tim Regan (1993): *Multimedia in Temporal LOTOS: A Lip-Synchronization Algorithm*. In: *PSTV*. North-Holland Publishing Co., pp. 127–142.

[16] Jean-Bernard Stefani, Laurent Hazard & François Horn (1992): *Computational model for distributed multimedia applications based on a synchronous programming language*. *Computer Communications* 15(2), pp. 114–128. Available at `http://dx.doi.org/10.1016/0140-3664(92)90131-W`.

[17] Department of Information Technology at Uppsala University & the Department of Computer Science at Aalborg University (2008). *UPPAAL home page*. `http://www.uppaal.com/`.

[18] Howard Wong-Toi (1997): *Analysis of Slope-Parametric Rectangular Automata*. In: *Hybrid Systems V*, *Lecture Notes in Computer Science* 1567. Springer Berlin / Heidelberg, pp. 390–413. Available at `http://www.springerlink.com/content/gbpdqhw4k8vq1d0l/`.

[19] Martin De Wulf, Laurent Doyen, Nicolas Markey & Jean-François Raskin (2004): *Robustness and Implementability of Timed Automata*. In: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Lecture Notes in Computer Science* 3253. Springer Berlin / Heidelberg, pp. 118–133. Available at `http://www.springerlink.com/content/ud8lq155dwcm7mrf/`.

[20] Martin De Wulf, Laurent Doyen, Nicolas Markey & Jean-François Raskin (2008): *Robust safety of timed automata*. *Formal Methods in System Design* 33(1-3), pp. 45–84. Available at `http://www.springerlink.com/content/l7416573x425785m/`.

[21] Mihalis Yannakakis & David Lee (1993): *An efficient algorithm for minimizing real-time transition systems*. In: *Computer Aided Verification, Lecture Notes in Computer Science* 697. Springer Berlin / Heidelberg, pp. 210–224. Available at `http://www.springerlink.com/content/p72t4041h860j677/`.