A Decidable Notion of Timed Non-Interference

Roberto Barbuti and Luca Tesei

Dipartimento di Informatica
Università di Pisa
Corso Italia, 40
56125 Pisa - Italy
email: {barbuti, tesei}@di.unipi.it
fax +39-0502212726

Abstract. We present a notion of non-interference which embodies the notion of time. It is useful to verify the strength of a system against attacks depending on the frequency of certain actions. In particular we give a decidable definition of non-interference which can be checked by using existing verification tools. We show an application example of our notion of non-interference by defining a variant of the classical Fischer's mutual exclusion protocol and by analyzing its strength against attacks.

1 Introduction

The use of the TCP/IP protocol for internetworking has led to a global system of interconnected systems which is referred as the Internet. Since its opening and commercialization, the Internet has become a popular target to attacks. Today the Internet security problems are the center of attention, generating much fear throughout the computer and telecommunications industry.

A class of possible attacks to network components is based on use of the time. Time can be used to gather information, as in [9, 15, 19], where the execution time is used to reveal a secret key. Time can be also used in direct attacks. If the frequency of the intrusion is high enough, the attacked system cannot work properly.

In this paper we show how to analyze the strength of a system with respect to the frequency of attacks. To this purpose we specify the system by *timed automata* [4]. Timed automata are widely recognized as a standard model for describing systems in which the time plays a fundamental role. They have been widely studied for their possible use in the verification of real-time systems [1–3, 5, 13, 14, 20].

In order to do the analysis, we define a notion of timed non-interference that embodies also the notion of time and is suitable to detect interference due to the frequency of certain actions. To show an example of our analysis technique we define a variant of Fischer's mutual exclusion protocol [16]. Together with the processes executing the protocol, we specify, still using timed automata, an intruder and the frequency of its attacks. The timed non-interference analysis of the strength of the protocol against the intruder can be based on a reachability test. We define a condition on the frequency which guarantees the correctness.

The paper is organized as follows. In Section 2 we shortly recall timed automata. In Section 3 we give the notion of *interference*, that is the ability of an attacker of altering the proper behavior of a system (actually, as usual, we define the *non-interference* property, that is the absence of interference). The notion of interference we give is decidable and it is related to the frequency of attacks. Section 4 presents a variant of Fischer's protocol of mutual exclusion. We show that the protocol can be broken only by intrusions with high frequency. Thus any attacker with a frequency of attacks lower than a given bound is non-interfering with it.

2 Timed automata

In this section we recall the definition of timed automata [4]. In the following, $\mathcal{R}$ is the set of real numbers, $\mathcal{R}^{\geq 0}$ is the set of non-negative real numbers and $\mathcal{R}^{+}$ is the set of positive real numbers. A *clock* takes values from $\mathcal{R}^{\geq 0}$. Given a set $\mathcal{X}$ of clocks, a *clock valuation* over $\mathcal{X}$ is a function assigning a non-negative real number to every clock. The set of clock valuations of $\mathcal{X}$ is denoted $\mathcal{V}_{\mathcal{X}}$. Given $\nu \in \mathcal{V}_{\mathcal{X}}$ and $\delta \in \mathcal{R}^{+}$, $\nu + \delta$ denotes the valuation that maps each clock $x \in \mathcal{X}$ into $\nu(x) + \delta$.

Given a set $\mathcal{X}$ of clocks, a *reset* γ is a subset of $\mathcal{X}$. The set of all resets of $\mathcal{X}$ is denoted by $\Gamma_{\mathcal{X}}$. Given a valuation $\nu \in \mathcal{V}_{\mathcal{X}}$ and a reset γ, $\nu\backslash\gamma$ denotes the valuation

$$\nu\backslash\gamma(x) = \begin{cases} 0 & \text{if } x \in \gamma \\ \nu(x) & \text{if } x \notin \gamma \end{cases}$$

Given a set $\mathcal{X}$ of clocks, the set $\Psi_{\mathcal{X}}$ of *clock constraints* over $\mathcal{X}$ are defined by the following grammar:

$\psi ::= \text{true} \mid \text{false} \mid \psi \wedge \psi \mid \neg\psi \mid x\#t$

where $x \in \mathcal{X}$, $t \in I\!N$, and $\#$ is a binary operator in $\{<, >, \leq, \geq, =\}$. Clock constraints are evaluated over clock valuations. The satisfaction by a valuation $\nu \in \mathcal{V}_{\mathcal{X}}$ of the clock constraint $\psi \in \Psi_{\mathcal{X}}$, denoted $\nu \models \psi$, is defined as follows:

$$\begin{array}{l} \nu \models true \text{ and } \nu \not\models false \\ \nu \models \psi_1 \wedge \psi_2 \quad \text{iff} \quad \nu \models \psi_1 \text{ and } \nu \models \psi_2 \\ \nu \models \neg\psi \quad \text{iff} \quad \nu \not\models \psi \\ \nu \models x \# t \quad \text{iff} \quad \nu(x) \# t \end{array}$$

Definition 1 (Timed automaton). *A timed automaton T is a tuple $(Q, \Sigma, \mathcal{E}, I, \mathcal{X})$, where Q is a finite set of states, Σ is a finite alphabet of actions, $\mathcal{E}$ is a finite set of edges, $I \subseteq Q$ is the set of initial states, $\mathcal{X}$ is a finite set of clocks. Each edge $e \in \mathcal{E}$ is a tuple in $Q \times \Psi_{\mathcal{X}} \times \Gamma_{\mathcal{X}} \times \Sigma \times Q$.*

If $e = (q, \psi, \gamma, \sigma, q')$ is an edge, q is the source*, q' is the* target*, ψ is the* constraint*, σ is the* label*, γ is the* reset*.*

The semantics of a timed automaton T is based on an infinite transition system $\mathcal{S}(T) = (S, \rightarrow)$, where S is a set of states and $\rightarrow$ is the transition relation. The states S of $\mathcal{S}(T)$ are pairs (q, ν), where $q \in Q$ is a state of T, and ν is a clock valuation. The initial state of $\mathcal{S}(T)$ is a state $s_0 = (q_0, \nu_0)$, where $q_0 \in I$ is an initial state of T and ν_0 is the valuation which assigns 0 to every clock in $\mathcal{X}$. At any state q, given a valuation ν, T can stay idle or it can perform an action labeling an outgoing edge e. If T stays idle, a transition is possible to a state of $\mathcal{S}(T)$ where the state of T is the same, but the valuation has been modified according to the elapsed time. If T moves along an outgoing edge $e = (q, \psi, \gamma, \sigma, q')$, this corresponds to a transition, labeled by σ, of $\mathcal{S}(T)$ from the state (q, ν) to the state $q', \nu \backslash \gamma$. This transition is possible only if the current clock valuation respects the constraint ψ of e. The rules to derive the transitions of $\mathcal{S}(T)$ are the following:

$$1. \frac{\delta \in \mathcal{R}^+}{(q,\nu) \xrightarrow{\delta} (q, \nu + \delta)} \qquad 2. \frac{(q, \psi, \gamma, \sigma, q') \in \mathcal{E}, \nu \models \psi}{(q,\nu) \xrightarrow{\sigma} (q', \nu \backslash \gamma)}$$

Rule 1. represents the case in which T stays idle in a state and the time passes, while Rule 2. corresponds to the occurrence of an action.

Definition 2 (Action sequence, Run). *Let $T = (Q, \Sigma, \mathcal{E}, I, \mathcal{X})$ be a timed automaton and let $d = s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} \ldots$ be an* infinite *derivation of the transition system $\mathcal{S}(T)$:*

- *The* time sequence *t_j of the time elapsed from state s_0 to state s_j in d is defined as follows:*

$$t_0 = 0$$
$$t_{i+1} = t_i + \begin{cases} 0 \ \textit{if}\ l_i \in \Sigma \\ l_i \ \textit{otherwise} \end{cases}$$

- *The* event sequence *of d is the sequence of the events occurred during d including the elapsed times: $(l_0, t_0)(l_1, t_1)\ldots$*
- *The* action sequence *of d is the projection of the event sequence of d on the pairs $\{(l, t) \mid l \in \Sigma\}$*

Let d be an infinite derivation of $\mathcal{S}(T)$ and let r be its action sequence. We say that r is a **run** *of T if it satisfies the following conditions:*

1. *r is infinite; i.e. whenever a state q of T is entered, a transition of T from q will be eventually taken in the rest of the derivation d*
2. *the* time progress condition *is satisfied: for each $M \in \mathcal{R}^{\geq 0}$ there exist an element (σ_i, t_i) of the sequence r such that $t_i > M$.*

The set of all the runs of a timed automaton T constitutes its trace semantics.

Note that the first condition discards the derivations in which the automaton enters a state and lets the time to elapse forever in that state without performing any other transition. Furthermore, the second condition discards the runs in which the time sequence converges to a real number (also called Zeno runs).

Timed automata can be equipped with a special action, the silent action ϵ. Some transitions of the automaton can be labeled by this action and this means that, when traversed, no action is observed. Note that $\epsilon \notin \Sigma$, so the ϵ-actions do not appear in the action sequence of a derivation and, thus, in the runs of the automaton.

The design of complex systems can be simplified by modeling subsystems with different timed automata and considering, for the whole system, the product of them. The synchronized product operation is a syntactic operation between timed automata.

Definition 3 (Synchronized product of timed automata).
Let $T_1 = (Q_1, \Sigma_1, \mathcal{E}_1, I_1, \mathcal{X}_1)$ *and* $T_2 = (Q_2, \Sigma_2, \mathcal{E}_2, I_2, \mathcal{X}_2)$ *be two timed automata with* $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$. *The synchronized product of* T_1 *and* T_2, *denoted by* $T_1 \parallel T_2$, *is the following timed automaton:*

$$T_1 \| T_2 = \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \mathcal{E}, I_1 \times I_2, \mathcal{X}_1 \cup \mathcal{X}_2 \rangle$$

where $\mathcal{E}$ *is such that:*

1. **Synchronization actions**
$\forall \sigma \in \Sigma_1 \cap \Sigma_2, \forall (q_1, \psi_1, \gamma_1, \sigma, q_1') \in \mathcal{E}_1, \forall (q_2, \psi_2, \gamma_2, \sigma, q_2') \in \mathcal{E}_2$
$\mathcal{E}$ *contains* $((q_1, q_2), \psi_1 \wedge \psi_2, \gamma_1 \cup \gamma_2, \sigma, (q_1', q_2'))$
2. T_1 **actions**
$\forall \sigma \in \Sigma_1 \backslash \Sigma_2, \forall (q, \psi, \gamma, \sigma, q') \in \mathcal{E}_1, \forall s \in Q_2$
$\mathcal{E}$ *contains* $((q, s), \psi, \gamma, \sigma, (q', s))$
3. T_2 **actions**
$\forall \sigma \in \Sigma_2 \backslash \Sigma_1, \forall (q, \psi, \gamma, \sigma, q') \in \mathcal{E}_2, \forall s \in Q_1$
$\mathcal{E}$ *contains* $((s, q), \psi, \gamma, \sigma, (s, q'))$

The automaton $T_1 \parallel T_2$ *is called the product automaton and the automata* T_1 *and* T_2 *are called components.*

The product automaton can perform the synchronization transitions only if all the components can perform it and, in this case, all the components perform it in the same instant. Other transitions can be performed by components independently from each other according to their original specification.

3 A Notion of Timed Non-interference for Timed Automata

Non-interference for concurrent systems has been widely studied at various levels. In particular it has been analyzed using specification formalisms as process algebra (see, for example, [10, 11, 17, 18]). Usually the actions of a system are divided into high level and low level ones, and an intruder is allowed to perform the high level ones. Intuitively, the system respects the non-interference property if its behavior in absence of high level actions is equivalent to its behavior, observed on low level actions, when high level actions occur. In this formulation, the system is unaffected from the attacks if its low level behavior does not change. The notion of non-interference has been reformulated in [12] in a real-time setting using a discrete time process algebra.

We now define a notion of non-interference that embodies the notion of time, in a dense time domain, using timed automata. This definition is such that a system is n-non-interfering if its low level behavior is unaffected by attacks which are separated by more than n time units.

A notion of n-non-interference was already introduced in [8], where the behavior of timed automata is described in terms of the accepted language. In this paper we base the notion of non-interference on trace semantics and reachability.

Definition 4 (Restriction to low level actions). *Let* $T = (Q, \Sigma, \mathcal{E}, I, \mathcal{X})$ *be a timed automaton over an alphabet* $\Sigma = (H, L)$ *where* $H \cap L = \emptyset$ *and* $H \cup L = \Sigma$. H

and L are the sets of high and low level actions respectively. We denote by $T|_L$ an automaton obtained from T by substituting syntactically every edge $(q, \psi, \sigma, \gamma, q') \in \mathcal{E}$ such that $\sigma \in H$ with the edge $(q, \text{false}, \sigma, \gamma, q')$.

Thus, $T|_L$ behaves as T but all the runs in which at least one high level action is executed are discarded because high level actions are never enabled. We refer to the behaviors of $T|_L$ as the low behaviors or basic behaviors.

Definition 5 (Hiding of high level actions). *Let $T = (Q, \Sigma, \mathcal{E}, I, \mathcal{X})$ be a timed automaton over an alphabet $\Sigma = (H, L)$. We denote by $T \backslash H$ an automaton obtained from T by substituting syntactically every edge $(q, \psi, \sigma, \gamma, q') \in \mathcal{E}$ such that $\sigma \in H$ with the edge $(q, \psi, \epsilon, \gamma, q')$.*

Thus, $T \backslash L$ behaves as T but all the high level actions are substituted by the silent action ϵ, thus their occurrence is not visible in the trace semantics.

Consider, now, the automaton Interf_H^n in Figure 1, where an arc having as label a set of action represents a set of edges, one for each action in the set (L or H), with the same clock constraint and clock reset. This automaton allows the execution of high-level actions only when they are separated by at least n time units.

Given a timed automaton T, the synchronized product of T with Interf_H^n, $T \parallel \text{Interf}_H^n$, allows to observe the set of all behaviors of T such that high-level actions occur at times separated by an interval whose length is greater than, or equal to n. Of course, we are assuming that T does not reset the clock x_{interf}.

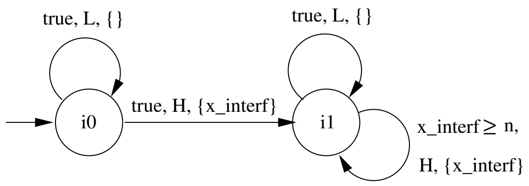

Fig. 1. The structure of Interf_H^n

Definition 6 (n-non-interference). *Let $\equiv$ be an arbitrary equivalence relation between timed automata. Let $T = (Q, \Sigma, \mathcal{E}, I, \mathcal{X})$ be a timed automaton with an alphabet $\Sigma = (H, L)$. T is* n-non-interfering *iff*

$$T|_L \equiv (T \parallel \text{Interf}_H^n) \backslash H \tag{1}$$

Thus, T is n-non-interfering iff its low behaviors do not change, with respect to the given equivalence, whenever either high level actions are not performed or they are performed with a delay of at least n time units between them and then they are hidden.

Proposition 1. *Let T be a timed automaton and $n \in I\!N$. If T is n-non-interfering then T is also $(n + 1)$-non-interfering.*

Proof. It is simple to see that the condition of n-non-interference depends on the frequency of the occurrence of high-level actions. More precisely, it can be missed only if such a frequency is too high. Thus, if the condition holds for a certain n it will hold for every $n' \geq n$ because the n'-non-interference requires the same equivalence allowing at most a lower frequency of high-level actions than the n-non-interference, which is verified by hypothesis.

Definition 7 (n-trace-non-interference). *Let $T = (Q, \Sigma, \mathcal{E}, I, \mathcal{X})$ be a timed automaton with an alphabet $\Sigma = (H, L)$ and let n be a natural number. T is* n-trace-non-interfering *iff the set of runs of $T|_L$ is equal to the set of runs of $(T \parallel \mathrm{Interf}^n_H)\backslash H$.*

This notion is natural for timed automata, but suffers of a negative decidability result for trace equivalence of timed automata (see [4]).

Now, we introduce an equivalence which is decidable and can be used to define a different notion of n-non-interference. This equivalence focuses the reachable states of the system. The decidability of the new notion relies on the decidability of the reachability test for timed automata [4]. This test is one of the most used ways of verification of properties of a timed automaton and all the tools managing timed automata implement it efficiently.

Definition 8 (n-state-non-interference). *Let $T = (Q, \Sigma, \mathcal{E}, I, \mathcal{X})$ be a timed automaton with an alphabet $\Sigma = (H, L)$ and let n be a natural number. T is* n-state-non-interfering *iff the set of reachable states of $T|_L$ is equal to the set of reachable states of $(T \parallel \mathrm{Interf}^n_H)\backslash H$, projected on the states of T.*

Let R be the set of reachable states of $T|_L$. Since this automaton can not perform high level actions, R contains that states of T that can be accessed without the aid of any high level activity. The notion of n-state-non-interference then requires that R does not change when a controlled high-level activity is allowed. This control imposes that any two subsequent high level actions are separated by at least n time units.

The following proposition shows that the two notions of non-interference take into account of different aspects of systems.

Proposition 2. *Let T be a timed automaton. T is n-trace-non-interfering $\not\Rightarrow$ T is n-state-non-interfering. T is n-state-non-interfering $\not\Rightarrow$ T is n-trace-non-interfering.*

Proof. Let we show two counter examples. Figure 2 shows an automaton which is n-trace-non-interfering for all $n \in \mathbb{N}$. This is because the high level transition labeled with h from state 0 to state 2 is followed, possibly without delay, by a transition labeled with the low level action l from state 2 to state 1. Thus, when the automaton $T|_L$ is considered, the action h can not be performed and the runs are of the form $(l, t_0)(l, t_1)\cdots$ in which it is only required that $t_i \leq t_{i+1}$. When the automaton $(T \parallel Interf^n_H)\backslash H$ is considered, the action h can be performed, but is not observable. Thus the runs are of the same form of those of $T|_L$ because there are not clock constraints. On the other hand, the system T of Figure 2 is not n-state-non-interfering for any $n \in \mathbb{N}$. This is because the state 2 is reachable in $(T \parallel Interf^n_H)\backslash H$ for all $n \in \mathbb{N}$ and it is not reachable in $T|_L$.

Figure 3 shows the counterexample for the converse. In this case it is clear that the automaton is n-state-non-interference. On the other hand, all the runs of $T|_L$

begin with $(l_0, t_0)\cdots$, but the automaton $(T \parallel Interf_H^n)\backslash H$ has a run beginning with $(l_1, t_0)\cdots$.

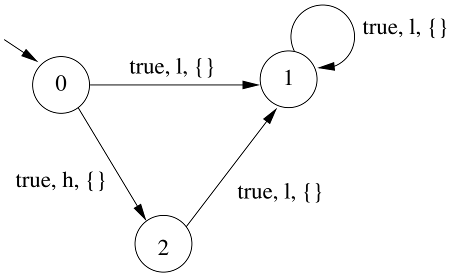

Fig. 2. A system that is n-trace-non-interfering, but not n-state-non-interfering

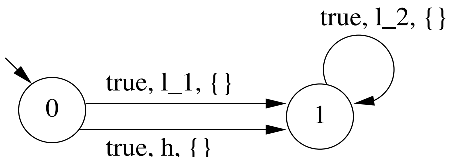

Fig. 3. A system that is n-state-non-interfering, but not n-trace-non-interfering

4 An example of timed non-interference

In this section we show the utility of non-interference analysis to state the robustness, with respect to external attacks, of a classical timing-based mutual exclusion protocol.

4.1 The Fischer Protocol for Mutual Exclusion

The protocol was suggested by Michael Fischer and reported in [16]. The significance of the protocol is due to its speed, that makes it suitable for multiprocessor computers or time-critical embedded systems.

Suppose that two processes, P_1 and P_2, are running in parallel, competing for a critical section, and assume that atomic reads and writes are permitted to a shared variable x. Assume also that every access to the shared memory containing x takes acc units of time. Each process executes the following algorithm, where the code both of the critical section and the one outside the protocol is assumed not to modify x.

```
repeat
       await x=0;
```

```
        x:=i;
        delay b
until   x=i;
Critical Section;
x:=0
```

Each process P_i is allowed to be in its critical section iff $x = i$. The statement `await x=0` waits until the value of x becomes 0. The statement `delay b` delays a process for b time units, as measured by the process clock. Here we assume that the local clocks of the two processes proceed at the same rate. Each statement takes an amount of time to be executed, in particular we assume that the assignment statement takes at most a time units. Recall that an access to the shared memory containing x is atomic, and the processes P_1 and P_2 can compete for accessing it, thus the time of each assignment, a, may depend on the resolution of conflicts.

The Fisher's protocol ensures mutual exclusion iff $a < b$.

Each process can be represented by a timed automaton like the one in Figure 4, a slightly different representation of the one given in [6, 7].

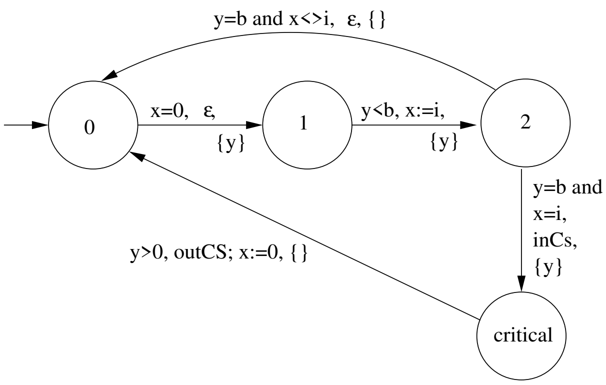

Fig. 4. Automaton P_i

State 0 corresponds to the local computation of the process. The process can start the protocol for accessing the critical section only if the value of x is equal to 0. At this point (state 1) it can assign x (in a time shorter than b[1]) and it waits b time units for testing it, and, depending on its value, for entering the critical region (state *critical*). Note that, for brevity, we put more than one action on edges. On exiting the critical section, the process sets the value of x to 0.

It is important to note that P_1 and P_2 must not synchronize on actions, but only on the value of variable x. Thus the same action executed by a process is considered different if executed by another process; for instance $(\texttt{x:=0})_{P_1}$, executed by P_1, is different from $(\texttt{x:=0})_{P_2}$, executed by P_2.

[1] Actually the assignment has to occur within a time units and a has to be less than b. Here, for simplicity, we use b in both cases.

The automaton of Figure 4 does not consider the time, acc, for accessing the shared memory. Thus, when P_1 and P_2 are combined in parallel they can both start an access to x, and the time interval between these accesses could be shorter then acc. This is in contrast with the assumption that accesses to x are atomic.

To force the accesses to be atomic, we add, to the system composed by the two processes, the serializer of Figure 5 which synchronizes with P_1 and P_2 on every action which performs an access to x.

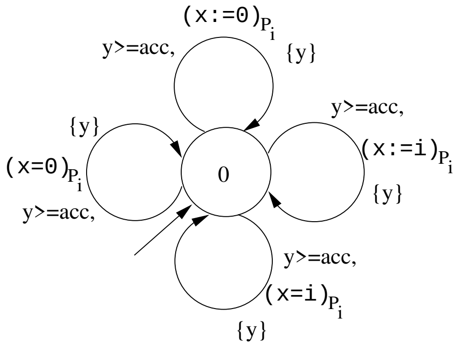

Fig. 5. The serializer

Let us analyze the behavior of the two processes, P_1 and P_2. The automaton ($P_1 \parallel P_2 \parallel$ serializer), obtained by the synchronized product of the components, can reach any possible state $\langle s_1, s_2, 0\rangle$ (where s_1 and s_2 are states of P_1 and P_2, and 0 is the unique state of the serializer) but the state $\langle critical, critical, 0\rangle$, because of the correctness of the mutual exclusion protocol.

Let us assume now that there is a upper bound, ucs, on the time needed to execute the critical section after the variable x is checked. This assumption is reasonable when the operations in the critical section are fast and simple, for example the ones for updating a counter. If $ucs \leq a + b$, the previous protocol can be modified in order to decrease the delay in accessing the critical section when a conflict is present. The idea is that, because $ucs \leq a + b$, after executing successfully the protocol and before entering the critical section, a process can signal to the other that the protocol can be executed again. This is safe because the time for executing the critical section is less than or equal to the time taken by the protocol itself. Actually ucs includes the time for assigning the value 0 to x.

The new protocol is now the following.

```
repeat
       await x=0;
       x:=i;
       delay b
until  x=i;
x:=0
```

```
Critical Section;
```

Let us represent process P_i' executing the new protocol by the timed automaton of Figure 6.

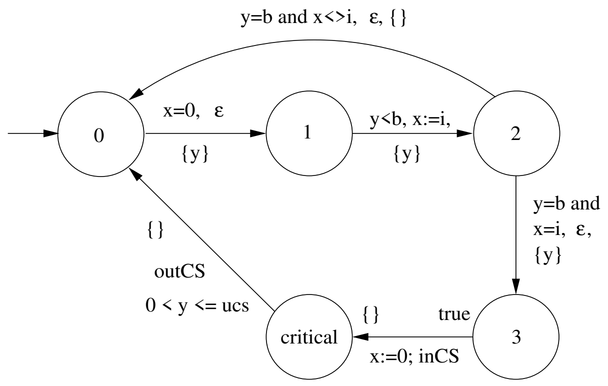

Fig. 6. Automaton P_i'

4.2 Timed Non-Interference

Suppose the existence of an intruder which is able to read and write the shared variable x. Such an intruder can be either a malicious host connected to the network, or simply another component of the system accessing the shared memory for other purposes. The possible behaviors of the intruder are described by the timed automaton in Figure 7[2]. Note that to implement atomic accesses to x the serializer must be extended for taking account also of the actions of the intruder.

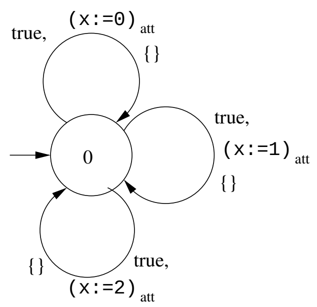

Fig. 7. The intruder

Consider now the automaton $T = (P_1' \parallel P_2' \parallel \text{serializer} \parallel \text{intruder})$

[2] Actually only writing actions are critical, thus we restrict the system to them.

In order to apply Definition 8 we consider the actions of the intruder as the high level actions, thus $H = \{(\mathtt{x} := 0)_{\mathrm{att}}, (\mathtt{x} := 1)_{\mathrm{att}}, (\mathtt{x} := 2)_{\mathrm{att}}\}$.

We want to show that T is n-state-non-interfering, for some n. This corresponds to show that the set of reachable states of the system $(P_1' \parallel P_2' \parallel \text{serializer} \parallel \text{intruder} \parallel \text{Interf}_H^n)\backslash H$, projected on the states of $(P_1' \parallel P_2' \parallel \text{serializer} \parallel \text{intruder})$, is equal to the set of reachable states of the system in which all the actions of the intruder are forbidden. We know, by the previous analysis of the system without the intruder, that the state $\langle \mathit{critical}, \mathit{critical}, 0\rangle$ is not reachable. Thus, in this example, the condition of Definition 8 reduces to verify that the state $\langle \mathit{critical}, \mathit{critical}, 0, 0\rangle$ of $(P_1' \parallel P_2' \parallel \text{serializer} \parallel \text{intruder} \parallel \text{Interf}_H^n)\backslash H$ (projected on the states of $(P_1' \parallel P_2' \parallel \text{serializer} \parallel \text{intruder})$) is still unreachable.

Remark that the original Fischer protocol was very weak from this point. If the intruder set the variable `x` to 0 when a process is in its critical region, the other process is enabled to enter the critical region too.

First let us show that if the value of n is lower enough the system can be n-state-interfering, as showed by the following table, where $acc = 1$, $b = 6$, $ucs = 6$ and $n = 2$. Each depicted interval corresponds to acc time units.

P_1'	`x=0`		`x:=1`					`x=1`			`inCS` `x:=0`		*crash*
P_2'		`x=0`			`x:=2`					`x=2`		`inCS` `x:=0`	*crash*
Intruder							`x:=1`		`x:=2`				

We can state the following general result for the system.

Proposition 3. *For all $n > b$ the system $(P_1 \parallel P_2 \parallel \mathit{serializer} \parallel \mathit{intruder})$ is n-state-non-interfering.*

Thus, by n-state-non-interference analysis, we are able to obtain an upper-bound on the frequency of attacks to the protocol under which it is not affected by the attack.

5 Concluding remarks

In this paper we have defined, using a timed automata based approach, a notion of timed non-interference suitable to capture systems free of interference due to high frequency of certain actions. Then, we have showed an example of timed non-interference.

Note that n-state-non-interference does not guarantee that the intruder could not interfere in another manner, for instance causing a deadlock or forbidding a process to reach the critical section all of the times. These other attacks do not strongly depend on the frequency of the action of the intruder and should be addressed using different methods.

References

1. Aceto, L., Bouyer, P., Burgueño, A. and Guldstrand Larsen, K. (1998) The Power of Reachability Testing for Timed Automata. In Proceedings of Foundations Software Technology and Theoretical Computer Science, Springer LNCS 1530, 245–256.
2. Aceto, L., Burgueño, A. and Guldstrand Larsen, K. (1998) Model Checking via Reachability Testing for Timed Automata. In Proceedings of TACAS'98, Springer LNCS 1384, 263–280.
3. Alur, R., Courcoubetis, C. and Dill, D.L. (1993) Model-Checking in Dense Real-time. *Information and Computation*, **104**, 2–34.
4. Alur, R. and Dill, D.L. (1994) A Theory of Timed Automata. *Theoretical Computer Science*, **126**, 183–235.
5. Alur, R. and Henzinger, T.A. (1994) A Really Temporal Logic. *Journal of ACM*, **41**, 181–204.
6. Abdulla, P.A. and Jonsson, B. (1998) Networks of timed processes. In Proceedings of TACAS'98, Springer LNCS 1384, 298-312.
7. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J. and Yovine, S. (1995) The Algorithmic Analysis of Hybrid Systems. *Theoretical Computer Science*, **138**, 3–34.
8. Barbuti, R., De Francesco, N., Santone, A. and Tesei, L. A Notion of non-Interference for Timed Automata (2001) In Proceedings of Concurrency, Specification and Programming Workshop (CS&P'2001), (L. Czaja ed.), Warsaw, Poland, October 2001.
9. Dhem, J-F., Koeune, F., Leroux, P-A., Mestr, P., Quisquater, J-J. and Willems, J-L. (2000) A Practical Implementation of the Timing Attack. In Proceedings of CARDIS 1998, Springer LNCS 1820, 167–182.
10. Focardi, R. and Gorrieri, R. (1996) Automatic Compositional Verification of Some Security Properties. In Proceedings of TACAS'96, Springer LNCS 1055, 167–186.
11. Focardi, R. and Gorrieri, R. (1997) The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties. *IEEE Transactions on Software Engineering*, **23**(9), 550–571.
12. Focardi, R., Gorrieri, R. and Martinelli, F. (2000) Information Flow in a Discrete Time Process Algebra. In Proceedings of 13th IEEE Computer Security Foundations Workshop (CSFW'00), (P. Syverson ed.), IEEE press, Cambridge, England, July 2000.
13. Henzinger, T.A. and Kopke, P.W. (1994) Verification Methods for the Divergent Runs of Clock Systems. In Proceedings of Formal Techniques in Real-Time and Fault-Tolerant Systems, Springer LNCS 863, 351–372.
14. Henzinger, T.A., Nicollin, X., Sifakis, J. and Yovine, S. (1994) Symbolic Model Checking for Real-Time Systems. *Information and Computation*, **111**, 193–244.
15. Kocher, P.C. (1996) Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In proceedings of CRYPTO 1996, Springer LNCS 1109, 104–113.
16. Lamport, L. (1987) A Fast Mutual Exclusion Algorithm. *ACM TOPLAS*, **5**, 1–11.
17. P. Y. A. Ryan, S. A. Schneider. (2001) Process Algebra and Non-Interference. *Journal of Computer Security*, **9**(1/2), 75-103.
18. A.W. Roscoe, J.C.P. Woodcock, L. Wulf. (1996) Non-Interference Through Determinism. *Journal of Computer Security*, **4**(1).
19. Schindler, W. (2000) A Timing Attack against RSA with the Chinese Remainder Theorem. In Proceedings of CHES 2000, Springer LNCS 1965, 109–124.
20. Yovine, S. (1996) Model Checking Timed Automata. Lectures on Embedded Systems, Springer LNCS 1494, 114–152.