Reachability of Communicating Timed Processes*

Lorenzo Clemente[1], Frédéric Herbreteau[1], Amelie Stainer[2], and Grégoire Sutre[1]

[1] Univ. Bordeaux, CNRS, LaBRI, UMR 5800, Talence, France
[2] University of Rennes 1, Rennes, France

Abstract. We study the reachability problem for communicating timed processes, both in discrete and dense time. Our model comprises automata with local timing constraints communicating over unbounded FIFO channels. Each automaton can only access its set of local clocks; all clocks evolve at the same rate. Our main contribution is a complete characterization of decidable and undecidable communication topologies, for both discrete and dense time. We also obtain complexity results, by showing that communicating timed processes are at least as hard as Petri nets; in the discrete time, we also show equivalence with Petri nets. Our results follow from mutual topology-preserving reductions between timed automata and (untimed) counter automata. To account for urgency of receptions, we also investigate the case where processes can test emptiness of channels.

1 Introduction

Communicating automata are a fundamental model for studying concurrent processes exchanging messages over unbounded channels [23,12]. However, the model is Turing-powerful, and even basic verification questions, like reachability, are undecidable. To obtain decidability, various restrictions have been considered, including making channels unreliable [3,14] or restricting to half-duplex communication [13] (later generalized to mutex [18]). Decidability can also be obtained when restricting to executions satisfying additional restrictions, such as bounded context-switching [21], or bounded channels. Finally, and this is the restriction that we consider here, decidability is obtained by constraining the communication topology. For communicating finite-state machines (CFSMs), it is well-known that reachability is decidable if, and only if, the topology is a polyforest [23,21]; in this case, considering channels of size one suffices for deciding reachability.

On a parallel line of research, *timed automata* [9] have been extensively studied as a finite-state model of timed behaviours. Recently, there have been several works bringing time into infinite-state models, including *timed Petri nets* [10,4], *timed pushdown automata* [2], and *timed lossy channel systems* [1]. In this paper, we study *communicating timed processes* [20], where a finite number of timed

* This work was partially supported by the ANR project Vacsim (ANR-11-INSE-004).

F. Pfenning (Ed.): FOSSACS 2013, LNCS 7794, pp. 81–96, 2013.

automata synchronize over the elapsing of time and communicate by exchanging messages over unbounded channels. Note that, when processes can synchronize, runs cannot be re-ordered to have uniformly bounded channels (contrary to polyforest CFSMs). For example, consider two communicating processes p and q, where p can send to q unboundedly many messages in the first time unit, and q can receive messages only after the first time unit has elapsed. Clearly, all transmissions of p have to occur before any reception by q, which excludes the possibility of re-ordering the run into another one with bounded channels.

We significantly extend the results of [20], by giving a complete characterization of the decidability border of reachability properties w.r.t. the communication topology. Quite surprisingly, we show that despite synchronization increases the expressive power of CFSMs, the undecidability results of [20] are not due to just synchronous time, but to an additional synchronization facility called *urgency* (cf. below). Our study comprises both dense and discrete time.

Dense Time: Communicating Timed Automata. Our main result is a complete characterization of the decidability frontier for communicating timed automata: We show that reachability is decidable if, and only if, the communication topology is a polyforest. Thus, adding time does not change the decidability frontier w.r.t. CFSMs. However, the complexity worsens: From our results it follows that communicating timed automata are at least as hard as Petri nets.[1]

Our decidability results generalize those of [20] over the standard semantics for communicating automata. In the same work, undecidability results are also presented. However, they rely on an alternative *urgent semantics*, where, if a message can be received, then all internal actions are disabled: This provides an extra means of synchronization, which makes already the very simple topology $p \to q \to r$ undecidable [20]. We show that, without urgency, this topology remains decidable.

Here, we do not consider urgency directly, but we rather model it by introducing an additional *emptiness test* operation on channels on the side of the receiver. This allows us to discuss topologies where emptiness tests (i.e., urgency) are restricted to certain components. We show that, with emptiness tests, undecidable topologies include not only the topology $p \to q \to r$ (as shown in [20]), but also $p \to q \leftarrow r$ and $p \leftarrow q \to r$. Thus, we complete the undecidability picture for dense time.

All our results for dense time follow from a mutual, topology-preserving reduction to a discrete-time model (discussed below). Over polyforest topologies, we reduce from dense to discrete time when no channel can be tested for emptiness. Over arbitrary topologies, we reduce from discrete to dense time, even in the presence of emptiness tests. While the latter is immediate, the former is obtained via a *Rescheduling Lemma* for dense-time timed automata which is interesting on its own, allowing us to schedule processes in fixed time-slots where senders are always executed before receivers.

[1] And probably exponentially worse, due to a blow-up when translating from dense to discrete time.

Discrete Time: Communicating Tick Automata. We provide a detailed analysis of communication in the discrete-time model, where actions can only happen at integer time points. As a model of discrete time, we consider *communicating tick automata*, where the flow of time is represented by an explicit tick action: A process evolves from one time unit to the next by performing a tick action, forcing all the other processes to perform a tick as well; all the other actions are asynchronous. This model of discrete-time is called *tick automata* in [17], which is related to the *fictitious-time* model of [9].

We provide a complete characterization of decidable and undecidable topologies for communicating tick automata: We show that reachability is decidable if, and only if, the topology is a polyforest (like for CFSMs), and, additionally, each weakly-connected component can test at most one channel for emptiness. Our results follow from topology-preserving mutual reductions between communicating tick automata and counter automata. As a consequence of the structure of our reductions, we show that channels and counters are mutually expressible, and similarly for emptiness tests and zero tests. This also allows us to obtain complexity results for communicating tick automata: We show that reachability in a system of communicating tick automata over a weakly-connected topology without emptiness tests has the same complexity as reachability in Petri nets.[2]

Related Work. Apart from [20], communication in a dense-time scenario has also been studied in [15,8,6]. In particular, [15] proposes timed message sequence charts as the semantics of communicating timed automata, and studies the scenario matching problem where timing constraints can be specified on local processes, later extended to also include send/receive pairs [8]. Communicating event-clock automata, a strict subclass of timed automata, are studied in [6] where, instead of considering the decidability frontier w.r.t. the communication topology, it is shown, among other results, that reachability is decidable for arbitrary topologies over existentially-bounded channels. A crucial difference w.r.t. our work is that we do not put any restriction on the channels, and we consider full timed automata. In a distributed setting, the model of global time we have chosen is not the only possible. In particular, [7] studies decidability of networks of (non-communicating) timed asynchronous automata in an alternative setting where each automaton has a local drift w.r.t. global time. In the discrete-time setting, we mention the work [19], which generalizes communicating tick automata to a loosely synchronous setting, where local times, though different, can differ at most by a given bound. While [19] shows decidability for a restricted two-processes topology, we characterize decidability for arbitrary topologies. We finally mention [16,5] that address reachability for parametrized ad hoc networks in both discrete and dense time. They consider an infinite number of processes and broadcast handshake communications over arbitrary topologies while our models have a finite number of processes that exchange messages over unbounded (unicast) channels.

[2] The latter problem is known to be ExpSpace-hard [22], and finding an upper bound is a long-standing open problem.

Outline. In Sec. 2 we introduce general notation; in particular, we define communicating timed processes, which allow us to uniformly model communication in both the discrete and dense time. In Sec. 3 we study the decidability and complexity for communicating tick automata (discrete time), while in Sec. 4 we deal with communicating timed automata (dense time). Finally, Sec. 5 ends the paper with conclusions and future work.

2 Communicating Timed Processes

A *labeled transition system* (LTS for short) is a tuple $\mathcal{A} = \langle S, S_I, S_F, A, \to \rangle$ where S is a set of *states* with *initial states* $S_I \subseteq S$ and *final states* $S_F \subseteq S$, A is a set of *actions*, and $\to \subseteq S \times A \times S$ is a *labeled transition relation*. For simplicity, we write $s \xrightarrow{a} s'$ in place of $(s, a, s') \in \to$. A *path* in $\mathcal{A}$ is an alternating sequence $\pi = s_0, a_1, s_1, \ldots, a_n, s_n$ of states $s_i \in S$ and actions $a_i \in A$ such that $s_{i-1} \xrightarrow{a_i} s_i$ for all $i \in \{1, \ldots, n\}$. We abuse notation and shortly denote π by $s_0 \xrightarrow{a_1 \cdots a_n} s_n$. The word $a_1 \cdots a_n \in A^*$ is called the *trace* of π. A *run* is a path starting in an initial state ($s_0 \in S_I$) and ending in a final state ($s_n \in S_F$).

We consider systems that are composed of several processes interacting with each other in two ways. Firstly, they implicitly synchronize over the passing of time. Secondly, they explicitly communicate through the asynchronous exchange of messages. For the first point, we represent delays by actions in a given *delay domain* $\mathbb{D}$. Typically, the delay domain is a set of non-negative numbers when time is modeled quantitatively, or a finite set of abstract delays when time is modeled qualitatively. Formally, a *timed process* over $\mathbb{D}$ is a labeled transition system $\mathcal{A} = \langle S, S_I, S_F, A, \to \rangle$ such that $A \supseteq \mathbb{D}$. Actions in A are either synchronous *delay actions* in $\mathbb{D}$, or *asynchronous actions* in $A \setminus \mathbb{D}$.

For the second point, we introduce FIFO channels between processes. Formally, a communication *topology* is a triple $\mathcal{T} = \langle P, C, E \rangle$, where $\langle P, C \rangle$ is a directed graph comprising a finite set P of *processes* and a set of communication *channels* $C \subseteq P \times P$. Additionally, the set $E \subseteq C$ specifies those channels that can be tested for emptiness. Thus, a channel $c \in C$ is a pair (p, q), with the intended meaning that process p can send messages to process q. For a process p, let $C[p] = C \cap (\{p\} \times P)$ be its set of outgoing channels, and let $C^{-1}[p] = C \cap (P \times \{p\})$ be its set of incoming channels. Processes may send messages to outgoing channels, receive messages from incoming channels, as well as test emptiness of incoming channels (for testable channels). Formally, given a finite set M of messages, the set of possible *communication actions* for process p is $A^p_{\text{com}} = \{c!m \mid c \in C[p], m \in M\} \cup \{c?m \mid c \in C^{-1}[p], m \in M\} \cup \{c \texttt{==} \varepsilon \mid c \in E \cap C^{-1}[p]\}$. The set of all communication actions is $A_{\text{com}} = \bigcup_{p \in P} A^p_{\text{com}}$. While send actions ($c!m$) and receive actions ($c?m$) are customary, we introduce the extra test action ($c \texttt{==} \varepsilon$) to model the *urgent semantics* of [20]. Actions not in ($\mathbb{D} \cup A_{\text{com}}$) are called *internal actions*.

Definition 1. *A system of communicating timed processes is a tuple* $\mathcal{S} = \langle \mathcal{T}, M, \mathbb{D}, (\mathcal{A}^p)_{p \in P} \rangle$ *where* $\mathcal{T} = \langle P, C, E \rangle$ *is a topology,* M *is a finite set of*

messages, $\mathbb{D}$ *is a delay domain, and, for each* $p \in P$, $\mathcal{A}^p = \langle S^p, S^p_I, S^p_F, A^p, \to^p \rangle$ *is a timed process over* $\mathbb{D}$ *such that* $A^p \cap A_{\text{com}} = A^p_{\text{com}}$.

States $s^p \in S^p$ are called *local states* of p, while a *global state* $\boldsymbol{s} = (s^p)_{p \in P}$ is a tuple of local states in $\prod_{p \in P} S^p$. We give the semantics of a system of communicating timed processes in terms of a global labeled transition system. The contents of each channel is represented as a finite word over the alphabet M. Processes move asynchronously, except for delay actions that occur simultaneously. Formally, the *semantics of a system of communicating timed processes* $\mathcal{S} = \langle \mathcal{T}, M, \mathbb{D}, (\mathcal{A}^p)_{p \in P} \rangle$ is the labeled transition system $[\![\mathcal{S}]\!] = \langle S, S_I, S_F, A, \to \rangle$ where $S = (\prod_{p \in P} S^p) \times (M^*)^C$, $S_I = (\prod_{p \in P} S^p_I) \times \{\lambda c \,.\, \varepsilon\}$, $S_F = (\prod_{p \in P} S^p_F) \times \{\lambda c \,.\, \varepsilon\}$, $A = \bigcup_{p \in P} A^p$, and there is a transition $(\boldsymbol{s}_1, w_1) \xrightarrow{a} (\boldsymbol{s}_2, w_2)$ under the following restrictions:

- if $a \in \mathbb{D}$, then $s^p_1 \xrightarrow{a} s^p_2$ for all $p \in P$,
- if $a \notin \mathbb{D}$, then $s^p_1 \xrightarrow{a} s^p_2$ for some $p \in P$, and $s^q_1 = s^q_2$ for all $q \in P \setminus \{p\}$
  - if $a = c!m$, then $w_2(c) = w_1(c) \cdot m$ and $w_2(d) = w_1(d)$ for all $d \in C \setminus \{c\}$,
  - if $a = c?m$, then $m \cdot w_2(c) = w_1(c)$ and $w_2(d) = w_1(d)$ for all $d \in C \setminus \{c\}$,
  - if $a = (c == \varepsilon)$, then $w_1(c) = \varepsilon$ and $w_1 = w_2$, and
  - if $a \notin A_{\text{com}}$, then $w_1 = w_2$.

To avoid confusion, states of $[\![\mathcal{S}]\!]$ will be called *configurations* in the remainder of the paper. Given a path π in $[\![\mathcal{S}]\!]$, its *projection* to process p is the path $\pi|_p$ in $\mathcal{A}^p$ obtained by projecting each transition of π to process p in the natural way.

The *reachability problem* asks, given a system of communicating timed processes $\mathcal{S}$, whether there exists a run in its semantics $[\![\mathcal{S}]\!]$. Note that we require all channels to be empty at the end of a run, which simplifies our constructions later by guaranteeing that every sent message is eventually received. (This is w.l.o.g. since reachability and control-state reachability are easily inter-reducible.) Two systems of communicating timed processes $\mathcal{S}$ and $\mathcal{S}'$ are said to be *equivalent* if $[\![\mathcal{S}]\!]$ has a run if and only if $[\![\mathcal{S}']\!]$ has a run.

Definition 2. *A* system of communicating tick automata *is a system of communicating timed processes* $\mathcal{S} = \langle \mathcal{T}, M, \mathbb{D}, (\mathcal{A}^p)_{p \in P} \rangle$ *such that* $\mathbb{D} = \{\tau\}$ *and each* $\mathcal{A}^p$ *is a* tick automaton*, i.e., a timed process over* $\mathbb{D}$ *with finitely many states and actions.*

Thus, tick automata communicate with actions in A_{com} and, additionally, synchronize over the tick action τ. This global synchronization makes communicating tick automata more expressive than CFSMs, in the sense that ticks can forbid re-orderings of communication actions that are legitimate without ticks. Notice that there is only one tick symbol in $\mathbb{D}$. With two different ticks, reachability is already undecidable for the one channel topology $p \to q$ without emptiness test.

3 Decidability of Communicating Tick Automata

In this section, we study decidability and complexity of communicating tick automata. Our main technical tool consists of mutual reductions to/from counter

automata, showing that, in the presence of tick actions, 1) each channel is equivalent to a counter, and 2) each emptiness test on a channel is equivalent to a zero test on the corresponding counter. This allows us to derive a complete characterization of decidable topologies, and to also obtain complexity results. We begin by defining communicating counter automata.

Communicating Counter Automata. A *counter automaton* is a classical Minsky machine $\mathcal{C} = \langle L, L_I, L_F, A, X, \Delta \rangle$ with finitely many locations L, initial locations $L_I \subseteq L$, final locations $L_F \subseteq L$, alphabet of actions A, finitely many non-negative counters in X, and transition rules $\Delta \subseteq L \times A \times L$. Operations on a counter $x \in X$ are $x\texttt{++}$ (increment), $x\texttt{--}$ (decrement) and $x\texttt{==0}$ (zero test). Let $\texttt{Op}(X)$ be the set of operations over counters in X. We require that $A \supseteq \texttt{Op}(X)$. As usual, the semantics is given as a labelled transition system $[\![\mathcal{C}]\!] = \langle S, S_I, S_F, A, \rightarrow \rangle$ where $S = L \times \mathbb{N}^X$, $S_I = L_I \times \{\lambda x.0\}$, $S_F = L_F \times \{\lambda x.0\}$, and the transition relation $\rightarrow$ is defined as usual. Acceptance is with zero counters.

A *system of communicating counter automata* is a system of communicating timed processes $\mathcal{S} = \langle \mathcal{T}, M, \mathbb{D}, ([\![\mathcal{C}^p]\!])_{p \in P} \rangle$ such that $\mathbb{D} = \emptyset$ and each $\mathcal{C}^p$ is a counter automaton. By Definition 1, this entails that each counter automaton performs communicating actions in A^p_{com}. Notice that, since the delay domain is empty, no synchronization over delay action is possible.

From Tick Automata to Counter Automata. Let $\mathcal{S}$ be a system of communicating tick automata over an arbitrary (i.e., possibly cyclic) weakly-connected[3] topology. We build an equivalent system of communicating counter automata $\mathcal{S}'$ over the same topology. Intuitively, we implement synchronization on the delay action τ in $\mathcal{S}$ by communication in $\mathcal{S}'$ (by definition, no synchronization on delay actions is allowed in $\mathcal{S}'$). We introduce a new type of message, also called τ, which is sent in broadcast by all processes in $\mathcal{S}'$ each time there is a synchronizing tick action in $\mathcal{S}$. Since communication is by its nature asynchronous, we allow the sender and the receiver to be momentarily desynchronized during the computation. However, we restrict the desynchronization to be asymmetric: The receiver is allowed to be "ahead" of the sender (w.r.t. the number of ticks performed), but never the other way around. This ensures causality between transmissions and receptions, by forbidding that a message is received before it is sent.

To keep track of the exact amount of desynchronization between sender and receiver (as the difference in number of ticks), we introduce counters in $\mathcal{S}'$: We endow each process p with a non-negative counter $\mathtt{x}^p_c$ for each channel $c \in C^{-1}[p]$ from which p is allowed to receive. The value of counter $\mathtt{x}^p_c$ measures the difference in number of ticks τ between p and the corresponding sender along c. Whenever a process p performs a synchronizing tick action τ in $\mathcal{S}$, in $\mathcal{S}'$ it sends a message τ in broadcast onto all outgoing channels; at the same time, all its counters $\mathtt{x}^p_c$ are incremented, recording that p, as a receiver process, is one more step ahead of its corresponding senders. When one such τ-message is received by a process

[3] A topology $\mathcal{T}$ is *weakly-connected* if, for every two processes, there is an undirected path between them.

q in $\mathcal{S}'$ along channel c, the corresponding counter $\mathtt{x}_c^q$ is decremented; similarly, this records that the sender process along c is getting one step closer to the receiver process q. The topology needs to be weakly-connected for the correct propagation of τ's.

While proper ordering of receptions and transmissions is ensured by non-negativeness of counters, testing emptiness of the channel is more difficult: In fact, a receiver, which in general is ahead of the sender, might see the channel as empty at one point (thus the test is positive), but then the sender might later (i.e., after performing some tick) send some message, and the earlier test should actually have failed (false positive). We avoid this difficulty by enforcing that the receiver q is synchronized with the corresponding sender along channel c on emptiness tests, by adding to the test action $c\mathtt{==}\varepsilon$ by q a zero test $\mathtt{x}_c^q\mathtt{==0}$.

Formally, let $\mathcal{S} = \langle \mathcal{T}, M, \mathbb{D}, (\mathcal{A}^p)_{p\in P}\rangle$ with $\mathbb{D} = \{\tau\}$ be a system of communicating tick automata over topology $\mathcal{T} = \langle P, C, E\rangle$, where, for each $p \in P$, $\mathcal{A}^p = \langle L^p, L_I^p, L_F^p, A^p, \to^p\rangle$ is a tick automaton, i.e., $\tau \in A^p$. We define the system of communicating counter automata $\mathcal{S}' = \langle \mathcal{T}, M', \mathbb{D}', (\llbracket \mathcal{C}^p \rrbracket)_{p\in P}\rangle$, *over the same topology* $\mathcal{T}$ *as* $\mathcal{S}$, s.t. $M' = M \cup \{\tau\}$, $\mathbb{D}' = \emptyset$, and, for every process $p \in P$, we have a counter automaton $\mathcal{C}^p$, which is defined as follows: $\mathcal{C}^p = \langle L^p, L_I^p, L_F^p, B^p, \mathtt{X}^p, \Delta^p\rangle$, where control locations L^p, initial locations L_I^p, and final locations L_F^p are the same as in the corresponding tick automaton $\mathcal{A}^p$, and counters are those in $\mathtt{X}^p = \{\mathtt{x}_c^p \mid c \in C^{-1}[p]\}$. For simplifying the definition of transitions, we allow sequences of actions instead of just one action—these can be clearly implemented by introducing more intermediate states. Transitions in Δ^p for $\mathcal{C}^p$ are defined as follows:

- Let $\ell \xrightarrow{\tau} \ell'$ be a transition in $\mathcal{A}^p$, and assume that outgoing channels of p are those in $C[p] = \{c_0, \ldots, c_h\}$, and that counters in $\mathtt{X}^p$ are those in $\{x_0, \ldots, x_k\}$. Then, $\ell \xrightarrow{c_0!\tau;\ldots;c_h!\tau;x_0\mathtt{++};\ldots;x_k\mathtt{++}} \ell'$ is a transition in $\mathcal{C}^p$.
- For every $\ell \in L^p$ and input channel $c \in C^{-1}[p]$, there is a transition $\ell \xrightarrow{c?\tau;\mathtt{x}_c^p\mathtt{--}} \ell$ in $\mathcal{C}^p$.
- If $\ell \xrightarrow{c\mathtt{==}\varepsilon} \ell'$ is a transition in $\mathcal{A}^p$, then $\ell \xrightarrow{\mathtt{x}_c^p\mathtt{==0};c\mathtt{==}\varepsilon} \ell'$ is a transition in $\mathcal{C}^p$.
- Every other transition $\ell \xrightarrow{a} \ell'$ in $\mathcal{A}^p$ is also a transition in $\mathcal{C}^p$.

The action alphabet of $\mathcal{C}^p$ is thus $B^p = (A^p \setminus \{\tau\}) \cup \{c?\tau \mid c \in C^{-1}[p]\} \cup \{c!\tau \mid c \in C[p]\}$; in particular, τ is no longer an action, but a message that can be sent and received. We show that $\mathcal{S}$ and $\mathcal{S}'$ are equivalent, obtaining the following result.

Proposition 1. *Let $\mathcal{T}$ be a weakly-connected topology with α channels, of which β can be tested for emptiness. For every system of communicating tick automata $\mathcal{S}$ with topology $\mathcal{T}$, we can produce, in linear time, an equivalent system of communicating counter automata $\mathcal{S}'$ with the same topology $\mathcal{T}$, containing α counters, of which β can be tested for zero.*

While the proposition above holds for arbitrary weakly-connected topologies, it yields counter automata *with channels*, which are undecidable in general. To

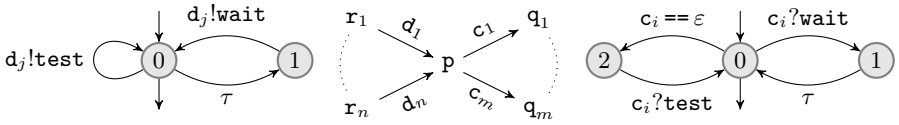

Fig. 1. Simulation of a counter automaton by a system of communicating tick automata: Tick automata for $\mathtt{r}_j$ (left) and $\mathtt{q}_i$ (right), Topology (middle)

avoid undecidability due to communication, we need to forbid cycles (either directed or undirected) in the topology. It has been shown that, on polytrees[4], runs of communicating processes (even infinite-state) can be rescheduled as to satisfy the so-called *eagerness* requirement, where each transmission is *immediately* followed by the matching reception [18]. Their argument holds also in the presence of emptiness tests, since an eager run cannot disable $c == \varepsilon$ transitions (eager runs can only make the channels empty more often). Thus, by restricting to eager runs, communication behaves just as a rendezvous synchronization, and we obtain a global counter automaton by taking the product of all component counter automata.

Theorem 1. *For every polytree topology $\mathcal{T}$ with α channels, of which β can be tested for emptiness, the reachability problem for systems of communicating tick automata with topology $\mathcal{T}$ is reducible, in linear time, to the reachability problem for products of (non-communicating) counter automata, with overall α counters, of which β can be tested for zero.*

From counter automata to tick automata. We reduce the reachability problem for (non-communicating) counter automata to the reachability problem for systems of communicating tick automata with star topology. Formally, a topology $\mathcal{T} = \langle P, C, E\rangle$ is called a *star topology* if there exist two disjoint subsets Q, R of P and a process p in $P \setminus (Q \cup R)$ such that $P = \{p\} \cup Q \cup R$ and $C = (R \times \{p\}) \cup (\{p\} \times Q)$. The idea is to simulate each counter with a separate channel, thus the number of counters fixes the number of channels in $\mathcal{T}$. However, our reduction is uniform in the sense that it works independently of the exact arrangement of channels in $\mathcal{T}$, which we take *not* to be under our control. W.l.o.g., we consider counter automata where all actions are counter operations (i.e., $\Delta \subseteq L \times \mathtt{Op}(X) \times L$).

For the remainder of this section, we consider an arbitrary star topology $\mathcal{T} = \langle P, C, E\rangle$ with set of processes $P = \{\mathtt{p}, \mathtt{q}_1, \ldots, \mathtt{q}_m, \mathtt{r}_1, \ldots, \mathtt{r}_n\}$, where $m, n \in \mathbb{N}$, and set of channels $C = \{\mathtt{p}\} \times \{\mathtt{q}_1, \ldots, \mathtt{q}_m\} \cup \{\mathtt{r}_1, \ldots, \mathtt{r}_n\} \times \{\mathtt{p}\}$ and $E = C$. This topology is depicted in Figure 1 (middle). Note that we allow the limit cases $m = 0$ and $n = 0$. To simplify the presentation, we introduce shorter notations for the channels of this topology: we define $\mathtt{c}_i = (\mathtt{p}, \mathtt{q}_i)$ and $\mathtt{d}_j = (\mathtt{r}_j, \mathtt{p})$ for every $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$.

[4] A *polytree* is a weakly-connected graph with neither directed, nor undirected cycles.

Let $\mathcal{C} = \langle L, L_I, L_F, X \cup Y, \Delta \rangle$ be a counter automaton with $m + n$ counters, namely $X = \{\mathtt{x}_1, \ldots, \mathtt{x}_m\}$ and $Y = \{\mathtt{y}_1, \ldots, \mathtt{y}_n\}$. The counters are split into X and Y to reflect the star topology $\mathcal{T}$, which is given a priori. We build, from $\mathcal{C}$, an equivalent system of communicating tick automata $\mathcal{S}$ with topology $\mathcal{T}$. Basically, the process $\mathtt{p}$ simulates the control-flow graph of the counter automaton, and the counters $\mathtt{x}_i$ and $\mathtt{y}_j$ are simulated by the channels $\mathtt{c}_i$ and $\mathtt{d}_j$, respectively. In order to define $\mathcal{S}$, we need to provide its message alphabet and its tick automata, one for each process p in P. The message alphabet is $M = \{\mathtt{wait}, \mathtt{test}\}$. Actions performed by processes in P are either communication actions or the delay action τ. Processes $\mathtt{r}_j$'s are assigned the tick automaton of Figure 1 (left), and processes $\mathtt{q}_i$'s are assigned the tick automaton of Figure 1 (right). Intuitively, communications on `wait` messages are loosely synchronized using the τ actions in $\mathtt{q}_i$ and $\mathtt{r}_j$, so that $\mathtt{p}$ can control the rate of their reception and transmission.

We now present the tick automaton $\mathcal{A}^{\mathtt{p}}$. As mentioned above, the control-flow graph of $\mathcal{C}$ is preserved by $\mathcal{A}^{\mathtt{p}}$, so we only need to translate counter operations of $\mathcal{C}$ by communication actions and τ actions. Each counter operation of $\mathcal{C}$ is simulated by a finite sequence of actions in $\Sigma^{\mathtt{p}}$. To simplify the presentation, we directly label transitions of $\mathcal{A}^{\mathtt{p}}$ by words in $(\Sigma^{\mathtt{p}})^*$. The encoding of counter operations is given by the mapping η from $\mathtt{Op}(X \cup Y)$ to $(\Sigma^{\mathtt{p}})^*$ defined as follows:

$$\begin{array}{rlrl}
\eta(\mathtt{x}_i\mathtt{++}) &= \mathtt{c}_i!\mathtt{wait} & \eta(\mathtt{x}_i\mathtt{--}) &= (\mathtt{c}_h!\mathtt{wait})_{1\le h\le m, h\neq i} \cdot \tau \cdot (\mathtt{d}_k?\mathtt{wait})_{1\le k\le n} \\
\eta(\mathtt{y}_j\mathtt{--}) &= \mathtt{d}_j?\mathtt{wait} & \eta(\mathtt{y}_j\mathtt{++}) &= (\mathtt{c}_h!\mathtt{wait})_{1\le h\le m} \cdot \tau \cdot (\mathtt{d}_k?\mathtt{wait})_{1\le k\le n, k\neq j} \\
\eta(\mathtt{x}_i\mathtt{==0}) &= \mathtt{c}_i!\mathtt{test} & \eta(\mathtt{y}_j\mathtt{==0}) &= (\mathtt{d}_j\mathtt{==}\varepsilon) \cdot (\mathtt{d}_j?\mathtt{test})
\end{array}$$

where $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. We obtain $\mathcal{A}^{\mathtt{p}}$ from $\mathcal{C}$ by replacing each counter operation by its encoding. Observe that these replacements require the addition of a set $S^{\mathtt{p}}_{\diamond}$ of fresh intermediate states to implement sequences of actions. Formally, $\mathcal{A}^{\mathtt{p}}$ is the tick automaton $\mathcal{A}^{\mathtt{p}} = \langle L \cup S^{\mathtt{p}}_{\diamond}, L_I, L_F, \Sigma^{\mathtt{p}}, \{\ell \xrightarrow{\eta(\mathtt{op})} \ell' \mid (\ell, \mathtt{op}, \ell') \in \Delta\} \rangle$. This completes the definition of the system of communicating tick automata $\mathcal{S} = \langle \mathcal{T}, M, \{\tau\}, (\mathcal{A}^p)_{p \in P} \rangle$.

Let us show that $[\![\mathcal{C}]\!]$ has a run if and only if $[\![\mathcal{S}]\!]$ has a run. We only explain the main ideas behind this simulation of $\mathcal{C}$ by $\mathcal{S}$. The number of `wait` messages in channels $\mathtt{c}_i$ and $\mathtt{d}_j$ encodes the value of counters $\mathtt{x}_i$ and $\mathtt{y}_j$, respectively. So, incrementing $\mathtt{x}_i$ amounts to sending `wait` in $\mathtt{c}_i$, and decrementing $\mathtt{y}_j$ amounts to receiving `wait` from $\mathtt{d}_j$. Both actions can be performed by $\mathtt{p}$. Decrementing $\mathtt{x}_i$ is more involved, since $\mathtt{p}$ cannot receive from the channel $\mathtt{c}_i$. Instead, $\mathtt{p}$ performs a τ action in order to force a τ action in $\mathtt{q}_i$, hence, a receive of `wait` by $\mathtt{q}_i$. But all other processes also perform the τ action, so $\mathtt{p}$ compensates (see the definition of $\eta(\mathtt{x}_i\mathtt{--})$) in order to preserve the number of `wait` messages in the other channels. The simulation of $\mathtt{y}_j\mathtt{++}$ by $\eta(\mathtt{y}_j\mathtt{++})$ is similar. Let us now look at zero test operations. When $\mathtt{p}$ simulates $\mathtt{x}_i\mathtt{==0}$, it simply sends `test` in the channel $\mathtt{c}_i$. This message is eventually received by $\mathtt{q}_i$ since all channels must be empty at the end of the simulation. The construction guarantees that the first receive action of $\mathtt{q}_i$ after the send action $\mathtt{c}_i!\mathtt{test}$ of $\mathtt{p}$ is the matching receive $\mathtt{c}_i?\mathtt{test}$. This means, in particular, that the channel is empty when $\mathtt{p}$ sends `test` in $\mathtt{c}_i$. The same device is used to simulate a zero test of $\mathtt{y}_j$, except that the roles

of $\mathtt{p}$ and its peer (here, $\mathtt{r}_j$) are reversed. Clearly, channels that need to be tested for emptiness are those encoding counters that are tested for zero. We obtain the following theorem.

Theorem 2. *Let $\mathcal{T}$ be an a priori given star topology with α channels, of which β can be tested for emptiness. The reachability problem for (non-communicating) counter automata with α counters, of which β can be tested for zero, is reducible, in linear time, to the reachability problem for systems of communicating tick automata with topology $\mathcal{T}$.*

Decidability and Complexity Results for Communicating Tick Automata. Thanks to the mutual reductions to/from counter automata developed previously, we may now completely characterize which topologies (not necessarily weakly-connected) have a decidable reachability problem, depending on exactly which channels can be tested for emptiness. Intuitively, decidability holds even in the presence of multiple emptiness tests, provided that each test appears in a different weakly-connected component.

Theorem 3 (Decidability). *Given a topology $\mathcal{T}$, the reachability problem for systems of communicating tick automata with topology $\mathcal{T}$ is decidable if and only if $\mathcal{T}$ is a polyforest*[5] *containing at most one testable channel in each weakly-connected component.*

Proof. For one direction, assume that the reachability problem for systems of communicating tick automata with topology $\mathcal{T}$ is decidable. The topology $\mathcal{T}$ is necessarily a polyforest, since the reachability problem is undecidable for non-polyforest topologies even without ticks [23,21]. Suppose that $\mathcal{T}$ contains a weakly-connected component with (at least) two channels that can be tested for emptiness. By an immediate extension of Theorem 2 to account for the undirected path between these two channels, we can reduce the reachability problem for two-counter automata to the reachability problem for systems of communicating tick automata with topology $\mathcal{T}$. Since the former is undecidable, each weakly-connected component in $\mathcal{T}$ contains at most one testable channel.

For the other direction, assume that $\mathcal{T}$ is a polyforest with at most one testable channel in each weakly-connected component, and let $\mathcal{S}$ be a system of communicating tick automata with topology $\mathcal{T}$. Thus, $\mathcal{S}$ can be decomposed into a disjoint union of independent systems $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_n$, where each $\mathcal{S}_k$ has an undirected tree topology containing exactly one testable channel. But we need to ensure that the $\mathcal{S}_k$'s perform the same number of ticks. By (the construction leading to) Theorem 1, each $\mathcal{S}_k$ can be transformed into an equivalent counter automaton $\mathcal{C}_k$ (by taking the product over all processes in $\mathcal{S}_k$), where exactly one counter, let us call it $\mathtt{x}_k$, can be tested for zero. We may suppose, w.l.o.g., that the counters of $\mathcal{C}_0, \ldots, \mathcal{C}_n$ are disjoint. Moreover, $\mathcal{C}_k$ can maintain, in an extra counter $\mathtt{y}_k$, the number of ticks performed by $\mathcal{S}_k$. We compose the counter machines $\mathcal{C}_0, \ldots, \mathcal{C}_n$ sequentially, and check, at the end, that $\mathtt{y}_0 = \cdots = \mathtt{y}_n$. Since

[5] A topology $\mathcal{T}$ is a *polyforest* if it is a directed acyclic graph with no undirected cycle.

all counters must be zero in final configurations, this check can be performed by adding, on the final state, a loop decrementing all the $\mathtt{y}_k$'s simultaneously. The construction guarantees that the resulting global counter machine $\mathcal{C}$ is equivalent to $\mathcal{S}$. However, $\mathcal{C}$ contains zero tests on many counters: $\mathtt{x}_0, \ldots, \mathtt{x}_n$. Fortunately, these counters are used one after the other, and they are zero at the beginning and at the end. Therefore, we may reuse $\mathtt{x}_0$ in place of $\mathtt{x}_1, \ldots, \mathtt{x}_n$. We only need to check that $\mathtt{x}_0$ is zero when switching from $\mathcal{C}_k$ to $\mathcal{C}_{k+1}$. Thus, we have reduced the reachability problem for systems of communicating tick automata with topology $\mathcal{T}$ to the reachability problem for counter automata where only one counter can be tested for zero. As the latter is decidable [24,11], the former is decidable, too.

When no test is allowed, we obtain a simple characterization of the complexity for polyforest topologies. A topology $\mathcal{T} = \langle P, C, E \rangle$ is *test-free* if $E = \emptyset$.

Corollary 1 (Complexity). *The reachability problem for systems of communicating tick automata with test-free polyforest topologies has the same complexity as the reachability problem for counter automata without zero tests (equivalently, Petri nets).*

Remark 1. Even though global synchronization makes communicating tick automata more expressive than CFSMs, our characterization shows that they are decidable for exactly the same topologies (polyforest). However, while reachability for CFSMs is PSPACE-complete, systems of communicating tick automata are equivalent to Petri nets, for which reachability is EXPSPACE-hard [22] (the upper bound being a long-standing open problem).

4 Decidability of Communicating Timed Automata

In this section, we consider communicating timed automata, which are communicating timed processes synchronizing over the dense delay domain $\mathbb{D} = \mathbb{R}_{\geq 0}$. We extend the decidability results for tick automata of Section 3 to the case of timed automata. To this end, we present mutual, topology-preserving reductions between communicating tick automata and communicating timed automata. We first introduce the latter model.

Communicating Timed Automata. A *timed automaton* $\mathcal{B} = \langle L, L_I, L_F, X, \Sigma, \Delta \rangle$ is defined by a finite set of locations L with initial locations $L_I \subseteq L$ and final locations $L_F \subseteq L$, a finite set of *clocks* X, a finite alphabet Σ and a finite set Δ of transitions rules $(\ell, \sigma, g, R, \ell')$ where $\ell, \ell' \in L$, $\sigma \in \Sigma$, the guard g is a conjunction of constraints $x \# c$ for $x \in X$, $\# \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$, and $R \subseteq X$ is a set of clocks to reset.

The semantics of $\mathcal{B}$ is given by the timed process $[\![\mathcal{B}]\!] = \langle S, S_I, S_F, A, \rightarrow \rangle$, where $S = L \times \mathbb{R}_{\geq 0}^X$, $S_I = L_I \times \{\lambda x.\, 0\}$, $S_F = L_F \times \{\lambda x.\, 0\}$, $A = \Sigma \cup \mathbb{R}_{\geq 0}$ is the set of actions, and there is a transition $(\ell, v) \xrightarrow{d} (\ell, v')$ if $d \in \mathbb{R}_{\geq 0}$ and $v'(x) = v(x) + d$ for every clock x, and $(\ell, v) \xrightarrow{\sigma} (\ell', v')$ if there exists

a rule $(\ell, \sigma, g, R, \ell') \in \Delta$ such that g is satisfied by v (defined in the natural way) and $v'(x) = 0$ when $x \in R$, $v'(x) = v(x)$ otherwise. We decorate a path $(\ell_0, u_0) \xrightarrow{a_0,t_0} (\ell_1, u_1) \xrightarrow{a_1,t_1} \cdots (a_n, u_n)$ in $[\![\mathcal{B}]\!]$ with additional *timestamps* $t_i = \sum\{a_j \mid j = 0, \ldots, i-1 \text{ and } a_j \in \mathbb{R}_{\geq 0}\}$. Note that we require clocks to be zero in final configurations, as this simplifies the forthcoming construction from tick automata to timed automata. It can be implemented by duplicating final locations, and by resetting all clocks when entering the new final locations. Without loss of generality, we do not consider location invariants as they can be encoded in the guards. To ensure that the invariant of the last state in a run is satisfied, we duplicate the final locations and we add an edge guarded by the invariant, to the new accepting copy. Combining the two constructions, we get that the invariant in the final configuration is satisfied as the clocks have value zero in accepting configurations.

A *system of communicating timed automata* is a system of communicating timed processes $\mathcal{S} = \langle \mathcal{T}, M, \mathbb{R}_{\geq 0}, ([\![\mathcal{B}^p]\!])_{p \in P} \rangle$ where each $\mathcal{B}^p$ is a timed automaton. Note that each timed automaton has access only to its local clocks. By Definition 1, each timed automaton performs communicating actions in A^p_{com} and synchronizes with all the other processes over delay actions in $\mathbb{R}_{\geq 0}$.

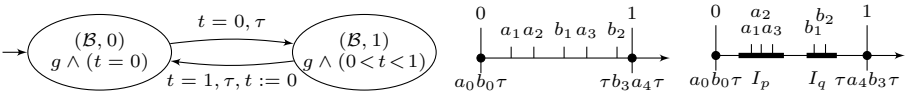

Fig. 2. From timed to tick automata: instrumentation of a timed automaton $\mathcal{B}$ with τ-transitions (left), addition of τ's along a run (middle) and rescheduling of a run (right)

From Timed Automata to Tick Automata. On test-free acyclic topologies, we show a topology-preserving reduction from communicating timed to communicating tick automata. We insist on a reduction that only manipulates processes locally, thus preserving the topology. The absence of emptiness tests on the channels enables such a modular construction.

Naïvely, one would just apply the classical region construction to each process [9]. However, while this preserves local reachability properties, it does not respect the global synchronization between different processes. While quantitative synchronization cannot be obtained by locally removing dense time, a qualitative synchronization suffices in our setting. We require that all processes are either at the same integer date $k \in \mathbb{N}$, or in the same open interval $(k, k+1)$. This suffices because, at integer dates (in fact, at any time-point), any interleaving is allowed, and, in intervals $(k, k+1)$, we can reschedule all processes s.t., for every channel $c = (p, q)$, all actions of p occur before all actions of q (cf. the Rescheduling Lemma below). The latter property ensures the causality between transmissions and receptions.

Qualitative synchronization is achieved by forcing each automaton $\mathcal{B}^p$ to perform a synchronizing tick action τ at each date k and at each interval $(k, k+1)$. See Figure 2 on the left, where $\mathcal{B}^p$ is split into two copies $(\mathcal{B}^p, 0)$ and $(\mathcal{B}^p, 1)$: Actions occurring on integer dates k are performed in $(\mathcal{B}^p, 0)$, and those in $(k, k+1)$ happen in $(\mathcal{B}^p, 1)$. This is ensured by adding a new clock $\mathtt{t}$ and τ-transitions that switch from one mode to the other. Formally, the *τ-instrumentation* of $\mathcal{B} = \langle L, L_I, L_F, X, \Sigma, \Delta\rangle$ is the timed automaton $\mathsf{Instr}(\mathcal{B}, \tau) = \langle L \times \{0,1\}, L_I \times \{1\}, F \times \{0,1\}, X \cup \{\mathtt{t}\}, \Sigma \cup \{\tau\}, \Delta'\rangle$, where $\mathtt{t} \notin X$ and Δ' is defined by: $(\ell, 0) \xrightarrow{a,(g \wedge \mathtt{t}=0),R} (\ell', 0)$ and $(\ell, 1) \xrightarrow{a,(g \wedge 0<\mathtt{t}<1),R} (\ell', 1)$ for all rules $\ell \xrightarrow{a,g,R} \ell'$ in Δ, and $(\ell, 0) \xrightarrow{\tau,\mathtt{t}=0,\emptyset} (\ell, 1)$ and $(\ell, 1) \xrightarrow{\tau,\mathtt{t}=1,\{\mathtt{t}\}} (\ell, 0)$ for all locations $\ell \in L$.

Finally, we obtain an equivalent system of tick automata by applying the exponential region construction to each instrumented process.

Theorem 4. *Let $\mathcal{T}$ be a test-free acyclic topology. For every system of communicating timed automata $\mathcal{S} = \langle \mathcal{T}, M, \mathbb{R}_{\geq 0}, (\llbracket \mathcal{B}^p \rrbracket)_{p \in P}\rangle$ with topology $\mathcal{T}$, we can produce, in exponential time, an equivalent system of communicating tick automata $\mathcal{S}' = \langle \mathcal{T}, M, \{\tau\}, (\mathcal{A}^p)_{p \in P}\rangle$ over the same topology $\mathcal{T}$, where the tick automaton $\mathcal{A}^p$ is obtained by applying the region graph construction to $\mathsf{Instr}(\mathcal{B}^p, \tau)$.*

One direction of the equivalence between $\mathcal{S}$ and $\mathcal{S}'$ is immediate, since every run in $\mathcal{S}$ induces a run in $\mathcal{S}'$ by just inserting τ actions in the right position. For the other direction, let ρ' be a run of $\mathcal{S}'$, and we show how to build a corresponding run ρ of $\mathcal{S}$. We have to schedule all the actions in ρ' on timestamps that are consistent with the guards in $\mathcal{S}$ and that preserve dependencies between transmissions and receptions of messages. Consider a channel $c = (p, q)$ without emptiness test. If p and q are untimed processes, it is always possible to first schedule transmissions of p, and then receptions of q. The Rescheduling Lemma below ensures the same for timed processes. This is depicted in Figure 2 in the middle (before rescheduling) and on the right (after rescheduling) where the a's are emissions of p and the b's are receptions of q.

Lemma 1 (Rescheduling Lemma). *Let $\mathcal{B}$ be a timed automaton, and $I \subseteq (0,1)$ an open interval. Then, every run of $\mathcal{B}$ $(\ell_0, v_0) \xrightarrow{a_0,t_0} \cdots (\ell_n, v_n)$ can be rescheduled such that integral timestamps $t_i \in \mathbb{N}$ are kept the same, and non-integral timestamps $t_i \in (k, k+1)$ belong to $k + I$.*

Intuitively, the lemma above allows us to restrict non-integer timestamps in $(k, k+1)$ to occur in a predefined sub-interval $I+k$. Let us first see how this helps in constructing ρ'. To each process p, we associate an open interval $I_p \subseteq (0,1)$, such that, for every channel (p, q), I_p and I_q are disjoint, and I_p comes before I_q. This is always possible on acyclic topologies. Then, all actions of process p in $(k, k+1)$ are rescheduled to occur in $k + I_p$ (according to the Rescheduling Lemma), which ensures causality between transmissions and receptions. Finally, the τ actions added by instrumentation tell, for each action performed by process p in ρ', whether it should be scheduled at an integer date k, or in $k + I_p$.

Remark 2. Our reduction is incorrect in the presence of emptiness tests. There are essential difficulties in rescheduling senders and receivers in fixed intervals, as emptiness tests introduce a sort of circular dependency and seem to require unboundedly many intervals.

We now comment about the correctness of the Rescheduling Lemma. Resets and guards in a timed automaton allow to enforce minimal and/or maximal delays between timestamps on a path. Since clocks are compared to integers only, it suffices to just distinguish between integral and non-integral dates. While for closed guards like $x \leq 1$ a non-integral time-point $t \in (0,1)$ would suffice to represent all non-integral dates, to accommodate open guards like $x < 1$ we need a dense interval $I \subseteq (0,1)$. The following characterization of decidable test-free topologies follows from Theorems 3 and 4.

Theorem 5 (Decidability). *Given a test-free topology $\mathcal{T}$, the reachability problem for systems of communicating timed automata with topology $\mathcal{T}$ is decidable if and only if $\mathcal{T}$ is a polyforest.*

Remark 3. While the reachability problem is known to be decidable for a system of two communicating timed automata with only one channel and emptiness test [20], that proof does not preserve the topology and it looks hardly adaptable to arbitrary polyforest topologies.

From tick automata to timed automata. Given a system of communicating tick automata $\mathcal{S}$, we produce an equivalent system of communicating timed automata $\mathcal{S}'$, over the same topology. The synchronization on τ's is easily simulated using clocks in $\mathcal{S}'$ by ensuring that all the processes elapse 1 time unit exactly when they (synchronously) perform a τ in $\mathcal{S}$. Thus, every run in $\mathcal{S}$ has a corresponding run in $\mathcal{S}'$. For the converse to hold, we have to make sure that for every run of $\mathcal{S}'$, all the processes perform the same number of τ's on the corresponding run of $\mathcal{S}$. This is ensured since we require clocks to be zero at the end of accepting runs, thus preventing time from elapsing on final locations.

The simple topology $p \to q \to r$ is known to be undecidable when both channels can be tested for emptiness [20]. Thanks to Theorem 3, we obtain generalized undecidability for every weakly-connected topology containing at least two testable channels.

Theorem 6 (Undecidability). *Given a weakly-connected topology $\mathcal{T}$ with two testable channels, the reachability problem for systems of communicating timed automata with topology $\mathcal{T}$ is undecidable.*

5 Conclusions and Future Work

We have studied the decidability and complexity of communicating timed processes. In discrete time, we give a complete characterization of decidable topologies with emptiness tests, as well as a tight connection with Petri nets in the test-free case. In dense time, we prove decidability for polyforest test-free topologies,

and we generalize the undecidability results of [20] to arbitrary weakly-connected topologies containing two testable channels. We leave open whether one can obtain, in the presence of emptiness tests, the same characterization as in discrete time. We conjecture that this is possible, although the techniques used here do not seem to easily extend to deal with emptiness tests. Finally, as another direction for future work one can study richer models where processes are allowed to send timestamps or clocks along channels, in the spirit of [1].

Acknowledgements. The authors wish to thank Jérôme Leroux, Anca Muscholl, and Igor Walukiewicz for helpful discussions. We also thank the anonymous referees for their useful comments and suggestions.

References

1. Abdulla, P.A., Atig, M.F., Cederberg, J.: Timed lossy channel systems. In: FSTTCS. LIPIcs (2012)
2. Abdulla, P.A., Atig, M.F., Stenman, J.: Dense-timed pushdown automata. In: LICS, pp. 35–44 (2012)
3. Abdulla, P.A., Jonsson, B.: Verifying programs with unreliable channels. Information and Computation 127(2), 91–101 (1996)
4. Abdulla, P.A., Nylén, A.: Timed Petri Nets and BQOs. In: Colom, J.-M., Koutny, M. (eds.) ICATPN 2001. LNCS, vol. 2075, pp. 53–70. Springer, Heidelberg (2001)
5. Abdulla, P.A., Delzanno, G., Rezine, O., Sangnier, A., Traverso, R.: On the Verification of Timed Ad Hoc Networks. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 256–270. Springer, Heidelberg (2011)
6. Akshay, S., Bollig, B., Gastin, P.: Automata and Logics for Timed Message Sequence Charts. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 290–302. Springer, Heidelberg (2007)
7. Akshay, S., Bollig, B., Gastin, P., Mukund, M., Narayan Kumar, K.: Distributed Timed Automata with Independently Evolving Clocks. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 82–97. Springer, Heidelberg (2008)
8. Akshay, S., Gastin, P., Mukund, M., Kumar, K.N.: Model checking time-constrained scenario-based specifications. In: FSTTCS. LIPIcs, vol. 8, pp. 204–215 (2010)
9. Alur, R., Dill, D.: A theory of timed automata. TCS 126(2), 183–235 (1994)
10. Bérard, B., Cassez, F., Haddad, S., Lime, D., Roux, O.H.: Comparison of Different Semantics for Time Petri Nets. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 293–307. Springer, Heidelberg (2005), http://dx.doi.org/10.1007/11562948_/23
11. Bonnet, R.: The Reachability Problem for Vector Addition System with One Zero-Test. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 145–157. Springer, Heidelberg (2011)
12. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30(2), 323–342 (1983)
13. Cécé, G., Finkel, A.: Verification of programs with half-duplex communication. Information and Computation 202(2), 166–190 (2005)

14. Chambart, P., Schnoebelen, P.: Mixing Lossy and Perfect Fifo Channels. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 340–355. Springer, Heidelberg (2008)
15. Chandrasekaran, P., Mukund, M.: Matching Scenarios with Timing Constraints. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 98–112. Springer, Heidelberg (2006)
16. Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized Verification of Ad Hoc Networks. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 313–327. Springer, Heidelberg (2010)
17. Gruber, H., Holzer, M., Kiehn, A., König, B.: On Timed Automata with Discrete Time – Structural and Language Theoretical Characterization. In: De Felice, C., Restivo, A. (eds.) DLT 2005. LNCS, vol. 3572, pp. 272–283. Springer, Heidelberg (2005)
18. Heußner, A., Leroux, J., Muscholl, A., Sutre, G.: Reachability analysis of communicating pushdown systems. Logical Methods in Comp. Sci. 8(3:23), 1–20 (2012)
19. Ibarra, O.H., Dang, Z., Pietro, P.S.: Verification in loosely synchronous queue-connected discrete timed automata. Theor. Comput. Sci. 290(3), 1713–1735 (2003)
20. Krcal, P., Yi, W.: Communicating Timed Automata: The More Synchronous, the More Difficult to Verify. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 249–262. Springer, Heidelberg (2006)
21. La Torre, S., Madhusudan, P., Parlato, G.: Context-Bounded Analysis of Concurrent Queue Systems. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 299–314. Springer, Heidelberg (2008)
22. Lipton, R.J.: The Reachability Problem Requires Exponential Space. Department of Computer Science, Yale University (1976)
23. Pachl, J.K.: Reachability problems for communicating finite state machines. Research Report CS-82-12, University of Waterloo (May 1982)
24. Reinhardt, K.: Reachability in petri nets with inhibitor arcs. ENTCS 223, 239–264 (2008)