Accuracy of Message Counting Abstraction in Fault-Tolerant Distributed Algorithms*

Igor Konnov[1], Josef Widder[1], Francesco Spegni[2], and Luca Spalazzi[2]

[1] TU Wien (Vienna University of Technology), Austria
[2] UnivPM, Ancona, Italy

Abstract. Fault-tolerant distributed algorithms are a vital part of mission-critical distributed systems. In principle, automatic verification can be used to ensure the absence of bugs in such algorithms. In practice however, model checking tools will only establish the correctness of distributed algorithms if message passing is encoded efficiently. In this paper, we consider abstractions suitable for many fault-tolerant distributed algorithms that count messages for comparison against thresholds, e.g., the size of a majority of processes. Our experience shows that storing only the numbers of sent and received messages in the global state is more efficient than explicitly modeling message buffers or sets of messages. Storing only the numbers is called message-counting abstraction. Intuitively, this abstraction should maintain all necessary information.
In this paper, we confirm this intuition for asynchronous systems by showing that the abstract system is bisimilar to the concrete system. Surprisingly, if there are real-time constraints on message delivery (as assumed in fault-tolerant clock synchronization algorithms), then there exist neither timed bisimulation, nor time-abstracting bisimulation. Still, we prove this abstraction useful for model checking: it preserves ATCTL properties, as the abstract and the concrete models simulate each other.

1 Introduction

The following algorithmic idea is pervasive in fault-tolerant distributed computing [36, 13, 30, 33, 21, 39]: each correct process counts messages received from distinct peers. Then, given the total number of processes n and the maximum number of faulty processes t, a process performs certain actions only if the message counter reaches a threshold such as $n - t$ (this number ensures that faulty processes alone cannot prevent progress in the computation). A list of benchmark algorithms that use such thresholds can be found in [27]. On the left of Figure 1, we give an example pseudo code [36]. This algorithm works in a timed environment [35] (with a time bound τ^+ on message delays) in the presence of Byzantine faults ($n > 3t$) and provides safety and liveness guarantees such as:

* Supported by: the Austrian Science Fund (FWF) through the National Research Network RiSE (S11403 and S11405), and project PRAVDA (P27722); and by the Vienna Science and Technology Fund (WWTF) through project APALACHE (ICT15-103).

```
1  local myval_i ∈ {0,1}
2
3
4
5
6  do atomically
7   -- messages are received implicitly
8   if myval_i = 1
9    and not sent ECHO before
10  then send ECHO to all
11
12  if received ECHO
13     from at least t + 1 distinct processes
14   and not sent ECHO before
15  then send ECHO to all
16
17  if received ECHO
18     from at least n − t distinct processes
19  then accept
20 od
```

```
21 local myval_i ∈ {0,1}
22 global nsntEcho ∈ ℕ_0 initially 0
23 local hasSent ∈ 𝔹 initially F
24 local rcvdEcho ∈ ℕ_0 initially 0
25
26 do atomically
27  if (*) -- choose non-deterministically
28   and rcvdEcho < nsntEcho + f
29  then rcvdEcho++;
30
31  if myval_i = 1 and hasSent = F
32  then { nsntEcho++; hasSent = T; }
33
34
35  if rcvdEcho ≥ t + 1 and hasSent = F
36  then { nsntEcho++; hasSent = T; }
37
38  if rcvdEcho ≥ n − t
39  then accept
40 od
```

Fig. 1. Pseudocode of a broadcast primitive to simulate authenticated broadcast [36] (left), and pseudocode of its message-counting abstraction (right)

a) If a correct process accepts (that is, executes Line 19) at time T, then all correct processes accept by time $T + 2\tau^+$.

b) If all correct processes start with $myval_i = 0$, then no correct process ever *accepts*.

c) If all correct processes start with $myval_i = 1$, then at least one correct process eventually accepts.

As is typical for the distributed algorithms literature, the pseudo code from Figure 1 omits "unnecessary book-keeping" details of message passing. That is, neither the local data structures that store the received messages nor the message buffers are explicitly described. Hence, if we want to automatically verify such an algorithm design, it is up to a verification expert to find adequate modeling and proper abstractions of message passing.

The authors of [23] suggested to model message passing using message counters instead of keeping track of individual messages. This modeling was shown experimentally to be efficient for fixed size systems, and later a series of parameterized model checking techniques was based upon it [22, 23, 25–27]. The encoding on the right of Figure 1 is obtained by adding a global integer variable `nsntEcho`. Incrementing this variable (Line 36) encodes that a correct process executes Line 15 of the original pseudo code. The ith process keeps the number of received messages in a local integer variable $\texttt{rcvdEcho}_i$ that can be increased, as long as the invariant $\texttt{rcvdEcho}_i \leq \texttt{nsntEcho} + f$ is preserved, where f is the actual number of Byzantine faulty processes in the run. (This models that correct processes can receive up to f messages sent by faulty processes.) In fact, this modeling can be seen as a *message-counting abstraction* of a distributed system that uses message buffers.

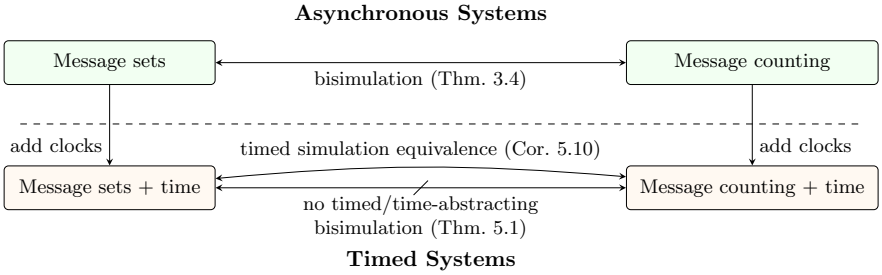

Fig. 2. Relationship between different modeling choices.

The broadcast primitive in Figure 1 is also used in the seminal clock synchronization algorithm from [35]. For clock synchronization, the precision of the clocks depends on the timing behavior[3] of the message system that the processes use to re-synchronize; e.g., in [35] it is required that each message sent at an instant T by a correct process must be delivered by a correct recipient process in the time interval $[T+\tau^-, T+\tau^+]$ for some bounds τ^- and τ^+ fixed in each run.

The standard theory of timed automata [7] does not account for message passing directly. To incorporate messages, one specifies a message passing system as a network of timed automata, i.e., a collection of timed automata that are scheduled with respect to interleaving semantics and interact via rendezvous, synchronous broadcast, or shared variables [12]. In this case, there are two typical ways to encode message passing: (i) for each pair of processes, introduce a separate timed automaton that models a channel between the processes, or (ii) introduce a single timed automaton that stores messages from timed automata (modeling the processes) and delivers the messages later by respecting the timing constraints. The same applies to Timed I/O automata [24]. Both solutions maintain much more details than required for automated verification of distributed algorithms such as [35]: First, processes do not compare process identifiers when making transitions, and thus are symmetric. Second, processes do not compare identifiers in the received messages, but only count messages.

For automated verification purposes, it appears natural to model such algorithms with timed automata that use a message-counting abstraction. However, the central question for practical verification is: *how precise is the message-counting abstraction?* In other words, given an algorithm A, what is the strongest equivalence between the model $M_S(A)$ using message sets and the model $M_C(A)$ using message counting. If the message counting abstraction is too coarse, then this may lead to spurious counterexamples, which may result in many refinement steps [17], or even may make the verification procedure incomplete.

[3] As we deal with distributed algorithms and timed automata, the notion of a *clock* appears in two different contexts in this paper, which should not be confused: The problem of clock synchronization is to compute adjustment for the hardware clocks (oscillators). In the context of timed automata, clocks are special variables used to model the timing behavior of a system.

Contributions. We introduce timed and untimed models suitable for the verification of threshold-based distributed algorithms, and establish relations between these models. An overview of the following contributions is depicted in Figure 2:

- We define a model of processes that count messages. We then compose them into asynchronous systems (interleaving semantics). We give two variants: message passing, where the messages are stored in sets, and message counting, where only the number of sent messages is stored in shared variables.
- We then show that in the asynchronous case, the message passing and the message counting variants are bisimilar. This proves the intuition that underlies the verification results from [23, 22, 27, 25]. It explains why no spurious counterexamples due to message-counting abstraction were experienced in the experimental evaluation of the verification techniques from [22].
- We obtain timed models by adding timing constraints on message delays that restrict the occurrence times of reception of messages depending on the sending times.
- We prove the surprising result that, in general, there is neither timed bisimulation nor time-abstracting bisimulation between the message passing and the message counting variants.
- Finally, we prove that there is timed simulation equivalence between the message passing and the message counting variants. This paves a way for abstraction-based model checking of timed fault-tolerant distributed algorithms such as [35].

In the following section, we briefly recall the classic definitions of transition systems, timed automata, and simulations [7, 16]. However, the timed automata defined there do not provide standard means to express processes that communicate via asynchronous message passing, as required for distributed algorithms. As we are interested in timed automata that capture this structure, we first define asynchronous message passing in Section 3 and then add timing constraints in Section 4 via message sets and message counting.

2 Preliminaries

We recall the classic definitions to the extent necessary for our work, and add two non-standard notions: First, our definition of a timed automaton assumes partitioning of the set of clocks into two disjoint sets: the message clocks (used to express the timing constraints of the message system underlying the distributed algorithm) and the specification clocks (used to express the specifications). Second, we assume that clocks are "not ticking" before they are started (more precisely, they are initialized to $-\infty$).

We will use the following sets: the set of Boolean values $\mathbb{B} = \{\mathsf{F}, \mathsf{T}\}$, the set of natural numbers $\mathbb{N} = \{1, 2, \dots\}$, the set $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, the set of non-negative reals $\mathbb{R}_{\geq 0}$, and the set of time instants $\mathbb{T} := \mathbb{R}_{\geq 0} \cup \{-\infty\}$.

Transition systems. Given a finite set AP of atomic propositions, a *transition system* is a tuple $TS = (S, S^0, R, L)$ where S is a set of states, $S^0 \subseteq S$ is a set of

initial states, $R \subseteq S \times S$ is a transition relation, and $L : S \to 2^{\mathsf{AP}}$ is a labeling function.

Clocks. A clock is a variable that ranges over the set $\mathbb{T}$. We call a clock that has the value $-\infty$ *uninitialized*. For a set X of clocks, a *clock valuation* is a function $\nu : X \to \mathbb{T}$. Given a clock valuation ν and a $\delta \in \mathbb{R}_{\geq 0}$, we define $\nu + \delta$ to be the valuation ν' such that $\nu'(c) = \nu(c) + \delta$ for $c \in X$ (Note that $-\infty + \delta = -\infty$). For a set $Y \subseteq X$ and a clock valuation $\nu : X \to \mathbb{T}$, we denote by $\nu[Y := 0]$ the valuation ν' such that $\nu'(c) = 0$ for $c \in Y \cap X$ and $\nu'(c) = \nu(c)$ for $c \in X \setminus Y$. Given a set of clocks Z, the set of *clock constraints* $\Psi(Z)$ is defined to contain all expressions generated by the following grammar:

$$\zeta := c \leq a \mid c \geq a \mid c < a \mid c > a \mid \zeta \wedge \zeta \qquad \text{for } c \in Z, a \in \mathbb{N}_0$$

Timed automata. Given a set of atomic propositions AP and a finite transition system (S, S^0, R, L) over AP, which models discrete control of a system, we model the system's real-time behavior with a *timed automaton*, i.e., a tuple $TA = (S, S^0, R, L, X \cup U, I, E)$ with the following properties:

- The set $X \cup U$ is the disjoint union of the sets of *message clocks* X and *specification clocks* U.
- The function $I : S \to \Psi(X \cup U)$ is a *state invariant*, which assigns to each discrete state a clock constraint over $X \cup U$, which must hold in that state. We denote by $\mu, \nu \models I(s)$ that the clock valuations μ and ν satisfy the constraints of $I(s)$.
- $E : R \to \Psi(X \cup U) \times 2^{(X \cup U)}$ is a *state switch relation* that assigns to each transition a guard on clock values and a (possibly empty) set of clocks that must be reset to zero, when the transition takes place.

We assume that AP is disjoint from $\Psi(X \cup U)$. Thus, the discrete behavior does not interfere with propositions on time. The semantics of a timed automaton $TA = (S, S^0, R, L, X \cup U, I, E)$ is an infinite transition system $TS(TA) = (Q, Q^0, \Delta, \lambda)$ over propositions $\mathsf{AP} \cup \Psi(U)$ with the following properties [6]:

1. The set Q of states consists of triples (s, μ, ν), where $s \in S$ is the discrete component of the state, whereas $\mu : X \to \mathbb{T}$ and $\nu : U \to \mathbb{T}$ are valuations of the message and specification clocks respectively such that $\mu, \nu \models I(s)$.
2. The set $Q^0 \subseteq Q$ of initial states comprises triples (s_0, μ_0, ν_0) with $s_0 \in S_0$, and clocks are set to $-\infty$, i.e., $\forall c \in X.\ \mu_0(c) = -\infty$ and $\forall c \in U.\ \nu_0(c) = -\infty$.
3. The transition relation Δ contains pairs $((s, \mu, \nu), (s', \mu', \nu'))$ of two kinds of transitions:
   (a) A time step: $s' = s$ and $\mu' = \mu + \delta$, $\nu' = \nu + \delta$, for $\delta > 0$, provided that for all $\delta' :\ 0 \leq \delta' \leq \delta$ the invariant is preserved, i.e., $\mu + \delta', \nu + \delta' \models I(s)$.
   (b) A discrete step: there is a transition $(s, s') \in R$ with $(\varphi, Y) = E((s, s'))$ whose guard φ is enabled, i.e., $\mu, \nu \models \varphi$, and the clocks from Y are reset, i.e., $\mu' = \mu[Y \cap X := 0]$, $\nu' = \nu[Y \cap U := 0]$, provided that $\mu', \nu' \models I(s)$.

   Given a transition $(q, q') \in \Delta$, we write $q \xrightarrow{\delta}_{\Delta} q'$ for a time step with delay $\delta \in \mathbb{R}_{\geq 0}$, or $q \to_{\Delta} q'$ for a discrete step.

4. The labeling function $\lambda : Q \to 2^{\mathsf{AP} \cup \Psi(U)}$ is defined as follows. For any state $q = (s, \mu, \nu)$, the labeling $\lambda(q) = L(s) \cup \{\varphi \in \Psi(U) : \mu, \nu \models \varphi\}$.

Comparing system behaviors. For transition systems $TS_i = (S_i, S_i^0, R_i, L_i)$ for $i \in \{1, 2\}$, a relation $H \subseteq S_1 \times S_2$ is a *simulation*, if (i) for each $(s_1, s_2) \in H$ the labels coincide $L_1(s_1) = L_2(s_2)$, and (ii) for each transition $(s_1, t_1) \in R_1$, there is a transition $(s_2, t_2) \in R_2$ such that $(t_1, t_2) \in H$. If, in addition, the set $H^{-1} = \{(s_2, s_1) \colon (s_1, s_2) \in H\}$ is also a simulation, then H is called *bisimulation*.

Further, if TA_1 and TA_2 are timed automata with $TS(TA_i) = (Q_i, Q_i^0, \Delta_i, \lambda_i)$ for $i \in \{1, 2\}$, then a simulation $H \subseteq Q_1 \times Q_2$ is called *timed simulation* [29], and a bisimulation $B \subseteq Q_1 \times Q_2$ is called *timed bisimulation* [15].

For transition systems $TS_i = (S_i, S_i^0, R_i, L_i)$ for $i \in \{1, 2\}$, we say that a simulation $H \subseteq S_1 \times S_2$ is *initial*, if $\forall s \in S_1^0 \ , \ \exists t \in S_2^0 \ . \ (s, t) \in H$. A bisimulation $B \subseteq S_1 \times S_2$ is initial, if the simulations B and B^{-1} are initial. The same applies to timed (bi-)simulations. Then, for $i \in \{1, 2\}$, we recall the standard preorders and equivalences on a pair of transition systems $TS_i = (S_i, S_i^0, R_i, L_i)$, and on a pair of timed automata TA_i, where $TS(TA_i) = (Q_i, Q_i^0, \Delta_i, \lambda_i)$:

1. $TS_1 \approx TS_2$ (*bisimilar*), if there is an initial bisimulation $B \subseteq S_1 \times S_2$.
2. $TA_1 \preceq^t TA_2$ (TA_2 *time-simulates* TA_1), if there is an initial timed simulation $H \subseteq Q_1 \times Q_2$.
3. $TA_1 \approx^t TA_2$ (*time-bisimilar*), if there is an initial timed bisimulation $B \subseteq Q_1 \times Q_2$.
4. $TA_1 \simeq^t TA_2$ (*time-simulation equivalent*), if $TA_1 \preceq^t TA_2$ and $TA_2 \preceq^t TA_1$.

Timed bisimulation forces time steps to advance clocks by the same amount of time. A coarser relation — called time-abstracting bisimulation [37] — allows two transition systems to advance clocks at "different speeds". Given two timed automata TA_i, for $i \in \{1, 2\}$ and the respective transition systems $TS(TA_i) = (Q_i, Q_i^0, \Delta_i, \lambda_i)$, a binary relation $B \subseteq Q_1 \times Q_2$ is a *time-abstracting bisimulation* [37], if the following holds for every pair $(q_1, q_2) \in B$:

1. The labels coincide: $\lambda_1(q_1) = \lambda_2(q_2)$;
2. For $\{j, k\} = \{1, 2\}$, and each discrete step $q_j \to_{\Delta_j} r_j$, there is a discrete step $q_k \to_{\Delta_k} r_k$ and $(r_j, r_k) \in B$;
3. For $\{j, k\} = \{1, 2\}$, a delay $\delta \in \mathbb{R}_{\geq 0}$, and a time step $q_j \xrightarrow{\delta}_{\Delta_j} r_j$, there is a delay $\delta' \in \mathbb{R}_{\geq 0}$ and a time step $q_k \xrightarrow{\delta'}_{\Delta_k} r_k$ such that $(r_j, r_k) \in B$.

Note that if we substitute δ' with δ in Point 3, then we obtain the definition of timed bisimulation.

3 Asynchronous message passing systems

Timed automata as defined above neither capture processes nor communication via messages, as would be required to model distributed algorithms. Hence we now introduce these notions and then construct an asynchronous system using

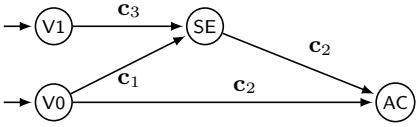

Fig. 3. A graphical representation of a process discussed in Example 3.1

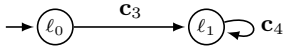

Fig. 4. A simple two-state process (used later in Theorem 5.1)

processes and message passing (or message counting). We assume that at every step a process receives and sends at most one message [19]. In Section 4, we add time to this modeling in order to obtain a timed automaton.

Single correct process. We assume a (possibly infinite) set of control states $\mathcal{L}$ and a subset $\mathcal{L}_0 \subseteq \mathcal{L}$ of initial control states. We fix a finite set MT of message types. We assume that the control states in $\mathcal{L}$ keep track of the messages sent by a process. Thus, $\mathcal{L}$ comes with a predicate $\mathsf{is\_sent}\colon \mathcal{L} \times \mathsf{MT} \to \mathbb{B}$, where $\mathsf{is\_sent}(\ell, m)$ evaluates to true if and only if a message of type m has been sent according to the control state ℓ. Finally, we introduce a set Π of parameters and store the parameter values in a vector $\mathbf{p} \in {\mathbb{N}_0}^{|\Pi|}$. As noted in [22], parameter values are typically restricted with a resilience condition such as $n > 3t$ (less than a third of the processes are faulty), so we will assume that there is a set of all admissible combinations of parameter values $\mathbf{P}_{RC} \subseteq {\mathbb{N}_0}^{|\Pi|}$.

The behavior of a single process is defined as a *process transition relation* $\mathcal{T} \subseteq \mathcal{L} \times {\mathbb{N}_0}^{|\Pi|} \times {\mathbb{N}_0}^{|\mathsf{MT}|} \times \mathcal{L}$ encoding transitions guarded by conditions on message counters that range over ${\mathbb{N}_0}^{|\mathsf{MT}|}$: when $(\ell, \mathbf{p}, \mathbf{c}, \ell') \in \mathcal{T}$, a process can make a transition from the control state ℓ to the control state ℓ', provided that, for every $m \in \mathsf{MT}$, the number of received messages of type m reaches the value $\mathbf{c}(m)$ in a configuration with parameter values $\mathbf{p}$.

Example 3.1. The process shown in Figure 1 can be written in our definitions as follows. The algorithm is using only one message type, and thus $\mathsf{MT} = \{\mathsf{ECHO}\}$. We assume a set of control states $\mathcal{L} = \{\mathsf{V0}, \mathsf{V1}, \mathsf{SE}, \mathsf{AC}\}$: $\mathsf{V0}$ and $\mathsf{V1}$ encode the initial states where $myval = 0$ and $myval = 1$ respectively, $\mathsf{pc} = \mathsf{SE}$ encodes the status "ECHO sent before", and $\mathsf{pc} = \mathsf{AC}$ encode the status "accept". The initial control states are: $\mathcal{L}_0 = \{\mathsf{V0}, \mathsf{V1}\}$. The transition relation contains four types of transitions: $t_1^{\mathbf{p}} = (\mathsf{V0}, \mathbf{p}, \mathbf{c}_1, \mathsf{SE})$, $t_2^{\mathbf{p}} = (\mathsf{V0}, \mathbf{p}, \mathbf{c}_2, \mathsf{AC})$, $t_3^{\mathbf{p}} = (\mathsf{V1}, \mathbf{p}, \mathbf{c}_3, \mathsf{SE})$, and $t_4^{\mathbf{p}} = (\mathsf{SE}, \mathbf{p}, \mathbf{c}_2, \mathsf{AC})$, for any $\mathbf{p} \in {\mathbb{N}_0}^{|\Pi|}$ and $\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3$ satisfying the following: $\mathbf{c}_1(\mathsf{ECHO}) \geq \mathbf{p}(t) + 1$, $\mathbf{c}_2(\mathsf{ECHO}) \geq \mathbf{p}(n) - \mathbf{p}(t)$, and $\mathbf{c}_3(\mathsf{ECHO}) \geq 0$. Finally, $\mathsf{is\_sent}(\ell, \mathsf{ECHO})$ iff $\ell \in \{\mathsf{SE}, \mathsf{AC}\}$. A concise graphical representation of the transition relation is given in Figure 3. There, each edge represents multiple transitions of the same type. Let us observe that while the action of sending a message can be inferred by simply checking all the transitions going from a state s to a state t such that $\neg\mathsf{is\_sent}(s)$ and $\mathsf{is\_sent}(t)$, the action of receiving an individual message is not part of the process description at this level. However, if a guarded transition is taken, this implies that a threshold has been reached, e.g., in case of $\mathbf{c}_1$, at least $t + 1$ messages were received. ◁

We make two assumptions typical for distributed algorithms [19, 35]:

A1 Processes do not forget that they have sent messages: If $(\ell, \mathbf{p}, \mathbf{c}, \ell') \in \mathcal{T}$, then $\mathsf{is\_sent}(\ell, m) \to \mathsf{is\_sent}(\ell', m)$ for every $m \in \mathsf{MT}$.

A2 At each step a process sends at most one message to all: If $(\ell, \mathbf{p}, \mathbf{c}, \ell') \in \mathcal{T}$ and $\neg\mathsf{is\_sent}(\ell, m) \wedge \mathsf{is\_sent}(\ell', m) \wedge \neg\mathsf{is\_sent}(\ell, m') \wedge \mathsf{is\_sent}(\ell', m')$ then $m = m'$.

Then, we call $(\mathsf{MT}, \mathcal{L}, \mathcal{L}_0, \mathcal{T})$ a process template.

Asynchronous message passing and counting in presence of Byzantine faults. In this section we introduce two ways of modeling message passing: by storing messages in sets, and by counting messages. As in [23], we do not explicitly model Byzantine processes [32], but capture their effect on the correct processes in the form of spurious messages. Although we do not discuss other kinds of faults (e.g., crashes, symmetric faults, omission faults), it is not hard to model other faults by following the modeling in [23].

We fix a set of processes Proc, which is typically defined as $\{1, \ldots, n\}$ for $n \geq 1$. Further, assume that there are two disjoint sets: the set $\mathsf{Corr} \subseteq \mathsf{Proc}$ of correct processes, and $\mathsf{Byz} \subseteq \mathsf{Proc}$ of Byzantine processes (possibly empty), with $\mathsf{Byz} \cup \mathsf{Corr} = \mathsf{Proc}$. Given a process template $(\mathsf{MT}, \mathcal{L}, \mathcal{L}_0, \mathcal{T})$, we refer to $(\mathsf{MT}, \mathcal{L}, \mathcal{L}_0, \mathcal{T}, \mathsf{Corr}, \mathsf{Byz})$ as a *design*. Note that a design does not capture how processes interact with messages. To do so, in Table 1, we define message passing and message counting models as interpretations of the signature $(\mathit{Msg}, \mathit{MsgSets}, \mathit{init}, \mathit{card}, \mathit{add}, \mathit{inTransit})$, with the following informal meaning:

- *Msg*: the set of all messages that can be exchanged by the processes,
- *MsgSets*: collections of messages,
- *init*: the empty collection of messages,
- *card*: a function that counts messages of the given type,
- *add*: a function that adds a message to a collection of messages
- *inTransit*: a function that checks whether a message is in transit and thus can be received.

Transition systems. Let us assume an interpretation $(\mathit{Msg}_I, \mathit{MsgSets}_I, \mathit{init}_I, \mathit{card}_I, \mathit{add}_I, \mathit{inTransit}_I)$ for $I \in \{MP, MC\}$. We define a transition system $TS^I = (S^I, S_0^I, R^I, L^I)$ of processes from Proc that communicate with respect to I. We call *message-passing system* the transition system obtained using the interpretation MP, and *message-counting system* the transition system obtained using the interpretation MC.

The set S^I contains configurations, i.e., tuples $(\mathbf{p}, \mathsf{pc}, \mathsf{rcvd}, \mathsf{sent})$ having the following properties: (a) $\mathbf{p} \in {\mathbb{N}_0}^{|\Pi|}$, (b) $\mathsf{pc} : \mathsf{Corr} \to \mathcal{L}$, (c) $\mathsf{rcvd} : \mathsf{Corr} \to \mathit{MsgSets}_I$, and (d) $\mathsf{sent} \in \mathit{MsgSets}_I$. In a configuration, for every process $p \in \mathsf{Corr}$, the values $\mathsf{pc}(p)$ and $\mathsf{rcvd}(p)$ comprise the *local view* of the process p, while the components sent and $\mathbf{p}$ comprise the *shared state* of the distributed system. A configuration $\sigma \in S^I$ belongs to the set S_0^I of initial configurations, if for each process $p \in \mathsf{Corr}$, it holds that: (a) $\sigma.\mathsf{pc}(p) \in \mathcal{L}_0$, (b) $\sigma.\mathsf{rcvd}(p) = \mathit{init}_I$, (c) $\sigma.\mathsf{sent} = \mathit{init}_I$, and (d) $\sigma.\mathbf{p} \in \mathbf{P}_{RC}$.

Message passing (MP)	**Message counting (MC):**
$Msg_{\mathsf{MP}} \triangleq \mathsf{MT} \times \mathsf{Proc}$	$Msg_{\mathsf{MC}} \triangleq \mathsf{MT} \times \{\mathsf{C}, \mathsf{F}\}$
$MsgSets_{\mathsf{MP}} \triangleq 2^{\mathsf{MT} \times \mathsf{Proc}}$	$MsgSets_{\mathsf{MC}} \triangleq \{0, \ldots, \lvert\mathsf{Corr}\rvert\}^{\lvert\mathsf{MT}\rvert} \times \{0, \ldots, \lvert\mathsf{Byz}\rvert\}^{\lvert\mathsf{MT}\rvert}$
Initial messages, $init \in MsgSets$	
$init_{\mathsf{MP}} \triangleq \emptyset$	$init_{\mathsf{MC}} \triangleq ((0, \ldots, 0), (0, \ldots, 0))$
Count messages, $card : \mathsf{MT} \times MsgSets \to \mathbb{N}_0$	
$card_{\mathsf{MP}}(m, M) \triangleq \lvert\{p \in \mathsf{Proc} \colon (m, p) \in M\}\rvert$	$card_{\mathsf{MC}}(m, (\mathbf{c}_\mathsf{C}, \mathbf{c}_\mathsf{F})) \triangleq \mathbf{c}_\mathsf{C}(m) + \mathbf{c}_\mathsf{F}(m)$
Add a message, $add : Msg \times MsgSets \to MsgSets$	
$add_{\mathsf{MP}}(\langle m, p\rangle, M) \triangleq M \cup \{\langle m, p\rangle\}$	$add_{\mathsf{MC}}((m, tag), (\mathbf{c}_\mathsf{C}, \mathbf{c}_\mathsf{F})) \triangleq (\mathbf{c}'_\mathsf{C}, \mathbf{c}'_\mathsf{F})$ such that $\mathbf{c}'_\mathsf{C}(m) = \mathbf{c}_\mathsf{C}(m) + 1$ and $\mathbf{c}'_\mathsf{F}(m) = \mathbf{c}_\mathsf{F}(m)$, if $tag = \mathsf{C}$ $\mathbf{c}'_\mathsf{F}(m) = \mathbf{c}_\mathsf{F}(m) + 1$ and $\mathbf{c}'_\mathsf{C}(m) = \mathbf{c}_\mathsf{C}(m)$, if $tag = \mathsf{F}$ and $\mathbf{c}'(m') = \mathbf{c}(m)$ for $m' \in \mathsf{MT}, m' \neq m$
Is there a message to deliver? $inTransit : Msg \times MsgSets \times MsgSets \to \mathbb{B}$	
$inTransit_{\mathsf{MP}}(\langle m, p\rangle, M, M') \triangleq$ $(p \in \mathsf{Corr} \wedge \langle m, p\rangle \in M' \setminus M) \vee (p \in \mathsf{Byz} \wedge \langle m, p\rangle \notin M)$	$inTransit_{\mathsf{MC}}((m, tag), (\mathbf{c}_\mathsf{C}, \mathbf{c}_\mathsf{F}), (\mathbf{c}'_\mathsf{C}, \mathbf{c}'_\mathsf{F})) \triangleq$ $(tag = \mathsf{C} \wedge \mathbf{c}'_\mathsf{C} > \mathbf{c}_\mathsf{C}) \vee (tag = \mathsf{F} \wedge \mathbf{c}_\mathsf{F} < \lvert\mathsf{Byz}\rvert)$

Table 1. The message-passing and message-counting interpretations

Definition 3.2. *The transition relation R^I contains a pair of configurations $(\sigma, \sigma') \in S^I \times S^I$, if there is a correct process $p \in \mathsf{Corr}$ that satisfies:*

1. *There is a local transition $(\ell, \mathbf{p}, \mathbf{c}, \ell') \in \mathcal{T}$ satisfying $\sigma.\mathsf{pc}(p) = \ell$ and $\sigma'.\mathsf{pc}(p) = \ell'$ and for all m in MT, $\mathbf{c}(m) = card_I(m, \sigma'.\mathsf{rcvd}(p))$. Also, it is required that $\sigma.\mathbf{p} = \sigma'.\mathbf{p} = \mathbf{p}$.*
2. *Messages are received and sent according to the signature:*
   (a) *Process p receives no message: $\sigma'.\mathsf{rcvd}(p) = \sigma.\mathsf{rcvd}(p)$, or there is a message in transit in σ that is received in σ', i.e., there is a message $msg \in Msg_I$ satisfying:*
   $inTransit_I(msg, \sigma.\mathsf{rcvd}(p), \sigma.\mathsf{sent}) \wedge \sigma'.\mathsf{rcvd}(p) = add_I(msg, \sigma.\mathsf{rcvd}(p))$.
   (b) *The shared variable sent is changed iff process p sends a message, that is, $\sigma'.\mathsf{sent} = add_I(msg, \sigma.\mathsf{sent})$, if and only if $\neg\mathsf{is\_sent}(\sigma.\mathsf{pc}(p), m)$ and $\mathsf{is\_sent}(\sigma'.\mathsf{pc}(p), m)$, for every $m \in \mathsf{MT}$ and $msg \in Msg_I$ of type m.*
3. *The processes different from p do not change their local states:*
   $\sigma'.\mathsf{pc}(q) = \sigma.\mathsf{pc}(q)$ *and* $\sigma'.\mathsf{rcvd}(q) = \sigma.\mathsf{rcvd}(q)$ *for* $q \in \mathsf{Corr} \setminus \{p\}$.

The labeling function $L^I : S^I \to \mathcal{L}^{|\mathsf{Corr}|} \times \left(\mathbb{N}_0^{|\mathsf{MT}|}\right)^{|\mathsf{Corr}|}$ labels each configuration $\sigma \in S^I$ with the vector of control states and message counters, i.e., $L^I(\sigma) = ((\ell_1, \ldots, \ell_{|\mathsf{Corr}|}), (\mathbf{c}_1, \ldots, \mathbf{c}_{|\mathsf{Corr}|}))$ such that $\ell_p = \sigma.\mathsf{pc}(p)$ and $\mathbf{c}_p(m) = card_I(m, \sigma.\mathsf{rcvd}(p))$ for $p \in \mathsf{Corr}$, $m \in \mathsf{MT}$. (For simplicity we use the convention that $\mathsf{Corr} = \{1, \ldots j\}$, for some $j \in \mathbb{N}_0$.) Note that L^I labels a configuration with the process control states and the number of messages received by each process.

The message-passing transition systems have the following features. The messages sent by *correct* processes are stored in the shared set sent. In this modeling, the messages from Byzantine processes are not stored in sent explicitly, but can be received at any step. Each correct process $p \in \mathsf{Corr}$ stores received messages in its local set $\mathsf{rcvd}(p)$, whose elements originate from the messages stored in the set sent or from Byzantine processes.

The message-counting transition systems have the following features. Messages are not stored explicitly, but are only counted. We maintain two vectors of counters: (i) representing the number of messages that originate from correct processes (these messages have the tag C), and (ii) representing the number of messages that originate from faulty processes (these messages have the tag F). Each correct process $p \in \mathsf{Corr}$ keeps two such vectors of counters $\mathbf{c}_\mathsf{C}$ and $\mathbf{c}_\mathsf{F}$ in its local variable $\mathsf{rcvd}(p)$. In the following, we refer to $\mathbf{c}_\mathsf{C}$ and $\mathbf{c}_\mathsf{F}$ using the notation $[\mathsf{rcvd}(p)]_\mathsf{C}$ and $[\mathsf{rcvd}(p)]_\mathsf{F}$. The number of sent messages is also stored as a pair of vectors $[\mathsf{sent}]_\mathsf{C}$ and $[\mathsf{sent}]_\mathsf{F}$. By the definition of the transition relation R^{MC}, the vector $[\mathsf{sent}]_\mathsf{F}$ is always equal to the zero vector, whereas the correct process p can increment its counter $[\mathsf{rcvd}(p)]_\mathsf{F}$, if $[\mathsf{rcvd}(p)]_\mathsf{F}(m) < |\mathsf{Byz}|$, for every $m \in \mathsf{MT}$.

To prove bisimulation between a message-passing system and a message-counting system — built from the same design — we introduce the following relation on the configurations both systems:

Definition 3.3. *Let $H^\# \subseteq S^{MP} \times S^{MC}$ such that $(\sigma, \sigma^\#) \in H^\#$ if for all processes $p \in \mathsf{Corr}$ and message types $m \in \mathsf{MT}$:*

1. $\sigma^\#.\mathsf{pc}(p) = \sigma.\mathsf{pc}(p)$
2. $\sigma^\#.[\mathsf{rcvd}(p)]_\mathsf{C}(m) = |\{q \in \mathsf{Corr} \colon \langle m, q\rangle \in \sigma.\mathsf{rcvd}(p)\}|$
3. $\sigma^\#.[\mathsf{rcvd}(p)]_\mathsf{F}(m) = |\{q \in \mathsf{Byz} \colon \langle m, q\rangle \in \sigma.\mathsf{rcvd}(p)\}|$
4. $\sigma^\#.[\mathsf{sent}]_\mathsf{C}(m) = |\{q \in \mathsf{Corr} \colon \langle m, q\rangle \in \sigma.\mathsf{sent}\}|$
5. $\sigma^\#.[\mathsf{sent}]_\mathsf{F}(m) = 0$
6. $\{q \in \mathsf{Proc} \colon \langle m, q\rangle \in \sigma.\mathsf{sent}\} \subseteq \mathsf{Corr}$
7. $\sigma.\mathsf{rcvd}(p) \subseteq \sigma.\mathsf{sent} \cup \{\langle m, q\rangle : m \in \mathsf{MT}, q \in \mathsf{Byz}\}$
8. $\mathsf{is\_sent}(\sigma.\mathsf{pc}(p), m) \leftrightarrow \langle m, p\rangle \in \sigma.\mathsf{sent}$

Theorem 3.4. *For a message-passing system TS^{MP} and a message-counting system TS^{MC} defined over the same design, $H^\#$ is a bisimulation.*

The key argument to prove the Theorem 3.4 is that given a message counting state $\sigma^\#$, if a step increases a counter $\mathsf{rcvd}(p)$, in the message passing system this transition can be mirrored by receiving an arbitrary message in transit. In fact, in both systems, once a message is sent it can be received at any future step. We will see that in the timed version this argument does not work anymore, due to the restricted time interval in which a message must be received.

4 Messages with time constraints

We now add time constraints to both, message-passing systems and message-counting systems. Following the definitions from distributed algorithms [35, 40], we assume that every message is delivered within a predefined time bound, that is, not earlier than τ^- time units and not later than τ^+ times units since the instant it was sent, with $0 \leq \tau^- \leq \tau^+$. We use naturals for τ^- and τ^+ for consistency with the literature on timed automata.

As can be seen from Section 2, to define a timed automaton, one has to provide an invariant and a switch relation. In the following, we fix the invariants and switch relations with respect to the timing constraints τ^- and τ^+ on messages. However, as the specification of distributed algorithms may refer to time (e.g., "If a correct process accepts the message *(round k)* at time t, then every correct process does so by time $t + t_{\text{del}}$" [35]), we assume that a *specification invariant* (or *user invariant*) $I_U : 2^{\mathsf{AP}} \to \Psi(U)$ and a *specification switch relation* (or *user switch relation*) $E_U : 2^{\mathsf{AP}} \times 2^{\mathsf{AP}} \to \Psi(U) \times 2^U$ are given as input. Then, we will refer to the tuple $(\mathcal{L}, \mathcal{L}_0, \mathcal{T}, \mathsf{Proc}, I_U, E_U)$ as a *timed design* and we will assume that a timed design is fixed in the following.

Using a timed design, we will use message-passing and message-counting systems to derive two timed automata. For a message of type m sent by a correct process p, the former uses a clock $c\langle m, p\rangle$ to store *the delay since the message* $\langle m, p\rangle$ *was sent*. Instead, message-counting stores *the delay since the* ith *message of type* m *was sent*, for all i and m. Both timed automata specify an invariant to constrain the time required to deliver a message.

Definition 4.1 (Message-passing timed automaton). *Given a message-passing system* $TS^{MP} = (S^{MP}, S_0^{MP}, R^{MP}, L^{MP})$ *defined over a timed system design* $(\mathcal{L}, \mathcal{L}_0, \mathcal{T}, \mathsf{Proc}, I_U, E_U)$*, we say that a timed automaton* $TA^{MP} = (S^{MP}, S_0^{MP}, R^{MP}, L^{MP}, U \cup X^{MP}, I^{MP}, E^{MP})$ *is a* message-passing timed automaton*, if it has the following properties:*

1. *There is one clock per message that can be sent by a correct process:* $X^{MP} = \{c\langle m, p\rangle : m \in \mathsf{MT}, p \in \mathsf{Corr}\}$.
2. *For each discrete transition* $(\sigma, \sigma') \in R^{MP}$*, the state switch relation* $E^{MP}(\sigma, \sigma')$ *ensures the specification invariant and resets the given specification clocks and the clocks corresponding to the message sent in transition* (σ, σ')*. That is, if* (φ_U, Y_U) *is the guard, and specification clocks are in* $E_U(L^{MP}(\sigma), L^{MP}(\sigma'))$*, then* $E^{MP}(\sigma, \sigma') = (\varphi_U, Y_U \cup \{c\langle m, p\rangle : \langle m, p\rangle \in \sigma'.\mathsf{sent} \setminus \sigma.\mathsf{sent}\})$.
3. *Each state* $\sigma \in S^{MP}$ *has the invariant* $I^{MP}(\sigma) = I_U(L^{MP}(\sigma)) \wedge \varphi^-_{MP} \wedge \varphi^+_{MP}$ *composed of:*
   (a) *the specification invariant* $I_U(L^{MP}(\sigma))$*;*
   (b) *the lower bound on the age of received messages:* $\varphi^-_{MP} = \bigwedge_{\langle m,p\rangle \in M} c\langle m, p\rangle \geq \tau^-$ *for* $M = \{\langle m, p\rangle \in \mathsf{MT} \times \mathsf{Corr} \colon \exists q \in \mathsf{Corr}.\ \langle m, p\rangle \in \sigma.\mathsf{rcvd}(q)\}$*; and*
   (c) *the upper bound on the age of messages that are in transit:* $\varphi^+_{MP} = \bigwedge_{(m,p)\in M} 0 \leq c\langle m, p\rangle \leq \tau^+$ *for* $M = \{\langle m, p\rangle \in \mathsf{MT} \times \mathsf{Corr} \colon \langle m, p\rangle \in \sigma.\mathsf{sent} \setminus \bigcap_{q\in\mathsf{Corr}} \sigma.\mathsf{rcvd}(q)\}$.

Definition 4.2 (Message-counting timed automaton). *Given a message-counting system* $TS_{MC} = (S^{MC}, S_0^{MC}, R^{MC}, L^{MC})$ *defined over a timed design* $(\mathcal{L}, \mathcal{L}_0, \mathcal{T}, \mathsf{Proc}, I_U, E_U)$*, we say that a timed automaton* $TA_{MC} = (S^{MC}, S_0^{MC}, R^{MC}, L^{MC}, U \cup X^{MC}, I^{MC}, E^{MC})$ *is a* message-counting timed automaton*, if it has the following properties:*

1. *There is one clock per message type and number of messages sent. That is,* $X^{MC} = \{c\langle m, i\rangle : m \in \mathsf{MT}, 1 \leq i \leq |\mathsf{Corr}|\}$.

2. *For each discrete transition* $(\sigma, \sigma') \in R^{MC}$*, the state switch relation* $E^{MC}(\sigma, \sigma')$ *ensures the specification invariant and resets the given specification clocks and the clocks corresponding to message counters updated by* (σ, σ')*. That is, if* $(\varphi_U, Y_U) = E_U(L^{MC}(\sigma), L^{MC}(\sigma'))$*, then the switch relation* $E^{MC}(\sigma, \sigma')$ *is* $(\varphi_U, Y_U \cup \{c\,\langle m, k\rangle : m \in \mathsf{MT}, k = \sigma'.\mathsf{sent}(m) = \sigma.\mathsf{sent}(m) + 1\})$.
3. *Each state* $\sigma \in S^{MC}$ *has the invariant* $I^{MC}(\sigma) = I_U(L^{MC}(\sigma)) \wedge \varphi^-_{MC} \wedge \varphi^+_{MC}$ *composed of:*
   (a) *the specification invariant* $I_U(L^{MC}(\sigma))$*;*
   (b) $\varphi^-_{MC} = \bigwedge_{m \in \mathsf{MT}} a(m) > 0 \to c\,\langle m, a(m)\rangle \geq \tau^-$ *for the numbers* $a(m) = \max_{p \in \mathsf{Corr}} [\sigma.\mathsf{rcvd}(p)(m)]_{\mathsf{C}}$*. If a correct process has received* $a(m)$ *messages of type* m *from correct processes, then the* $a(m)$*-th message of type* m*, for every* $m \in \mathsf{MT}$*, was sent at least* τ^- *time units earlier.*
   (c) $\varphi^+_{MC} = \bigwedge_{m \in \mathsf{MT}} \bigwedge_{b(m) < j \leq \sigma.\mathsf{sent}(m)} 0 \leq c\,\langle m, j\rangle \leq \tau^+$ *for the numbers* $b(m) = \min_{p \in \mathsf{Corr}} [\sigma.\mathsf{rcvd}(p)(m)]_{\mathsf{C}}$*. If there is a correct process that has received* $b(m)$ *messages of type* m *from correct processes, then for every number of messages* $j > b(m)$*, the respective clock is bounded by* τ^+*.*

While the number of employed clocks is the same, the latter model is "more abstract": by forgetting the identity of the sender, indeed, several configurations of the message-passing timed automaton can be mapped on the same configuration of the message-counting timed automaton.

5 Precision of Message Counting with Time Constraints

While Theorem 3.4 establishes a strong equivalence — that is, a bisimulation relation — between message-passing transition systems, we will show in Theorem 5.1 that message-passing timed automata and message-counting timed automata are not necessarily equivalent in the sense of timed bisimulation. Remarkably, such automata are also not necessarily equivalent in the sense of time-abstracting bisimulation. These results show an upper bound on the degree of precision achievable by model checking of timed properties of FTDAs by counting messages. Nevertheless, we show that such automata simulate each other, and thus they satisfy the same ATCTL formulas (Corollary 5.10 and Corollary 6.2).

Theorem 5.1. *There exsits a timed design whose message-passing timed automaton* TA^{MP} *and message-counting timed automaton* TA^{MC} *satisfy:*

1. *There is no initial timed bisimulation between* TA^{MP} *and* TA^{MC}*.*
2. *There is no initial time-abstracting bisimulation between* TA^{MP} *and* TA^{MC}*.*

Proof (sketch). We give an example of a timed design proving Point 2. Since timed bisimulation is a special case of time-abstracting bisimulation, this example also proves Point 1.

We use the process template shown in Figure 4 on page 7. Formally, this template is defined as follows: there is one parameter, i.e., $\Pi = \{n\}$, one message type, i.e., $\mathsf{MT} = \{M\}$, and two control states, i.e., $\mathcal{L} = \{\ell_0, \ell_1\}$. There are two

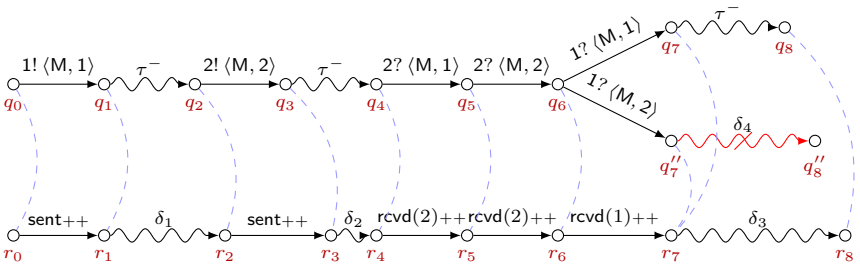

Fig. 5. Two runs of TA^{MP} (above) and one run of TA^{MC} (below) that violate time-abstracting bisimulation when $\tau^+ = 2\tau^-$. Circles and edges illustrate states and transitions. Edge labels are as follows: τ^- or δ_i designate a time step with the respective delay; $\mathsf{i}!\,\langle \mathsf{M}, \mathsf{j}\rangle$ and $\mathsf{i}?\,\langle \mathsf{M}, \mathsf{j}\rangle$ designate send and receive of a message $\langle \mathsf{M}, \mathsf{j}\rangle$ by process i in the message-passing system; sent_{++} and $\mathsf{rcvd}(\mathsf{i})_{++}$ designate send and receive of a message M by some process and process i respectively.

types of transitions: $t_1^{\mathbf{P}} = (\ell_0, \mathbf{p}, \mathbf{c}_3, \ell_1)$ and $t_2^{\mathbf{P}} = (\ell_1, \mathbf{p}, \mathbf{c}_4, \ell_1)$. The conditions $\mathbf{c}_3$ and $\mathbf{c}_4$ require that $\mathbf{c}_3(M) = 0$ and $\mathbf{c}_4(M) \geq 0$ respectively. Every process sends a message of type M when going from ℓ_0 to ℓ_1, i.e., $\mathsf{is\_sent}(\ell, M) = \mathsf{T}$ iff $\ell = \ell_1$. Then the processes self-loop in the control state ℓ_1 (by doing so, they can receive messages from the other processes).

Consider the system of two correct processes and no Byzantine processes, that is, $\mathsf{Corr} = \{1, 2\}$ and $\mathsf{Byz} = \emptyset$. We fix the upper bound on message delays to be $\tau^+ = 2\tau^- > 0$. For the sake of this proof, we set $U = \emptyset$, and thus I_U and E_U are defined trivially. Together, these constraints define a timed design.

Figure 5 illustrates two runs of a TA^{MP} and a run of TA^{MC} that should be matched by a time-abstracting bisimulation, if one exists. We show by contradiction that no such relation exists. Note that the message $\langle M, 1\rangle$ has been received by all processes at the timed state q_7 and has not been received by the first process at the timed state q_7''. Thus the timed state q_7 admits a time step, and the timed state q_7'' does not. Indeed, on one hand, the timed automaton TA^{MP} can advance the clocks by at most $\tau^+ - \tau^- = \tau^-$ time units in q_7 before the clock attached to the message $\langle M, 2\rangle$ expires; on the other hand, in q_7'', the timed automaton TA^{MP} cannot advance the clocks before the clock attached to the message $\langle M, 1\rangle$ expires. However, both states must be time-abstract related to the state r_7 of TA^{MC}, because they both received the same number of messages of type M and thus their labels coincide, from which we derive the required contradiction. Hence, proving that there is no time-abstracting bisimulation. □

From Theorem 5.1, it follows that message counting abstraction is not precise enough to preserve an equivalence relation as strong as bisimulation. However, for abstraction-based model checking a coarser relation, namely, timed-simulation equivalence, would be sufficient. In one direction, timed-simulation is easy: a discrete configuration of a message-passing timed automaton can be mapped to the configuration of the message-counting timed automaton by just counting

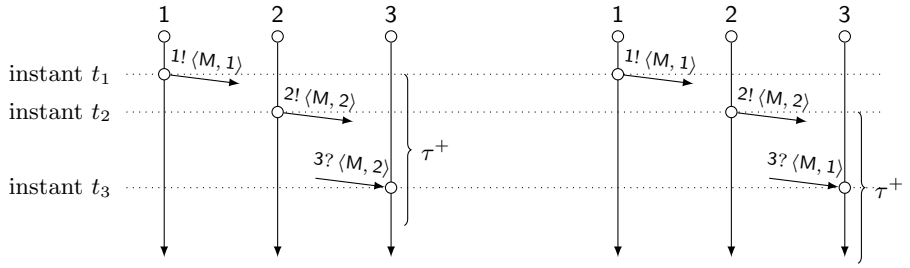

Fig. 6. Receiving messages in order relaxes constraints of delay transitions

the messages for each message type, while the clocks assignments are kept the same. The other direction is harder: A first approach would be to map a configuration of a message-counting timed automaton to all the configurations of the message-passing timed automaton, where the message counters are equal to the cardinalities of the sets of received messages. This mapping is problematic because of the interplay of message re-ordering and timing constraints:

Example 5.2. Figure 6 exemplifies a problematic behavior of a message passing system with three processes. We will see that the problem originates from the interplay of message re-ordering and timing constraints. Assume that in the message-passing system, process 3 receives messages out of order, that is, it enters a local state ℓ_m in which a message $\langle \mathsf{M}, 2\rangle$ is received and $\langle \mathsf{M}, 1\rangle$ is not received while the latter was sent at an earlier time than the former. Intuitively, as in ℓ_m message $\langle \mathsf{M}, 1\rangle$ is still in transit, and due to the invariants from Definition 4.1[3c] not too much time may pass before $\langle \mathsf{M}, 1\rangle$ is received, viz. at most $\tau^+ - (t_3 - t_1)$ time. Let $\ell_{m'}$ be the local state of 3 that is obtained by receiving $\langle \mathsf{M}, 1\rangle$ instead. Observe that the constraint on the delays of possible time steps after $\ell_{m'}$ is weaker than the one after ℓ_m, i.e at most $\tau^+ - (t_3 - t_2)$ which, by our assumptions, is greater than $\tau^+ - (t_3 - t_1)$. ◁

In the following, we exclude states where in-transit messages have been sent before a received one (e.g., ℓ_m from the above example) from the simulation relation, and only consider so-called well-formed states where the received messages strictly adhere to the *temporal* order of the sending. Indeed, we use the fact that the timing constraints of well-formed states in the message-passing system match the timing constraints in the message-counting system.

Definition 5.3 (Well-formed state). *For a message-passing timed automaton* TA^{MP} *with* $TS(TA^{MP}) = (Q, Q_0, \Delta, \lambda)$*, a state* $(s, \mu, \nu) \in Q$ *is* well-formed*, if for each message type* $m \in \mathsf{MT}$*, each process* $p \in \mathsf{Corr}$ *that has received a message* $\langle m, p'\rangle$ *has also received all messages of type* m *sent earlier than* $\langle m, p'\rangle$*:*

$$\langle m, p'\rangle \in s.\mathsf{rcvd}(p) \wedge \mu(c\,\langle m, p''\rangle) > \mu(c\,\langle m, p'\rangle) \\ \rightarrow \langle m, p''\rangle \in s.\mathsf{rcvd}(p) \text{ for } p', p'' \in \mathsf{Corr} \quad (1)$$

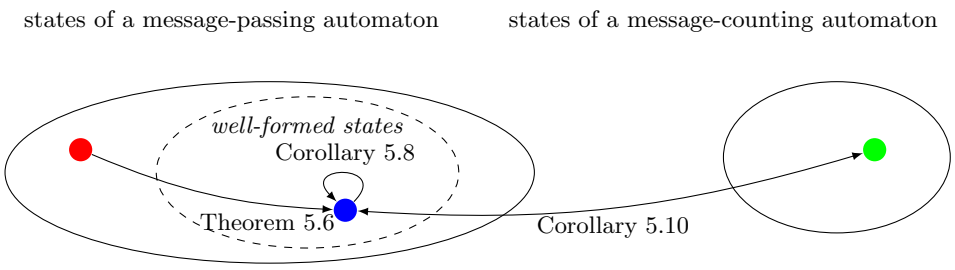

Fig. 7. Simulations constructed in Theorems 5.6–5.10. Small circles depict states of the transition systems. An arrow from a state s to a state t illustrates that the pair (s, t) belongs to a timed simulation

Observe that because messages can be sent at precisely the same time, there can be different well-formed states s and s' with $s.\mathsf{rcvd}(p) \neq s'.\mathsf{rcvd}(p)$. Also, considering only well-formed states does not imply that the messages are received according to the sending order in a run (which would correspond to FIFO).

We will use a mapping TO to abstract arbitrary states of any message passing timed automaton to sets of well-formed states in the same automaton.

Definition 5.4. *Given a message-passing timed automaton TA^{MP} with the transition system $TS(TA^{MP}) = (Q, Q_0, \Delta, \lambda)$, we define a mapping $\mathsf{TO} : Q \to 2^Q$ that maps an automaton state $(s, \mu, \nu) \in Q$ into a set of well-formed states with each $(s', \mu', \nu') \in \mathsf{TO}((s, \mu, \nu))$ having the following properties:*

1. *$\mu' = \mu$, $\nu' = \nu$, $s'.\mathsf{sent} = s.\mathsf{sent}$, and $s.\mathsf{pc}(p) = s'.\mathsf{pc}(p)$ for $p \in \mathsf{Corr}$, and*
2. *$|\{q\colon \langle m, q\rangle \in s'.\mathsf{rcvd}(p)\}| = |\{q\colon \langle m, q\rangle \in s.\mathsf{rcvd}(p)\}|$ for $m \in \mathsf{MT}, p \in \mathsf{Corr}$.*

We show that every timed state $q \in Q$ has at least one state in $\mathsf{TO}(q)$:

Proposition 5.5. *Let TA^{MP} be a message-passing timed automaton, and $TS(TA^{MP}) = (Q, Q_0, \Delta, \lambda)$. For every state $q \in Q$, the set $\mathsf{TO}(q)$ is not empty.*

Using Proposition 5.5, we show that the well-defined states simulate all the timed states of a message-passing timed automaton:

Theorem 5.6. *If TA^{MP} is a message-passing timed automaton, and $TS(TA^{MP}) = (Q, Q_0, \Delta, \lambda)$, then $\{(q, r)\colon q \in Q, r \in \mathsf{TO}(q)\}$ is an initial timed simulation.*

Theorem 5.6 suggests that timed automata restricted to well-formed states might help us in avoiding the negative result of Theorem 5.1. To this end, we introduce a *well-formed message-passing timed automaton*. Before that, we note that Equation (1) of Definition 5.3 can be transformed to a state invariant. We denote such a state invariant as I^{WF}.

Definition 5.7 (Well-formed MPTA). *Given a message-passing timed automaton $TA^{MP} = (S, S_0, R, L, U \cup X, I, E)$, its well-formed restriction TA^{MP}_{WF} is the timed automaton $(S, S_0, R, L, U \cup X, I \wedge I^{WF}, E)$.*

Since the well-formed states are included in the set of timed states, and the well-formed states simulate timed states (Theorem 5.6), we obtain the following:

Corollary 5.8. *Let TA^{MP} be a message-passing timed automaton and TA^{MP}_{WF} be its well-formed restriction. These timed automata are timed-simulation equivalent: $TA^{MP} \simeq^t TA^{MP}_{WF}$.*

As a consequence of Theorems 3.4, 5.6, and Corollary 5.8, we now show that there is a timed bisimulation equivalence between a well-formed message-passing timed automaton and the corresponding message-counting timed automaton, which is obtained by forgetting the sender of the messages and just counting the sent and delivered messages.

Theorem 5.9. *Let TA^{MP} be a message-passing timed automaton and TA^{MC} be a message-counting timed automaton defined over the same timed system design. Further, let TA^{MP}_{WF} be the well-formed restriction of TA^{MP}. There exists an initial timed bisimulation: $TA^{MP}_{WF} \approx^t TA^{MC}$.*

By collecting Theorem 5.9 and Corollary 5.8 we conclude that there is a timed simulation equivalence between MPTA and MCTA:

Corollary 5.10. *Let TA^{MP} be a message-passing timed automaton and TA^{MC} be a message-counting timed automaton defined over the same timed system design. TA^{MP} and TA^{MC} are timed-simulation equivalent: $TA^{MP} \simeq^t TA^{MC}$.*

Figure 7 uses arrows to depict the timed simulations presented in this work.

6 Conclusions

Asynchronous systems. For systems considered in Section 3, we conclude from Theorem 3.4 that message-counting systems are detailed enough for model checking of properties written in $\mathsf{CTL}^\star$:

Corollary 6.1. *For a $\mathsf{CTL}^\star$ formula φ, a message-passing system TS^{MP} and a message-counting system TS^{MC} defined over the same design, $TS^{MP} \models \varphi$ if and only if $TS^{MC} \models \varphi$.*

The corollary implies that the message counting abstraction does not introduce spurious behavior. In contrast, data and counter abstractions introduced in [22] may lead to spurious behavior as only simulation relations have been shown for these abstractions.

Timed systems. For systems considered in Section 4, we consider specifications in the temporal logic ATCTL [14], which restricts TCTL [6] as follows: first, negations only appear next to propositions $p \in \mathsf{AP} \cup \Psi(U)$, and second, the temporal operators are restricted to $\mathsf{A}\,\mathsf{F}_{\sim c}$, $\mathsf{A}\,\mathsf{G}_{\sim c}$, and $\mathsf{A}\,\mathsf{U}_{\sim c}$.

To derive that message-counting timed automata are sufficiently precise for model checking of ATCTL formulas (in the following corollary), we combine the following results: (i) Simulation-equivalent systems satisfy the same formulas of ACTL, e.g. see [11, Theorem 7.76]; (ii) Reduction of TCTL model checking to CTL model checking by clock embedding [11, p. 706]; (iii) Corollary 5.10.

Corollary 6.2. *For a message-passing timed automaton* TA^{MP} *and a message-counting timed automaton* TA^{MC} *defined over the same timed design and an* ATCTL*-formula* φ*, the following holds:* $TA^{MP} \models \varphi$ *if and only if* $TA^{MC} \models \varphi$*.*

Future work. Most of the timed specifications of interest for FTDAs (e.g., fault-tolerant clock synchronization algorithms [35, 39, 40]) are examples of time-bounded specifications, thus belonging to the class of timed safety specifications. These algorithms can be encoded as message-passing timed automata (Definition 4.1). In this paper, we have shown that model checking of these algorithms can also be done at the level of message-counting timed automata (Definition 4.2). Based on this it appears natural to apply the abstraction-based parameterized model checking technique from [22]. However, we are still facing the challenge of having a parameterized number of clocks in Definition 4.2. We are currently working on another abstraction that addresses this issue. This will eventually allow us to do parameterized model checking of timed fault-tolerant distributed algorithms using UPPAAL [12] as back-end model checker.

Related work. As discussed in [23], while modeling message passing is natural for fault-tolerant distributed algorithms (FTDAs), message counting scales better for asynchronous systems, and also builds a basis for efficient parameterized model checking techniques [22, 28]. We are interested in corresponding results for timed systems, that is, our long-term research goal is to build a framework for the automatic verification of timed properties of FTDAs. Such kind of properties are particularly relevant for the analysis of distributed clock synchronization protocols [35, 39, 40]. This investigation combines two research areas: (i) verification of FTDAs and (ii) parameterized model checking (PMC) of timed systems.

To the best of our knowledge, most of the existing literature on (i) can model only the discrete behaviors of the algorithms themselves [38, 20, 22, 28, 18, 4, 5]. Consequently they can neither reason about nor verify their timed properties. This motivated us to extend existing techniques for modeling and abstracting FTDAs, such as message passing and message counting systems together with the message counting abstraction, to timed systems.

Most of the results about PMC of timed systems [2, 31, 8, 10, 3, 1, 34, 9] are restricted to systems whose interprocess communication primitives have other systems in mind than FTDAs. For instance, the local state space is fixed and finite and independent of the parameters, while message counting in FTDAs requires that the local state space depends on the parameters. This motivated us to introduce the notions of *message passing timed automata* and *message counting timed automata*. Besides, the literature typically focuses on decidability, e.g., [3, 1, 34, 9]) analyze decidability for different variants of the parameterized model checking problem (e.g., integer vs. continuous time, safety vs. liveness, presence vs. absence of controller). Our work focuses on establishing relations between different timed models, with the goal of using these relations for abstraction-based model checking.

References

1. P.A. Abdulla, J. Deneux, and P. Mahata. Multi-clock timed networks. In *LICS*, pages 345–354, 2004.
2. Parosh Aziz Abdulla, Frédéric Haziza, and Lukáš Holík. All for the price of few. In *VMCAI*, pages 476–495, 2013.
3. Parosh Aziz Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, 290(1):241–264, 2003.
4. Francesco Alberti, Silvio Ghilardi, Andrea Orsini, and Elena Pagani. Counter abstractions in model checking of distributed broadcast algorithms: Some case studies. In *CILC*, pages 102–117, 2016.
5. Francesco Alberti, Silvio Ghilardi, and Elena Pagani. Counting constraints in flat array fragments. In *IJCAR*, pages 65–81, 2016.
6. Rajeev Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. *LICS*, pages 414–425, 1990.
7. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
8. Benjamin Aminof, Tomer Kotek, Sasha Rubin, Francesco Spegni, and Helmut Veith. Parameterized Model Checking of Rendezvous Systems. *CONCUR*, 23(c):1–16, 2014.
9. Benjamin Aminof, Sasha Rubin, Florian Zuleger, and Francesco Spegni. Liveness of parameterized timed networks. In *ATVA*, pages 375–387, 2015.
10. Simon Außerlechner, Swen Jacobs, and Ayrat Khalimov. Tight cutoffs for guarded protocols with fairness. In *VMCAI*, pages 476–494, 2016.
11. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
12. Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In *QEST*, pages 125–126, 2006.
13. Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.
14. Peter Bulychev, Thomas Chatain, Alexandre David, and Kim G Larsen. Efficient on-the-fly algorithm for checking alternating timed simulation. In *Formal Modeling and Analysis of Timed Systems*, pages 73–87. Springer, 2009.
15. Kārlis Čerāns. Decidability of bisimulation equivalences for parallel timer processes. In *CAV*, volume 663 of *LNCS*, pages 302–315, 1993.
16. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
17. Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.
18. Cezara Drăgoi, Thomas A. Henzinger, Helmut Veith, Josef Widder, and Damien Zufferey. A logic-based framework for verifying consensus algorithms, 2014.
19. Michael J. Fischer, Nancy A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.
20. Dana Fisman, Orna Kupferman, and Yoad Lustig. On verifying fault tolerance of distributed protocols. In *TACAS*, volume 4963 of *LNCS*, pages 315–331. Springer, 2008.
21. Matthias Függer and Ulrich Schmid. Reconciling fault-tolerant distributed computing and systems-on-chip. *Distributed Computing*, 24(6):323–355, 2012.

22. Annu John, Igor Konnov, Ulrich Schmid, Helmut Veith, and Josef Widder. Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In *FMCAD*, pages 201–209, 2013.
23. Annu John, Igor Konnov, Ulrich Schmid, Helmut Veith, and Josef Widder. Towards modeling and model checking fault-tolerant distributed algorithms. In *SPIN*, volume 7976 of *LNCS*, pages 209–226, 2013.
24. Dilsun Kirli Kaynar, Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. *The Theory of Timed I/O Automata*. Synthesis Lectures on Computer Science. Morgan & Claypool Publishers, 2006.
25. Igor Konnov, Marijana Lazić, Helmut Veith, and Josef Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In *POPL*, 2017. (to appear, preliminary version at http://arxiv.org/abs/1608.05327).
26. Igor Konnov, Helmut Veith, and Josef Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. In *CONCUR*, volume 8704 of *LNCS*, pages 125–140, 2014.
27. Igor Konnov, Helmut Veith, and Josef Widder. SMT and POR beat counter abstraction: Parameterized model checking of threshold-based distributed algorithms. In *CAV (Part I)*, volume 9206 of *LNCS*, pages 85–102, 2015.
28. Igor Konnov, Helmut Veith, and Josef Widder. What you always wanted to know about model checking of fault-tolerant distributed algorithms. In *PSI 2015, Revised Selected Papers*, volume 9609 of *LNCS*, pages 6–21. Springer, 2016.
29. Nancy A. Lynch and Frits W. Vaandrager. Forward and backward simulations for timing-based systems. In *Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991, Proceedings*, pages 397–446, 1991.
30. Achour Mostéfaoui, Eric Mourgaya, Philippe Raipin Parvédy, and Michel Raynal. Evaluating the condition-based approach to solve consensus. In *DSN*, pages 541–550, 2003.
31. Kedar S. Namjoshi and Richard J. Trefler. Uncovering symmetries in irregular process networks. In *VMCAI*, pages 496–514, 2013.
32. Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J.ACM*, 27(2):228–234, 1980.
33. Yee Jiun Song and Robbert van Renesse. Bosco: One-step Byzantine asynchronous consensus. In *DISC*, volume 5218 of *LNCS*, pages 438–450, 2008.
34. Luca Spalazzi and Francesco Spegni. Parameterized Model-Checking of Timed Systems with Conjunctive Guards. In *Verified Software: Theories, Tools and Experiments*, pages 235–251. Springer, 2014.
35. T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *J. ACM*, 34(3):626–645, 1987.
36. T.K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80–94, 1987.
37. Stavros Tripakis and Sergio Yovine. Analysis of timed systems using time-abstracting bisimulations. *FMSD*, 18:25–68, 2001.
38. Tatsuhiro Tsuchiya and André Schiper. Verification of consensus algorithms using satisfiability solving. *Distributed Computing*, 23(5–6):341–358, 2011.
39. Josef Widder and Ulrich Schmid. Booting clock synchronization in partially synchronous systems with hybrid process and link failures. *Distributed Computing*, 20(2):115–140, 2007.
40. Josef Widder and Ulrich Schmid. The Theta-Model: Achieving synchrony without clocks. *Distributed Computing*, 22(1):29–47, April 2009.