Virtually Timed Ambients: A Calculus of Nested Virtualization[☆]

Einar Broch Johnsen, Martin Steffen, Johanna Beate Stumpf[*]

University of Oslo, Oslo, Norway

Abstract

Nested virtualization enables a virtual machine, which is a software layer representing an execution environment, to be placed inside another virtual machine. Nested virtual machines form a location hierarchy where virtual machines at every level in the hierarchy compete with other processes at that level for processing time. With nested virtualization, the computing power of a virtual machine depends on its position in this hierarchy and may change if the virtual machine moves. This paper introduces the calculus of virtually timed ambients, a formal model of hierarchical locations for execution with explicit resource provisioning, motivated by these effects of nested virtualization. Resource provisioning in this model is based on virtual time slices as a local resource. To reason about timed behavior in this setting, weak timed bisimulation for virtually timed ambients is defined as an extension of bisimulation for mobile ambients. We show that the equivalence of contextual bisimulation and reduction barbed congruence is preserved by weak timed bisimulation. Simulation with time relaxation is defined to express that a system is slower than another system up to a given time bound. The calculus of virtually timed ambients is illustrated by examples.

Keywords: process calculi, ambient calculi, models of distributed systems, models of nested virtualization, weak timed bisimulation

[☆]The work was supported by the Research Council of Norway through the CUMULUS project and through the SIRIUS Centre for Scalable Data Access (`www.sirius-labs.no`).

[*]Corresponding author

Email addresses: `einarj@ifi.uio.no` (Einar Broch Johnsen), `msteffen@ifi.uio.no` (Martin Steffen), `johanbst@ifi.uio.no` (Johanna Beate Stumpf)

1. Introduction

Virtualization technology enables the resources of an execution environment to be represented as a software layer, a so-called *virtual machine*. Application-level processes are agnostic to whether they run on such a virtual machine or directly on physical hardware. Since a virtual machine is a process, it can be executed on another virtual machine. Technologies such as VirtualBox, VMWare ESXi, Ravello HVX, Microsoft Hyper-V, and the open-source Xen hypervisor increasingly support running virtual machines inside each other in this way. This *nested virtualization*, originally introduced by Goldberg [16], is necessary to host virtual machines with operating systems which themselves support virtualization [5], such as Microsoft Windows 7 and Linux KVM. Nested virtualization has many uses, for example for end-user virtualization for guests, in development, and in deployment testing. Nested virtualization is also a crucial technology to support the hybrid cloud, as it enables virtual machines to migrate between different cloud providers [37].

To study the logical behavior of virtual machines in the context of nested virtualization, this paper develops a calculus of virtually timed ambients with explicit resource provisioning. Previous work on process algebra with resources typically focusses on binary resources such as locks (e.g., [22, 29]) and previous work on process algebra with time mainly considers timeouts (e.g., [31, 4, 28, 17, 6]). In contrast, time and resources in virtually timed ambients are *quantitative* notions: a process which gets *more resources* typically executes *faster*. Virtually timed ambients can be understood as locations for the deployment of processes; the resource requirements of processes executing at a location are matched by resources made available by the virtually timed ambient. The amount of resources made available by a virtually timed ambient constitutes its computing power. This amount is determined by the time slices the virtually timed ambient receives from its parent ambient. A virtually timed ambient that shares the time slices of its parent ambient with another process has less available time slices to execute its own processes.

The time model used to realize this kind of resource provisioning for virtually timed ambients is here called *virtual time*. Virtual time is provided to a virtually timed ambient by its parent ambient, similar to the time slices that an operating system provisions to its processes. When we consider many levels of nested virtualization, virtual time becomes a *local* notion of time which depends on a virtually timed ambient's position in the location hierarchy. Virtually timed ambients are mobile, reflecting that virtual machines may migrate between host virtual machines. Observe that such migration affects the execution speed of processes executed in the virtually timed ambient which moves, in the virtually timed ambients it leaves, and in the virtually timed ambient it enters. The model of resource provisioning in virtually timed ambients is inspired by Real-Time ABS [20], but extended to address nested virtualization in our calculus.

To formalize nested virtualization, notions of location mobility and nesting are essential. The calculus of mobile ambients, originally developed by Cardelli and Gordon [9], captures processes executing at distributed locations in networks such as the Internet. Mobile ambients model both location mobility and nested locations, which makes this calculus well-suited as a starting point for our work. Combining these notions from the ambient calculus with the concepts of virtual time and resource provisioning, the calculus of virtually timed ambients can be seen as a model of nested virtualization. To capture migration, virtually timed ambients will have capabilities reminiscent of those for mobile ambients, but the capabilities of virtually timed ambients need to deal with virtual time and the corresponding changes to the resource provisioning. Thus different locations, barriers between locations, barrier crossing, and their relation to virtual time and resource provisioning are important for the virtually timed ambients; the number and position of virtually timed ambients available for processing tasks influences the overall processing time of a program. This allows the effects of, e.g., load balancing and scaling to be observed using weak timed bisimulation.

Contributions. To study the effects of nested virtualization, the main contributions of this paper can be summarized as follows:

- we define a calculus of *virtually timed ambients*, to the best of our knowledge the first process algebra capturing notions of virtual time and resource provisioning for nested virtualization;
- we define *weak timed bisimulation* for virtually timed ambients, and show that weak timed bisimulation is equivalent to reduction barbed congruence [25] with time;
- we define *time relaxation* for virtually timed ambients as a simulation relation allowing deviation by a bounded amount of time.

A short version of this paper appeared in the proceedings of WADT 2016 [21].

	n	name
Systems:		
$M, N ::=$	$\mathbf{0}$	inactive system
	$M \mid N$	parallel composition
	$(\nu n)M$	restriction
	$n[P]$	ambient
Processes:		
$P, Q ::=$	$\mathbf{0}$	inactive process
	$P \mid Q$	parallel composition
	$(\nu n)P$	restriction
	$!C.P$	replication
	$C.P$	prefixing
	$n[P]$	ambient
Capabilities:		
$C ::=$	*in* n	can enter n
	out n	can exit n
	open n	can open n

Table 1: Syntax of the mobile ambient calculus.

2. Preliminaries on Mobile Ambients

Mobile ambients [9] have originally been introduced to represent "administrative domains" for processes. The syntax, as well as the semantics we consider, is based on [25] and largely unchanged compared to [9]. The main difference compared to [9] lies in the separation of processes into two levels, as *processes* and *systems*. This distinction is used to simplify proofs in the bisimulation section. Systems characterize the outermost layer of an ambient structure.

The syntax in Table 1 represents the basic calculus for mobile ambients. The inactive process $\mathbf{0}$ does nothing. The parallel composition $P \mid Q$ allows both processes P and Q to proceed concurrently, where the binary operator $\mid$ is commutative and associative. The restriction operator $(\nu n)P$ creates a new and unique name with process P as its scope. In the calculus, administrative domains for processes, called *ambients*, are represented by names. A process P located in an ambient named m is written $m[P]$.

Ambients can be nested, and the nesting structure can change dynami-

$P \rightarrowtriangle Q \Rightarrow (\nu n)P \rightarrowtriangle (\nu n)Q$	(R-Res)
$P \rightarrowtriangle Q \Rightarrow P \mid R \rightarrowtriangle Q \mid R$	(R-Par)
$P \rightarrowtriangle Q \Rightarrow n[P] \rightarrowtriangle n[Q]$	(R-Amb)
$P' \equiv P, P \rightarrowtriangle Q, Q \equiv Q' \Rightarrow P' \rightarrowtriangle Q'$	(R-Red $\equiv$)
$n[\mathit{in}\ m.P \mid Q] \mid m[R] \rightarrowtriangle m[n[P \mid Q] \mid R]$	(R-In)
$m[n[\mathit{out}\ m.P \mid Q] \mid R] \rightarrowtriangle n[P \mid Q] \mid m[R]$	(R-Out)
$\mathit{open}\ n.P \mid n[Q] \rightarrowtriangle P \mid Q$	(R-Open)

Table 2: Reduction rules.

cally. A change of the nesting structure is specified by prefixing a process with a *capability*. There are three basic capabilities. The input capability $\mathit{in}\ n$ indicates the willingness of a process (respectively its containing ambient) to enter an ambient named n, running in parallel with its own ambient; e.g., $k[\mathit{in}\ n.P] \mid n[Q] \rightarrowtriangle n[k[P] \mid Q]$. The output capability $\mathit{out}\ n$ enables an ambient to leave its surrounding (or parental) ambient n; e.g., $n[k[\mathit{out}\ n.P] \mid Q] \rightarrowtriangle k[P] \mid n[Q]$. The open capability $\mathit{open}\ n$ allows an ambient named n at the same level as the capability to be opened; e.g., $k[\mathit{open}\ n.P \mid n[Q]] \rightarrowtriangle k[P \mid Q]$.

2.1. Syntax

We use the following notational conventions. Parallel composition has the lowest precedence among the operators and the process $E.F.P$ is read as $E.(F.P)$. For names, the ν-operator acts as a binder and the sets of free names *fn* of a process is defined as expected. We use $\widetilde{m}$ to denote a tuple of names $m_1, m_2, \ldots, m_k$. Note that Table 1, following [25], only allows replicated prefixing $!C.P$. Since replicated input in the π-calculus has the same expressive power as full replication [18] and recursion [26, 34], the replication of an ambient in the ambient calculus can be implemented with replicated prefixing and stuttering. We shall use the notion $!n[]$ in the examples as an abbreviation for this stuttering bypass.

2.2. Semantics

The semantics is given as a reduction semantics which combines structural congruence with reduction rules. The reduction relation $P \rightarrowtriangle Q$, which describes the evolution of a process, is captured by the reduction rules in Table 2. Structural congruence $P \equiv Q$ relates different syntactic representations of the same process and can be seen in Table 3.

$P \equiv P$	(S-Refl)
$P \mid \mathbf{0} \equiv P$	(S-ZeroPar)
$P \equiv Q \Rightarrow Q \equiv P$	(S-Symm)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(S-Trans)
$P \mid Q \equiv Q \mid P$	(S-ParComm)
$(P \mid Q) \mid R \equiv Q \mid (P \mid R)$	(S-Par-Assoc)
$!C.P \equiv C.P \mid !C.P$	(S-Repl Par)
$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$	(S-ResRes)
$(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q$ if $n \notin fn(P)$	(S-Res-Par)
$(\nu n)(m[P]) \equiv m[(\nu n)P]$ if $n \neq m$	(S-Res-Amb)

Table 3: Structural congruence.

3. Virtually Timed Ambients

Mobile ambients [9] are located processes, arranged in a hierarchy which may change dynamically. Interpreting the location as a place of deployment, virtually timed ambients extend mobile ambients with notions of virtual time and resource consumption. The timed behavior depends on the one hand on the *local* timed behavior, but on the other hand on the placement or deployment of the component in the hierarchical ambient structure. Virtually timed ambients combine timed processes and timed capabilities with the features of the calculus for mobile ambients summarized above.

3.1. Syntax and Semantics

Timed systems and processes are defined analogously to Table 1, with the difference that each virtually timed ambient contains a *local clock* and other virtually timed ambients or processes.

Definition 1 (Virtually timed ambients)**.** *Virtually timed ambients are given by the syntax in Table 4.*

In Table 4 we can see that in the calculus of virtually timed ambients every closed system of ambients must be contained in a root ambient with a *source clock* triggering the clocks of the local subambients recursively. The timed capabilities of virtually timed ambients extend the capabilities of mobile ambients with an additional effect on time management, explained below. In order to define computing power a capacity **consume** for resource consumption of processes is added.

	n	name
	`tick`	virtual time slice
Global systems:		
$G ::=$	$\mathbf{0}$	inactive system
	$G \mid G$	parallel composition
	$n[\textsc{Source} \mid M]$	virtually timed root ambient
Timed systems:		
$M, N ::=$	$\mathbf{0}$	inactive system
	$M \mid N$	parallel composition
	$(\nu n)M$	restriction
	$n[\textsc{Clock} \mid P]$	virtually timed ambient
Timed processes:		
$P, Q ::=$	$\mathbf{0}$	inactive process
	$P \mid Q$	parallel composition
	$(\nu n)P$	restriction
	$!C.P$	replication
	$C.P$	prefixing
	$n[\textsc{Clock} \mid P]$	virtually timed ambient
Timed capabilities:		
$C ::=$	**in** n	can enter n and adjust the local clock there
	out n	can exit n and adjust the local clock on the outside
	open n	can open n and adjust own local clock
	consume	consumes one resource

Table 4: Syntax of the virtually timed ambient calculus.

The semantics is given as a reduction semantics similar to the semantics of mobile ambients. The rules for structural congruence $P \equiv Q$ are equivalent to the ones for mobile ambients in Table 3. The reduction relation $P \rightarrowtriangle Q$ is captured by the rules in Table 5 and Table 6. In Table 5 we make use of the notion of *observables* aka barbs. This well-known concept, originally introduced for the π-calculus [27], captures a notion of immediate observability. In the ambient calculus, what is immediately observable is the presence of a top-level ambient whose name is not restricted. The obervability predicate $\downarrow_n$ or "barb" is defined as follows.

Definition 2 (Barbs)**.** *Process P strongly barbs on n, written $P\downarrow_n$, if $P \equiv (\nu\widetilde{m})(n[P_1] \mid P_2)$, where $n \notin \{\widetilde{m}\}$.*

By moving the ν-binders to the outside and only taking the inside of their scope into consideration we can observe the bound ambients inside the scope of the ν-binders.

Definition 3. *For a process P we define $P_\downarrow$ as the set of all top level ambients $P_\downarrow := \{n \mid P \equiv (\nu\widetilde{m})P' \land P'$ is ν-binder free $\land P'\downarrow_n\}$.*

3.2. Virtual Time and Local Clocks

To represent the outlined time model the local clock contained in each virtually timed ambient is responsible for triggering timed behavior and local resource consumption. Each time slice emitted by a local clock triggers the clock of one of its subambients in a round-robin way or is consumed by a process as a resource. This corresponds to a simple form of fair, *preemptive scheduling*, which makes the system's behavior sensitive to the number of co-located virtually timed ambients and resource consuming processes.

Clocks have a *speed*, interpreted *relative* to the speed of the surrounding virtually timed ambient. The speed of a clock is given by the pair (p, q), where p is the number of local time slices emitted for a number q of time slices received from the surrounding ambient, $p, q \in \mathbb{N}$. Thus, time in a nested ambient is relative to the global time, and depends on the speed of the clocks of the ambients in which it is contained and on its number of siblings.

The speed of the source clocks is defined as a pair $(n, 0)$, where $n \in \mathbb{N}$, as the sources do not need any input, while for the speed of a local clock it holds that an input of $q = 0$ is only valid if $p = 0$, too. As virtually timed ambients with speed $(0, 0)$ do not require any times slices from their parental ambient and do not exhibit any timed behavior, they are not considered *time consuming*. However, processes which are prefixed with the resource consumption capability **consume**.P are considered time consuming. Note that mobile ambients can be represented as virtually timed ambients with a clock with speed $(0, 0)$.

$$\frac{\begin{array}{l}
\textsc{Clock}_k = \textsc{Clock}\{c_k, (p_k, q_k), \{m, n\}, \emptyset\} \\
\textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, \dots, a_k\}, \{a_{k+1}, \dots a_n\}\} \\
\textsc{Clock}_n = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \dots, b_i\}, \{b_{i+1}, \dots b_j\}\} \\
\textsc{Clock}^*_k = \textsc{Clock}\{c_k, (p_k, q_k), \{m\}, \emptyset\} \\
\textsc{Clock}^*_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, \dots, a_k, n\}, \{a_{k+1}, \dots a_n\}\} \\
\textsc{Clock}^*_n = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \dots, b_i\} \cup P_{\downarrow}, \{b_{i+1}, \dots b_j\}\}
\end{array}}{\begin{array}{c}
k[\textsc{Clock}_k \mid n[\textsc{Clock}_n \mid \textbf{in}\ m.P \mid Q] \mid m[\textsc{Clock}_m \mid R]] \\
\rightharpoonup k[\textsc{Clock}^*_k \mid m[\textsc{Clock}^*_m \mid R \mid n[\textsc{Clock}_n \mid P \mid Q]]]
\end{array}} \quad (\text{TR-}\textsc{In})$$

$$\frac{\begin{array}{l}
\textsc{Clock}_k = \textsc{Clock}\{c_k, (p_k, q_k), \emptyset, \{m\}\} \\
\textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \dots, a_k, n\}, \{a_{k+1}, \dots a_n\}\} \\
\textsc{Clock}_n = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \dots, b_i\}, \{b_{i+1}, \dots b_j\}\} \\
\textsc{Clock}^*_k = \textsc{Clock}\{c_k, (p_k, q_k), \{n\}, \{m\}\} \\
\textsc{Clock}^*_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \dots, a_k\}, \{a_{k+1}, \dots a_n\}\} \\
\textsc{Clock}^*_n = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \dots, b_i\} \cup P_{\downarrow}, \{b_{i+1}, \dots b_j\}\}
\end{array}}{\begin{array}{c}
k[\textsc{Clock}_k \mid m[\textsc{Clock}_m \mid n[\textsc{Clock}_n \mid \textbf{out}\ m.P \mid Q] \mid R]] \\
\rightharpoonup k[\textsc{Clock}^*_k \mid n[\textsc{Clock}_n \mid P \mid Q] \mid m[\textsc{Clock}^*_m \mid R]]
\end{array}} \quad (\text{TR-}\textsc{Out})$$

$$\frac{\begin{array}{l}
\textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{n\}, \emptyset\} \\
\textsc{Clock}^*_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \dots, a_n\} \cup P_{\downarrow}, \emptyset\}
\end{array}}{m[\textsc{Clock}_m \mid \textbf{open}\ n.P \mid n[\textsc{Clock}_n \mid R] \mid Q] \rightharpoonup m[\textsc{Clock}^*_m \mid P \mid R \mid Q]} \quad (\text{TR-}\textsc{Open})$$

$$\frac{\begin{array}{l}
\textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \dots, a_n\}, \emptyset\} \\
\textsc{Clock}^*_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \dots, a_n, \textbf{consume}.P\}, \emptyset\} \\
p_m > 0
\end{array}}{m[\textsc{Clock}_m \mid \textbf{consume}.P \mid R] \rightharpoonup m[\textsc{Clock}^*_m \mid R]} \quad (\text{TR-}\textsc{Resource})$$

Table 5: Timed reduction rules for timed capabilities, where $a_1, a_2, \dots, a_n$ are time consuming virtually timed ambients and processes in R and $b_1, b_2, \dots, b_j$ in Q, respectively. Here a blue backdrop marks the trigger of the reduction, red the changes in the clocks and green eventual constraints.

$$\frac{\begin{array}{l}\textsc{Clock} = \textsc{Clock}\{\ c\ , (p,q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\} \\ \textsc{Clock}^* = \textsc{Clock}\{\ c+1\ , (p,q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\} \\ c+1 < q\end{array}}{m[\ \texttt{tick}\ \mid \textsc{Clock} \mid R] \rightharpoonup m[\textsc{Clock}^* \mid R]} \quad (\text{TR-}\textsc{Tick}_1)$$

$$\frac{\begin{array}{l}\textsc{Clock} = \textsc{Clock}\{c, (p,q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\} \\ c+1 = q\end{array}}{m[\ \texttt{tick}\ \mid \textsc{Clock} \mid R] \rightharpoonup m[\ \mathrm{RR}(\textsc{Clock} \mid R)\ ]} \quad (\text{TR-}\textsc{Tick}_2)$$

$$\frac{\textsc{Source} = \textsc{Source}\{0, (n,0), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}}{\textsc{Source} \mid R \rightharpoonup\ \mathrm{RR}(\textsc{Source} \mid R)} \quad (\text{TR-}\textsc{Source})$$

Table 6: Timed reduction rules, where $a_1, a_2, \ldots, a_n$ are time consuming virtually timed ambients and processes in R, and for (TR-Source) on the top level. Here a blue backdrop marks the trigger of the reduction, red the changes in the clocks and green eventual constraints.

Definition 4 (Local clocks)**.** *A local clock contains a* counter *to record the number of received time slices, its own* speed*, and two* sets*:*

$$\textsc{Clock}\{counter, (p,q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}\ .$$

The first set contains the names of time consuming processes running in the ambient as well as time consuming virtually timed subambients in the surrounding ambient which have not yet received a time slice in the current cycle and the second set those which have.

When a clock receives a time slice, denoted `tick`, from its surrounding ambient, one of the following actions occurs: If $counter + 1 < q$, then the clock records this time slice and continues waiting (i.e., $\textsc{Clock}\{counter := counter + 1, (p,q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}$); if $counter + 1 = q$, then the input number is reached, the counter is set to 0 and the clock emits time slices to p subambients of the first set and puts them in the second set (i.e., $\textsc{Clock}\{counter := 0, (p,q), \{a_{p+1} \ldots, a_k\}, \{a_{k+1}, \ldots a_n, a_1, a_2, \ldots, a_p\}\}$). As soon as the first of the two sets is empty, the first and second set are switched. Thus, no ambient receives a second time slice before every other subambient has received the first one. In the following, we omit the representation of the counter and the sets of subambients. For a better overview in the examples we denote the speed of the clocks as superscript $\textsc{Clock}^{p,q}$. If an ambient is not time consuming, i.e, it has a clock with speed $(0, 0)$, we do not mention the clock. For actions which do not require time we assume *maximal progress.* In terms of structural congruence a clock is treated analogously to a process.

The following example shows the encoding of a system with a load balancer in virtually timed ambients.

Example 1 (System with load balancer)**.** *A system with a load balancer can be defined as follows:*

$$\begin{aligned}
\textit{load balancer system:}\ & (\nu\ lb, a, b)\ lbs[\textsc{Clock}^{2,1} \mid lb \mid a \mid b] \\
\textit{incoming request:}\ & request[P.done\_signal \mid in\ lbs.enter\_signal.\textbf{\textit{open}}\ move] \\
\textit{load balancer:}\ & lb[!\textbf{\textit{open}}\ start.wait\_for\_enter.\textbf{\textit{open}}\ lock_a. \\
& \quad wait\_for\_enter.\textbf{\textit{open}}\ lock_b.start[]\ |!lock_a[x[]] \mid \\
& \quad !lock_b[y[]]\ |!(\textbf{\textit{open}}\ x.move[\textbf{\textit{out}}\ lb.\textbf{\textit{in}}\ request.\textbf{\textit{in}}\ a] \mid \\
& \quad \textbf{\textit{open}}\ y.move[\textbf{\textit{out}}\ lb.\textbf{\textit{in}}\ request.\textbf{\textit{in}}\ b])] \\
\textit{ambient a:}\ & a[\textsc{Clock}^{1,1}\ |!\textbf{\textit{open}}\ request.wait\_for\_done. \\
& \quad done[\textbf{\textit{out}}\ a.\textbf{\textit{out}}\ lbs]] \\
\textit{ambient b:}\ & b[\textsc{Clock}^{1,1}\ |!\textbf{\textit{open}}\ request.wait\_for\_done. \\
& \quad done[\textbf{\textit{out}}\ b.\textbf{\textit{out}}\ lbs]].
\end{aligned}$$

Here, the untimed load balancer creates a move ambient which moves incoming requests alternately into the virtually timed ambients a and b. For each time slice it receives from the source clock of the surrounding root ambient, the local clock of lbs distributes two time slices. Therefore, both subambients a and b receive one time slice. When a request has been executed, it releases an ambient done which emerges to the outside of the system and becomes observable. The enter- and done-signals are shorthand notions, but can easily be implemented in the calculus of virtually timed ambients.

3.3. Timed Capabilities

The timed capabilities **in** n, **out** n, and **open** n enable virtually timed ambients to move in a timed system. When moving virtually timed ambients, we must consider that the clocks need to know about their current subambients, therefore their list of subambients need to be adjusted.

RR: Round-based scheduling function

```
input: $\textsc{Clock}\{\mathit{counter}, (p, q), \{a_1, a_2, \dots, a_k\}, \{a_{k+1}, \dots, a_n\}\} \mid R$
$S := \{a_1, a_2, \dots, a_k\}$
$T := \{a_{k+1}, \dots, a_n\}$
**if** $S = \emptyset$ **then**
**return** $\textsc{Clock}\{0, (p, q), \emptyset, \emptyset\}$
**else**
    **while** $p \geq |S|$ **do**
        **for all** $a_i \in S$ **do**
            **if** $a_i =$ **consume**$.P$ **then** $S := S \setminus a_i$; $R := R \mid P$
                $S := S \cup P_{\downarrow}$
            **else** $R := a_1 \mid \cdots \mid a_i[\texttt{tick} \mid \cdots] \mid \cdots \mid a_n$
            **end if**
        **end for**
        $p := p - |S|$; $S := S \cup T$; $T := \emptyset$
    **end while**
    Choose a subset $S' \subset S$ such that $|S'| = p$.
    **for all** $a_i \in S'$ **do**
        **if** $a_i =$ **consume**$.P$ **then** $S' := S \setminus a_i$, $R := R \mid P$
            $S := S \cup P_{\downarrow}$
        **else** $R := a_1 \mid \cdots \mid a_i[\texttt{tick} \mid \cdots] \mid \cdots \mid a_n$
        **end if**
    **end for**
    $S := S \setminus S'$; $T := T \cup S'$
**end if**
**return** $\textsc{Clock}\{0, (p, q), S, T\} \mid R$
```

We now explain the reduction rules for virtually timed ambients, which are given in Table 5 and Table 6. Observe that if we would not adjust the clocks then the moving subambient would not receive time slices from its new parental clock. In (TR-In) and (TR-Out), the clocks of the old and new parental ambient of the moving ambient have to be updated. In (TR-In) the clock of the opening ambient itself is updated. Note also that here the clock of the opened ambient is deleted. For virtually timed ambients with a clock with speed $(0, 0)$, the timed capabilities are equivalent to the capabilities for mobile ambients, as ambients, which are not time consuming, are not considered in the time management of the clocks. In (TR-Resource) the time consuming process is moved into the clock, where it awaits the distribution of a time slice as resource before it can continue. This reduction can only happen in virtually timed ambients with $p > 0$, meaning ambients which actually emit resources. Ambients which do not emit resources can therefore be used to safely transfer request between ambients which are used as computation environments without interfering with the contents of the requests. In (TR-TICK$_1$) the required number q of input time slices to trigger the local clock is not reached, thus the incoming time slice, denoted as `tick`, is only registered in the counter. In (TR-TICK$_2$) the local clock releases p time slices to its subambients and potentially to time consuming processes. This is denoted with the function RR for round-based scheduling of time slices. The function takes as input the CLOCK and distributes time slices `tick` to the subambients and processes in the given sets, thereby adjusting the sets of remaining and of served ambients. The source clocks SOURCE can reduce without parental time slices as given in (TR-SOURCE).

3.4. Resource Consumption

Processes expend the processing power of the ambient they are contained in by consuming the local time slices as resources. Thus, time consuming processes and time consuming subambients in a virtually timed ambient compete for the same resource. The consumption of a computing resource is defined as the capability **consume**. A process P without any appearance of **consume** is called *not time consuming*. An ambient with a higher local clock speed produces more time slices and therefore also more resources for each parental time slice, which in turn allows more work to be done for each parental time slice.

We illustrate resource consumption by considering a request which was sent to the system of Example 1.

Example 2 (Resource consumption)**.** *Consider the virtually timed system with a load balancer from Example 1, with an incoming request.*

$$lbs[\cdots] \mid request[\boldsymbol{consume}.\boldsymbol{consume}.done\_signal \mid \\ \boldsymbol{in}\ lbs.enter\_signal.\boldsymbol{open}\ move]$$

The request enters the system and is transferred by the load balancer into a, where it is opened during the reduction and awaits resource consumption.

After one time signal of the source clock, the virtually timed ambient a emits one resource, which is consumed by the request:

$$a[\textsc{Clock}^{1,1} \mid !\mathit{open}\ \mathit{request}.\mathit{wait\_for\_done}.\mathit{done}[\boldsymbol{out}\ a.\boldsymbol{out}\ \mathit{lbs}] \\ \mid \mathit{wait\_for\_done}.\mathit{done}[\boldsymbol{out}\ a.\boldsymbol{out}\ \mathit{lbs}] \mid \boldsymbol{consume}.\mathit{done\_signal}].$$

After another time signal from the source clock the ambient with name done can emerge to the top level:

$$a[\textsc{Clock}^{1,1} \mid !\mathit{open}\ \mathit{request}.\mathit{wait\_for\_done}.\mathit{done}[\boldsymbol{out}\ a.\mathit{out}\ \mathit{lbs}]] \mid \\ \mathit{done}[\boldsymbol{out}\ \mathit{lbs}].$$

3.5. Accumulated Speed

The *accumulated speed* (a_m, b_m) in an ambient m is the relative speed of the ambient with respect to the source clock and the siblings. As the clocks distribute time slices in a form of preemptive scheduling, such that each child gets one time slice in a round robin way, it holds that the accumulated speed of an ambient is influenced by the parental speed and the number of children n in the parental ambient. Thus, this approach is not only *path sensitive*, but also *sibling sensitive*.

Definition 5. *Let (a_{parent}, b_{parent}) be the accumulated speed in the direct parental ambient of an ambient m, $n_{children}$ the number of children of the parent, and C the chain of all parental ambients of m up to the global level, then the accumulated speed for preemptive scheduling in a subambient m is given as follows:*

$$\begin{aligned}(a_m, b_m) &= \left(p_m, \left\lceil q_m \cdot n_{children} \cdot \frac{max\{1, b_{parent}\}}{a_{parent}} \right\rceil\right) \\ &= \left(p_m, \left\lceil q_m \cdot \prod_{k \in C} n_{children\ of\ k} \cdot \prod_{k \in C} \frac{max\{1, b_k\}}{a_k} \right\rceil\right)\end{aligned}$$

Example 3 (Change of accumulated speed)**.** *Consider the virtually timed system lbs defined in Example 1, we define now a new system lbs_c which equals lbs, except that it contains a third subambient c.*

system: $(\nu\ lb, a, b, c)\ lbs_c[\textsc{Clock}^{2,1} \mid lb \mid a \mid b \mid c]$

ambient c: $c[\textsc{Clock}^{1,2} \mid !\boldsymbol{open}\ \mathit{request}.\mathit{wait\_for\_done}.\mathit{done}[\boldsymbol{out}\ c.\mathit{out}\ lbs_c]]$

As the approach is sibling sensitive, the accumulated speeds of the subambients of the system are reduced compared to the setup in Example 1 with only two subambients. Before, the accumulated speed of both a and b was $(1, 1)$*, now the speed of a and b is* $(1, 2)$ *and the accumulated speed of c is* $(1, 3)$*.*

4. Bisimulation and Barbs for Virtually Timed Ambients

When comparing the behavior of virtually timed ambients, we want to consider time as a factor. We first introduce a timed version of bisimulation and later of reduction barbed congruence.

The concept of (bi)simulation is an important, well-established, and extensively studied technique to define equivalences of concurrent or reactive systems [33]. It comes in many flavors and may or may not be a congruence, as well, depending on the constructs of the language and minutiae of the definition and the semantics. Being a congruence, of course, is generally intended, as that allows compositional arguments about equivalence of systems and replacing a subsystem by "equivalent" ones, without changing the overall behavior. A very important sub-class of bisimulations are so-called *weak* bisimulations; they are based on a distinction between observable and non-observable or internal actions, ignoring the latter. Ignoring internal behavior as unobservable is essential for being a useful basis of comparison, but unfortunately the issue of being a congruence or not becomes more tricky.

This section will define a notion of weak bisimulation for virtually timed ambients, generalizing the formalization for mobile ambients from [25], taking care of the timed behavior. To relate systems concerning their timed behavior or speed, in particular the "ticks" of the global clocks will be treated as observable. The standard notion bisimulation (weak or strong) relates two systems via the *transitions* each system makes, requiring that each (observable or every) step one system takes is mimicked accordingly by the other system, and vice versa.

In that setting, the steps of a system represent its atomic interactions with its environment, modeled as *labelled* transitions. The *reduction* semantics from Section 3, however, captures the behavior of *closed* or global systems. The semantics is formalized as (unlabelled) *reduction* steps $\rightarrowtriangle$ (additionally assisted by structural congruence rules), i.e., without interaction with the environment or with a surrounding context. To define bisimulation, we first formalize an *open* version of the operational semantics, using a labelled transition relation. To express interaction with a surrounding *context* in the open setting, transitions will have *labels* which capture interaction with an environment.

For this purpose, we define a semantics of virtually timed ambients based on labeled transition systems, which can perform global time steps as observable actions. The semantics is intended, of course, not as a semantically different alternative to the reduction semantics, but as capturing the same behavior, described from the perspective of open systems. For that transition semantics, we define a notion of weak timed bisimulation (cf. Section 4.1). Section 4.2 establishes that the introduced timed notion of bisimulation is a congruence. Following a standard line of development, this is done indirectly. First, one need to define a notion of bisimulation for the original semantics. It's defined contextually and based on so-called barbs, thereby fitting the closed-system reduction semantics. The resulting definition is a congruence, and the second part of the argument establishes that both definitions of bisimulation coincide.

4.1. Weak Bisimulation for Virtually Timed Ambients

Let us start by defining the available labels for the transition system semantics.

Definition 6 (Labels). *Let the set of labels Lab, with typical element α, be given as follows:*

$$\begin{array}{lcl} \alpha \in \mathit{Lab} & ::= & \tau \\ & \mid & k.\mathtt{enter\_}n \mid k.\mathtt{exit\_}n \mid k.\overline{\mathtt{enter\_}}n \mid n.\mathtt{open\_}k \\ & \mid & *.\mathtt{exit\_}n \mid *.\mathtt{enter\_}n \\ & \mid & k.\mathtt{tick} \end{array}$$

where k and n represent names of ambients. τ is the internal label*, the rest are called* observable *labels. We refer to labels of the forms $*.\mathtt{exit\_}n$ and $*.\mathtt{enter\_}n$ as* anonymous *and other labels as* non-anonymous*, and let the* untimed *labels exclude the tick labels.*

The behavior of a timed system interacting with its environment is given as a transmission system with transition labels from *Lab*. Note that the **consume** capability does not represent an interaction with an environment but an internal action and is therefore not captured by a separate observable label apart from τ.

$(\nu\widetilde{m})(m[\textsc{Clock} \mid \textbf{in } n.P \mid Q] \mid M), m \in \widetilde{m}$	
$\xrightarrow{*.\texttt{enter\_}n} (\nu\widetilde{m})(n[m[(\textsc{Clock} \mid P) \mid Q] \mid \circ\,] \mid M)$	(Enter Shh)
$(\nu\widetilde{m})(k[\textsc{Clock} \mid \textbf{in } n.P \mid Q] \mid M), k \notin \widetilde{m}$	
$\xrightarrow{k.\texttt{enter\_}n} (\nu\widetilde{m})(n[k[(\textsc{Clock} \mid P) \mid Q] \mid \circ\,] \mid M)$	(Enter)
$(\nu\widetilde{m})(m[\textsc{Clock} \mid \textbf{out } n.P \mid Q] \mid M), m \in \widetilde{m}$	
$\xrightarrow{*.\texttt{exit\_}n} (\nu\widetilde{m})(m[(\textsc{Clock} \mid P) \mid Q] \mid n[M \mid \circ\,])$	(Exit Shh)
$(\nu\widetilde{m})(k[\textsc{Clock} \mid \textbf{out } n.P \mid Q] \mid M), k \notin \widetilde{m}$	
$\xrightarrow{k.\texttt{exit\_}n} (\nu\widetilde{m})(k[(\textsc{Clock} \mid P) \mid Q] \mid n[M \mid \circ\,])$	(Exit)
$(\nu\widetilde{m})(k[(\textsc{Clock} \mid P)] \mid M), k \notin \widetilde{m}$	
$\xrightarrow{k.\overline{\texttt{enter}}\texttt{\_}n} (\nu\widetilde{m})(k[\textsc{Clock}^* \mid n[\circ] \mid P] \mid M)$	(Co-Enter)
$(\nu\widetilde{m})(k[(\textsc{Clock} \mid P)] \mid M)$	
$\xrightarrow{n.\texttt{open\_}k} n[\,\circ \mid (\nu\widetilde{m})(P \mid M)]$	(Open)
$(\nu\widetilde{m})(k[\textsc{Clock} \mid Q] \mid M), k \notin \widetilde{m}$	
$\xrightarrow{k.\texttt{tick}} (\nu\widetilde{m})(k[\textsc{Clock} \mid \texttt{tick} \mid Q] \mid M)$	(Tick)

Table 7: Rules for timed labeled transition systems, where in (Co-Enter) given $\textsc{Clock}_k = \textsc{Clock}\{c_k, (p_k, q_k), \{a_1, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}$ the updated clock is denoted by $\textsc{Clock}^*_k = \textsc{Clock}\{c_k, (p_k, q_k), \{a_1, \ldots, a_k, n\}, \{a_{k+1}, \ldots a_n\}\}$ as seen in Table 5.

Definition 7 (Timed labeled transitions)**.** *The observable steps $M \xrightarrow{\alpha} M'$ of the timed labeled transition semantics for timed systems is given by the rules of Table 7. For internal behavior, τ-steps are the result of reduction steps, i.e., $M \twoheadrightarrow M'$ implies $M \xrightarrow{\tau} M'$.*

The untimed labels, recording the system-environment interactions, i.e., ambient movements induced by the capabilities, coincide with the ones from the untimed case of mobile ambients [25].

In rules (Enter) and (Exit), an ambient k enters, respectively exits, from an ambient n provided by the environment. The rules (Enter Shh) and (Exit Shh) model the same behavior for ambients with private names. In rule (Co-Enter), an ambient n, provided by the environment, enters an ambient k of the process. In rule (Open), the environment provides an ambient n in which the ambient k of the process is opened. In rule (Tick), the transition $M \xrightarrow{k.\mathtt{tick}} M'$ expresses that the top-level ambient k of the system M receives one time slice `tick` from the source clock on the global level. Note that the post-configurations after the transitions contain the symbol $\circ$, which is used as placeholder variable. The labels, capturing interaction with the environment, carry partial information about the "data" exchanged with the environment. For example, label $k.\mathtt{enter}\_n$ carries information about the identity k of the ambient being entered, which is contained in the system, as well as about the identity of the one entering named n, which, before the step, is still part of the environment. If thus the enter-label conceptually indicates that some arbitrary ambient $n[R \mid \textsc{Clock}]$ enters the system as effect of executing the **in** n-capability, then the name n is mentioned as part of the label but its "body" $R \mid \textsc{Clock}$ is not. Later, when relating the respective actions of two systems via a notion of bisimulation, intuitively, if one system does a transition where $n[R \mid \textsc{Clock}]$ enters, the second system must be able to exhibit the same transition, i.e., have the "same" ambient entering without breaking their (bi)simulation relationship. In principle, though, the second system can simulate the first doing a step where an ambient $n[R]$ enters, with the body $S \equiv R \mid \textsc{Clock}$. To achieve that (without overburdening the labels by interpreting them up-to structural congruence $\equiv$), the definition will make use of the placeholder $\circ$ and requiring preservation of the relationship for all *instantiations* of the placeholders for both systems by the same body (cf. Definition 8 below). The substitution of the placeholder by a pair of process and it local clock is written as $P \bullet (\textsc{Clock} \mid Q)$ and defined as expected.

The reduction semantics of a process can be encoded in the labelled transition system, because a reduction step can be seen as an interaction with an empty context. We are interested in bisimulations that abstract from τ-actions and use the notion of *weak actions*; let $\Longrightarrow$ denote the reflexive and transitive closure of $\xrightarrow{\tau}$, let $\overset{\alpha}{\Rightarrow}$ denote $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$, and let $\overset{\hat{\alpha}}{\Rightarrow}$ denote $\Longrightarrow$ if $\alpha = \tau$ and $\overset{\alpha}{\Rightarrow}$ otherwise.

An example of a system consuming one parental `tick` and performing the subsequent τ-actions is given below:

Example 4 (Timed transition)**.** *Let us reconsider Example 2. After one time signal of the source clock, the virtually timed ambient a emits one resource, which is consumed by the request:*

$$a[P'] := \; a[\textsc{Clock}^{1,1} \mid ! \boldsymbol{open}\ request.wait\_for\_done.done[\boldsymbol{out}\ a.\boldsymbol{out}\ lbs]$$

$$\mid \mathit{wait\_for\_done}.\mathit{done}[\textbf{\textit{out}}\ a.\textbf{\textit{out}}\ \mathit{lbs}] \mid \textbf{\textit{consume}}.\mathit{done\_signal}].$$

After another time signal from the source clock and some internal τ steps the ambient named done can emerge, thus here it holds that:

$$\mathit{lbs}[\textsc{Clock}^{2,1} \mid \mathit{lb} \mid a[P'] \mid b] \xrightarrow{\mathit{lbs}.\mathtt{tick}} \mathit{lbs}[\textsc{Clock}^{2,1} \mid \mathtt{tick} \mid \mathit{lb} \mid a[P'] \mid b]$$
$$\Longrightarrow \mathit{lbs}[\textsc{Clock}^{2,1} \mid \mathit{lb} \mid a \mid b] \mid \mathit{done}[].$$

For virtually timed ambients from Table 4 and their labelled transition system, we now define the notion of weak timed bisimulation, where `tick` steps are counted among the observable transitions.

Definition 8 (Weak timed bisimulation)**.** *A symmetric relation $\mathcal{R}$ over timed systems is a* weak timed bisimulation *if $M \mathrel{\mathcal{R}} N$ and $M \xrightarrow{\alpha} M'$ implies:*

1. *If α is a non-anonymous label, then $N \overset{\hat{\alpha}}{\Rightarrow} N'$ for some N', such that $M'\bullet(\textsc{Clock} \mid P) \ \mathcal{R}\ N'\bullet(\textsc{Clock} \mid P)$ (for all clocks* Clock *and processes P).*
2. *For anonymous labels:*
   (a) *If $\alpha = *.\mathtt{enter\_}n$, then $N \mid n[\circ] \Longrightarrow N'$ for some N', such that $M'\bullet(\textsc{Clock} \mid P) \ \mathcal{R}\ N'\bullet(\textsc{Clock} \mid P)$ (for all clocks* Clock *and processes P).*
   (b) *If $\alpha = *.\mathtt{exit\_}n$, then $n[\ \circ \mid N] \Longrightarrow N'$ for some N', such that $M'\bullet(\textsc{Clock} \mid P) \ \mathcal{R}\ N'\bullet(\textsc{Clock} \mid P)$ (for all clocks* Clock *and processes P).*

Systems M and N are *weakly timed bisimilar*, written $M \approx^t N$, if $M \mathrel{\mathcal{R}} N$ for some weak timed bisimulation $\mathcal{R}$. If two systems are weakly timed bisimilar in a timed setting where we observe the ticking of the source clock, then it follows from the definition of weak timed bisimulation that they are weakly bisimilar in a setting where we do not observe the ticking of the clocks but interpret all `tick`-actions as τ-actions, instead. Since strong versions of bisimulation tend to be less useful [34], we mainly consider weak bisimulation and let unqualified terms such as bisimulation and bisimilarity refer to these weak versions.

Lemma 1 (Consistency)**.** *$M \approx^t N$ implies that M and N are weakly bisimilar, $M \approx N$.*

Note that for virtually timed ambients which are not time consuming, i.e. with a speed of $(0, 0)$, weak timed bisimulation and weak bisimulation coincide.

The following example illustrates that systems can be weakly bisimilar in a setting where time is not observed without being weakly timed bisimilar in a timed setting.

Example 5 (Comparing systems with load balancers)**.** *We compare the behavior of the system lbs from Example 1, which we will now call N, with a second system called M, which is defined as follows:*

load balancer system M: $(\nu\ lb, a)\ s[\text{CLOCK}^{1,1} \mid lb \mid a]$
incoming request: $request[P.done\_signal \mid \boldsymbol{in}\ s.enter\_signal.\boldsymbol{open}\ move]$
load balancer: $lb[\ !wait\_\ for\ \_enter.move[\boldsymbol{out}\ lb.\boldsymbol{in}\ request.\boldsymbol{in}\ a]]$
ambient a: $a[\text{CLOCK}^{2,1}\ |!\boldsymbol{open}\ request.wait\_\ for\ \_done.done[\boldsymbol{out}\ a.\boldsymbol{out}\ s]]$

In contrast to N, system M only contains one virtually timed ambient, which receives all requests. If we do not observe time, the systems behave the same, as they both answer requests by emitting an observable done-signal. However, the systems are not weakly timed bisimilar:

$$N \approx M \quad \textit{and} \quad N \not\approx^t M\ .$$

We will revisit the example in the next section, after having defined the notion of contexts (cf. Example 6 below).

4.2. Reduction Barbed Congruence

The purpose of this section is to establish that weak timed bisimulation $\approx^t$ is a congruence. The result is established indirectly via a second equivalence, reduction barb congruence (cf. Definition 14 below). This relation is defined as the largest relation which is preserved by all constructs of the language (and thereby a congruence by definition), preserved by the internal steps of the reduction semantics, and finally preserved by so called barbs, which are simple observables of terms. Honda and Yoshida's method [19] can be used to define weak reduction barbed congruence for mobile ambients [25]. This approach can be extended to virtually timed ambients.

Definition 9 (Contexts)**.** *A* context *is a process with a hole. A* system context *is a context that transforms systems into systems. System contexts are generated by the following grammar:*

$$\begin{array}{rcl} \mathcal{C}[-] & ::= & - \quad | \quad \mathcal{C}[-] \mid M \quad | \quad M \mid \mathcal{C}[-] \\ & | & (\nu n)\mathcal{C}[-] \quad | \quad n[\mathcal{C}[-] \mid P] \quad | \quad n[P \mid \mathcal{C}[-]] \ , \end{array}$$

where M is an arbitrary system and P is an arbitrary process.

Example 6 (Comparing systems with load balancers (2))**.** *Revisiting Example 5, we use the notion of contexts to show that $N \approx M$ and $N \not\approx^t M$. The argument follows a so-called* up-to proof technique *(e.g., [34]). There are two parts to establish.*

1. *For the untimed equality $N \approx M$, we define a relation $\mathcal{R}$ as follows:*

$$\begin{array}{rl} \mathcal{R} = \{(M_1, M_2) \mid & M_1 = (\ (\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R]\ , \\ & M_2 = (\nu\ lb, a)\ s[lb \mid a \mid R]), \textit{for arbitrary } s, R \\ & \textit{s.t. } lb,\ a, \textit{ and } b \notin \mathit{fn}(R)\ \}\ . \end{array}$$

   We need to confirm then, that $\mathcal{R}$ is a bisimulation up to context and up to structural congruence. So, assume $N \xrightarrow{\alpha} N'$ and proceed by case analysis on the structure of α.
   Case: $\alpha = \tau$ (internal action).
   Depending on the nature of N', there are three subcases to consider.
   Subcase: $N' \equiv (\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R']$,
   with $R \xrightarrow{\tau} R'$. It follows that $(\nu\ lb, a)\ s[lb \mid a \mid R] \xrightarrow{\tau} M'$, where $M' \equiv s(\nu\ lb, a)\ [lb \mid a \mid R']$ and $N' \equiv \mathcal{R} \equiv M'$
   Subcase: $N' \equiv (\nu n)(\nu\ lb, a, b)\ (m[R'] \mid s[lb \mid a \mid b \mid R''])$,
   with $R \equiv (\nu n)(m[\textbf{out}\ s \mid R'] \mid R'')$. It follows that $(\nu\ lb, a)\ s[lb \mid a \mid R] \xrightarrow{\tau} M'$, where $M' \equiv (\nu n)(m[R'] \mid (\nu\ lb, a)\ s[lb \mid a \mid R''])$. Now we can factor out the system context $C[-] = (\nu n)(m[R'] \mid -)$, thus we are still in $\mathcal{R}$ up to context and up to structural congruence.
   Subcase: $N' \equiv (\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R] \mid done[]$.
   *This must have been derived from $(\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R \mid request[P]]$, for an adequate P. This in turn is an instantiation of $s.$`enter`*_request. It holds that $(\nu\ lb, a)\ s[lb \mid a \mid R \mid request[P]]$ reduces to $(\nu\ lb, a)\ s[lb \mid a \mid R] \mid done[]$. Now we can factor out the system context $C[-] = done[] \mid -$, thus we are still in $\mathcal{R}$ up to context and up to structural congruence.*

Case: $\alpha = s.\mathtt{enter\_}n$.
Then $N' \equiv n[(\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R'] \mid \circ]$ *and* R *must have execited the capability* ***in*** n*, reducing to* R'*. Thus,* $(\nu\ lb, a)\ s[lb \mid a \mid R] \xrightarrow{s.\mathtt{enter\_}n} M'$*, where* $M' \equiv n[(\nu lb, a)\ s[lb \mid a \mid R'] \mid \circ]$*. Now we can factor out the system context* $C[-] = n[\circ \mid -]$*, thus we are still in* $\mathcal{R}$ *up to context and up to structural congruence.*

Case: $\alpha = s.\mathtt{exit\_}n$.
Then $N' \equiv (\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R] \mid n[\circ]$ *and* R *must have unleashed the capability* ***out*** n*, reducing to* R'*. Thus,* $(\nu\ lb, a)\ s[lb \mid a \mid R] \xrightarrow{s.\mathtt{exit\_}n} M'$*, where* $M' \equiv (\nu\ lb, a)\ s[lb \mid a \mid R'] \mid n[\circ]$*. Now we can factor out the system context* $C[-] = n[\circ] \mid -$*, thus we are still in* $\mathcal{R}$ *up to context and up to structural congruence.*

Case: $\alpha = n.\mathtt{open\_}s$.
Then $N' \equiv (\nu\ lb, a, b)\ n[lb \mid a \mid b \mid R \mid \circ]$ *and* $(\nu\ lb, a)\ s[lb \mid a \mid R] \xrightarrow{n.open\_s} M'$*, where* $M' \equiv (\nu\ lb, a)\ n[lb \mid a \mid R \mid \circ]$*. By the definition of* $\mathcal{R}$ *it holds that* $N' \bullet (\textsc{Clock} \mid P) \equiv \mathcal{R} \equiv M' \bullet (\textsc{Clock} \mid P)$ *for all clocks* $\textsc{Clock}$ *and processes* P*.*

Case: $\alpha = s.\overline{\mathtt{enter}}\mathtt{\_}k$.
Then $N' \equiv (\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R \mid n[\circ]]$ *and* $(\nu\ lb, a)\ s[lb \mid a \mid R] \xrightarrow{s.\overline{\mathtt{enter}}\_k} M'$*, where* $M' \equiv (\nu\ lb, a)\ s[lb \mid a \mid R \mid n[\circ]]$*. By the definition of* $\mathcal{R}$ *it holds that* $N' \bullet (\textsc{Clock} \mid P) \equiv \mathcal{R} \equiv M' \bullet (\textsc{Clock} \mid P)$ *for all clocks* $\textsc{Clock}$ *and processes* P*.*
It follows that $N \approx M$*.*

2. *For the timed setting we need to show* $N \not\approx^t M$*. Assume for a contradiction that* $N \approx^t M$*. Consider the case* $\alpha = \tau$ *with the subcase* $M' \equiv M'' \mid done[]$*. This must have been derived from* $(\nu\ lb, a)\ s[lb \mid a \mid R \mid P]$*, for an adequate* P*. This in turn is an instantiation of* $s.\overline{\mathtt{enter}}\_n$*. Let* $P = request[\textbf{\textit{consume}}.\textbf{\textit{consume}}.done\_signal \mid enter\_signal.\textbf{\textit{open}}\ move]$*. Then it holds that* $(\nu\ lb, a)\ s[lb \mid a \mid R \mid P] \xRightarrow{s.\mathtt{tick}} (\nu\ lb, a)\ s[lb \mid a \mid R] \mid done[]$*. As* $\approx^t$ *is a weak timed bisimulation it has to hold that* $(\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R \mid P] \xRightarrow{s.\mathtt{tick}} (\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R] \mid done[]$ *for this process* P *as well. However, in the case of system* N *the reduction result* $(\nu\ lb, a, b)\ s[lb \mid a \mid b \mid R] \mid done[]$ *can not be derived with only one execution of* `tick`*. As clocks and resources are restricted to their parental ambients no additional* `tick` *can be added via contexts, and it holds that* $N \xRightarrow{s.\mathtt{tick}} N'$*, where* $N \equiv (\nu\ lb, a, b)\ s[lb \mid a[P'] \mid b \mid R]$*.*

Thus, $M' \approx^t N'$ does not hold.

Definition 10 (Preservation under contexts). *A relation $\mathcal{R}$ is* preserved by system contexts *if $M \mathcal{R} N$ implies $\mathcal{C}[M] \mathcal{R} \mathcal{C}[N]$ for all system contexts $\mathcal{C}[-]$.*

Theorem 1. *Weak timed bisimilarity is preserved by system contexts.*

Proof. The proof is similar to the related proof for mobile ambients by Merro and Zappa Nardelli [25], and extended for the $k.\texttt{tick}$ action. As $k.\texttt{tick}$ be seen as a special case of the $k.\overline{\texttt{enter}}\_n$ action, the same proof method can be used here. □

We extend the notion of barbs from Definition 2 by a weak version.

Definition 11 (Weak barbs). *Process P weakly barbs on n, $P{\Downarrow_n}$ if $P \Longrightarrow P'$ and $P'{\downarrow_n}$, for some P' .*

Besides being preserved by contexts, the congruence definition stipulates preservation under basic observations.

Definition 12 (Preservation of barb). *A relation $\mathcal{R}$ over processes is* barb preserving *if $P \;\mathcal{R}\; Q$ and $P{\downarrow_n}$ implies $Q{\Downarrow_n}$.*

Definition 13 (Preservation under reduction). *A relation is* reduction closed *(or preserved under reduction) if $P \mathcal{R} Q$ and $P \rightarrow P'$, implies $Q \rightarrow Q'$ for some Q'*

Definition 14 (Reduction barbed congruence over timed systems). Reduction barbed congruence over timed systems $\simeq_s$ *is the largest symmetrical relation over timed systems which is preserved by all system contexts, is reduction closed and barb preserving.*

To show that reduction barbed congruence for timed systems and weak timed bisimilarity coincide, the more challenging direction to establish is the inclusion $\simeq_s \subseteq \approx^t$. Contrapositively formulated it means any two open systems distinguishable by $\approx^t$, are also distinguishable via $\simeq_s$, i.e., via a context which makes this distinction. To do this we need to define a system context that (in particular) observes the action $k.\texttt{tick}$; contexts to observe the other non-anonymous actions of the labeled transition system are defined as for mobile ambients [25]. Again, we can consider $k.\texttt{tick}$ as a special case of the $k.\overline{\texttt{enter}}\_n$ action and can therefore define the tick-observing context as follows:

$$\mathcal{C}_{n.\texttt{tick}}[-] = (\nu a, b)a[\textbf{in}\ n.\texttt{tick}[\textbf{out}\ a.b[\textbf{out}\ \texttt{tick}.\textbf{out}\ n.done[\textbf{out}\ b]]]] \mid - \ .$$

It remains to show that these contexts can indeed be used to observe ticks (respectively other actions for correspondingly defined contexts). This proof obligation has two sides, the first captured in Lemma 2. It stipulates that the tick-observing context put together with a tick-emitting system does observe the tick in that both reduce to a configuration containing the ambient *done* at top-level; the latter is used as marker to witness this successful interaction. The reverse property in Lemma 3 makes sure that, with the help of barbs, a context $\mathcal{C}_{n.\texttt{tick}}$ (or $\mathcal{C}_\alpha$ in the general cases) assures that the system under observation does make a $\texttt{tick}$-step (resp. α-step).

Note that all top level ambients of the system which is entered in the context will receive time via τ-actions. With this context, we can extend Lemma 4.8 of Merro and Zappa Nardelli [25] to virtually timed ambients as follows:

Lemma 2. *Let α be an observable, non-anonymous label and M a system. Then for all clocks* $\textsc{Clock}$ *and processes P, if $M \xrightarrow{\alpha} M'$ then $\mathcal{C}_\alpha[M] \bullet (\textsc{Clock} \mid P) \Longrightarrow \simeq_s (M' \bullet (\textsc{Clock} \mid P)) \mid done[]$.*

Proof. We will only consider $\alpha = n.\texttt{tick}$. All other cases are similar to the ones for untimed mobile ambients. Let P be a process. We know that $M \xrightarrow{n.\texttt{tick}} M'$. Then $M' \equiv n[\texttt{tick} \mid Q] \mid M''$. Now,

$$\begin{aligned}
&\mathcal{C}_{n.\texttt{tick}}[M] \bullet (\textsc{Clock} \mid P)\\
\equiv&((\nu a, b)a[\textbf{in}\ n.\texttt{tick}[\textbf{out}\ a.b[\textbf{out}\ \texttt{tick}.\textbf{out}\ n.done[\textbf{out}\ b]])]] \mid M)\\
&\bullet (\textsc{Clock} \mid P)\\
\rightarrowtriangle&((\nu a, b)n[a[\texttt{tick}[\textbf{out}\ a.b[\textbf{out}\ \texttt{tick}.\textbf{out}\ n.done[\textbf{out}\ b]]]] \mid Q] \mid M'')\\
&\bullet (\textsc{Clock} \mid P)\\
\rightarrowtriangle&((\nu a, b)n[a[] \mid \texttt{tick}[b[\textbf{out}\ \texttt{tick}.\textbf{out}\ n.done[\textbf{out}\ b]]] \mid Q] \mid M'')\\
&\bullet (\textsc{Clock} \mid P)\\
\rightarrowtriangle&((\nu a, b)n[a[] \mid \texttt{tick} \mid Q] \mid b[done[\textbf{out}\ b]] \mid M'') \bullet (\textsc{Clock} \mid P)\\
\rightarrowtriangle&((\nu a, b)n[a[] \mid \texttt{tick} \mid Q] \mid b[] \mid done[] \mid M'') \bullet (\textsc{Clock} \mid P)\\
\equiv&(M' \bullet (\textsc{Clock} \mid P)) \mid done[]. \qquad \square
\end{aligned}$$

To prove the correspondence between actions α and their contexts $\mathcal{C}_\alpha[-]$, we have to prove the converse of the above lemma as well. The proof of this result uses particular contexts $\mathtt{spy}_\alpha\langle i, j, -\rangle$ as a technical tool to guarantee that the process P provided by the environment does not perform any action. Define the context $\mathtt{spy}_{n.\mathtt{tick}}\langle i, j, -\rangle := (i[] \mid -) \oplus (j[] \mid -)$, where $\oplus$ represents (an encoding of) the internal choice between the left and right process. We can now extend Lemma 4.12 of [25] as follows:

Lemma 3. *Let α be an observable, non-anonymous action and M a system. Let i and j be fresh names for M. For all processes P with $\{i, j\} \cap fn(P) = \emptyset$, if $C_\alpha[M] \bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i, j, P\rangle) \Longrightarrow\equiv N \mid done[]$ and $N \Downarrow_i$ and $N \Downarrow_j$, then there exists a system M' such that $M \xrightarrow{\alpha} M'$ and $M' \bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i, j, P\rangle) \simeq_s N$.*

Proof. We only show the case of $\alpha = n.\mathtt{tick}$. All other cases are similar to the cases for mobile ambients.

$$\begin{aligned}
&\mathcal{C}_\alpha[M] \bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i, j, P\rangle) \\
&\equiv (\nu a, b)a[\mathbf{in}\ n.\mathtt{tick}[\mathbf{out}\ a.b[\mathbf{out}\ \mathtt{tick}.\mathbf{out}\ n.done[\mathbf{out}\ b]]]] \mid \\
&\quad M \bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i, j, P\rangle) \\
&\twoheadrightarrow (\nu a, b)n[a[] \mid \mathtt{tick} \mid Q] \mid b[] \mid M'' \\
&\quad \bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i, j, P\rangle) \mid done[] \bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i, j, P\rangle) \\
&\equiv M' \bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i, j, P\rangle) \mid done[] \bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i, j, P\rangle) \\
&\equiv N \mid done[]\ .
\end{aligned}$$

We conclude that $M \xrightarrow{n.\mathtt{tick}} M'$. Thus, $M' \bullet (\textsc{Clock} \mid \mathtt{spy}_\alpha\langle i, j, P\rangle) \simeq_s N$ holds. □

Theorem 2. *Weak timed bisimilarity and reduction barbed congruence over timed systems coincide.*

Proof. By Theorem 1, $\approx^t$ is preserved by contexts. The relation is additionally preserved under barbs and under reductions, two properties which carry over from (untimed) mobile ambients. With $\simeq_s$ defined as the *largest* congruence enjoying those preservation properties, immediately $\approx^t \subseteq \simeq_s$. For the reverse inclusion, we can now extend Theorem 4.14 in [25] by means of Lemmas 2 and 3 above. The cases for the anonymous actions are the same as for the mobile ambients. Hence, reduction barbed congruence and weak timed bisimilarity coincide, i.e. $M \approx^t N$ iff $M \simeq_s N$. □

5. Relaxation over Time

It is possible that two virtually timed systems behave the same for a certain type of requests but not for others, or that a system behaves equivalently to another except for needing some extra time slices per executed request. Example 5 shows that the timed behavior of a system depends on the kind of requests that are made, the number of virtually timed ambients which are used as computing environments, their speeds and the way the requests are distributed to the computing environments.

5.1. Bounded Bisimulation

To compare systems only for certain types of requests and to consider if one system is faster or slower in executing these requests, we define a bisimulation which does not consider all processes P but only specific ones and also relaxes the time condition. First we introduce an asymmetric order relation, a *simulation* which allows to compare systems of different "speed", but otherwise equivalent behavior.

Definition 15 (Weak k-timed $\mathcal{P}$-simulation)**.** *Let $\mathcal{P}$ be a class of processes and $k \in \mathbb{N}$. A relation $\mathcal{R}$ over systems is a* weak k-timed $\mathcal{P}$-simulation *if $M \;\mathcal{R}\; N$ and $M \xrightarrow{\alpha} M'$ implies*

1. *If α is a non-anonymous label, then $N \xrightarrow{m.\mathtt{tick}^i}\xrightarrow{\hat{\alpha}}\xrightarrow{m.\mathtt{tick}^j} N'$ for some N' and where $i + j \leq k$, such that $M'\bullet(\textsc{Clock} \mid P) \;\; \mathcal{R} \;\; N'\bullet(\textsc{Clock} \mid P)$ (for all $P \in \mathcal{P}$ and all $\textsc{Clock}$).*
2. *For anonymous labels:*
   (a) *If $\alpha = *.\mathtt{enter\_}n$, then $N \mid n[\circ] \xrightarrow{m.\mathtt{tick}^j} N'$ for some N' and were $j \leq k$. such that $M'\bullet(\textsc{Clock} \mid P) \;\; \mathcal{R} \;\; N'\bullet(\textsc{Clock} \mid P)$, (for all $P \in \mathcal{P}$ and all $\textsc{Clock}$).*
   (b) *If $\alpha = *.\mathtt{exit\_}n$, then $n[\;\circ \mid N] \xrightarrow{m.\mathtt{tick}^j} N'$ for some N' and were $j \leq k$. such that $M'\bullet(\textsc{Clock} \mid P) \;\; \mathcal{R} \;\; N'\bullet(\textsc{Clock} \mid P)$, (for all $P \in \mathcal{P}$ and all $\textsc{Clock}$).*

Here $m.\mathtt{tick}^j$ denotes the j-fold execution of the $m.\mathtt{tick}$ action. A system M is *weakly k-timed $\mathcal{P}$-simulated* by N, written $M \preceq^t_{k,\mathcal{P}} N$, if $M \;\mathcal{R}\; N$ for some k-timed $\mathcal{P}$-simulation and some class of processes $\mathcal{P}$. Let k_{min} be the minimum k such that $M \preceq^t_{k,\mathcal{P}} N$ holds. This means that N needs at least k_{min} more tick steps than M to execute single requests of type $\mathcal{P}$. We will consider all further k to be the minimal k_{min} and call k the *slack* of the simulation.

Definition 16 (Weak k-timed $\mathcal{P}$-bisimulation). *Let $\mathcal{P}$ be a class of processes and $k \in \mathbb{N}$. A symmetric weak k-timed $\mathcal{P}$-simulation $\mathcal{R}$ over systems is a* weak k-timed $\mathcal{P}$-bisimulation.

Systems M and N are *weakly k-timed $\mathcal{P}$-bisimilar*, written $M \approx^t_{k,\mathcal{P}} N$, if $M \mathrel{\mathcal{R}} N$ for some k-timed $\mathcal{P}$-bisimulation $\mathcal{R}$ and some class of processes $\mathcal{P}$. This means that the systems need at least k additional tick steps to simulate each other's behavior. If $M \approx^t_{0,\mathcal{P}} N$ for a class of processes $\mathcal{P}$ then the systems are not distinguishable for these processes. Observe that if $M \approx^t_{0,\mathcal{P}} N$ for $\mathcal{P}$ the class of all processes then $M \approx^t N$. This follows from the definitions. We can make a similar observation about weak bisimulation without time.

Lemma 4. *Let $\mathcal{P}$ be the class of all processes. Then $M \approx^t_{\infty,\mathcal{P}} N$ if and only if $M \approx N$.*

Proof. Assume $M \approx N$. This means, we consider a setting where we do not observe the ticking of the clocks but interpret all `tick`-actions as τ-actions, instead. Treating all `tick`-actions as τ-actions is equivalent to setting $k = \infty$ in the definition of k-timed $\mathcal{P}$-bisimilarity. Thus, $M \approx^t_{\infty,\mathcal{P}} N$ for $\mathcal{P}$ the class of all processes.

Assume now $M \approx^t_{\infty,\mathcal{P}} N$ for $\mathcal{P}$ the class of all processes. This is equivalent to treating `tick`-actions as unobservable τ-actions, thus $M \approx N$ in a setting where we do not observe the ticking of the clocks but interpret all `tick`-actions as τ-actions. □

We illustrate the concepts of weakly k-timed $\mathcal{P}$-bisimulation and slack in the following example.

Example 7 (Slack between two systems). *Consider the two systems M and N from Example 5. System M has a clock with speed $(1,1)$, a load balancer and the computing environment $a[\textsc{Clock}^{2,1} \mid \cdots]$, and further system N has a clock with speed $(2,1)$, another load balancer and the computing environments $a[\textsc{Clock}^{1,1} \mid \cdots]$ and $b[\textsc{Clock}^{1,1} \mid \cdots]$. As M has only one virtually timed subambient, which is used as a computing environment, with a speed that equals the combined speeds in N, it holds that M can always simulate N; thus, $N \preceq^t_{0,\mathcal{P}} M$ for all $\mathcal{P}$. On the other hand the number of additional tick steps needed by N to simulate M depends on the load balancer in the system. If we assume that the load balancing is round-robin then the two systems are not distinguishable for processes which only need one computing resource to*

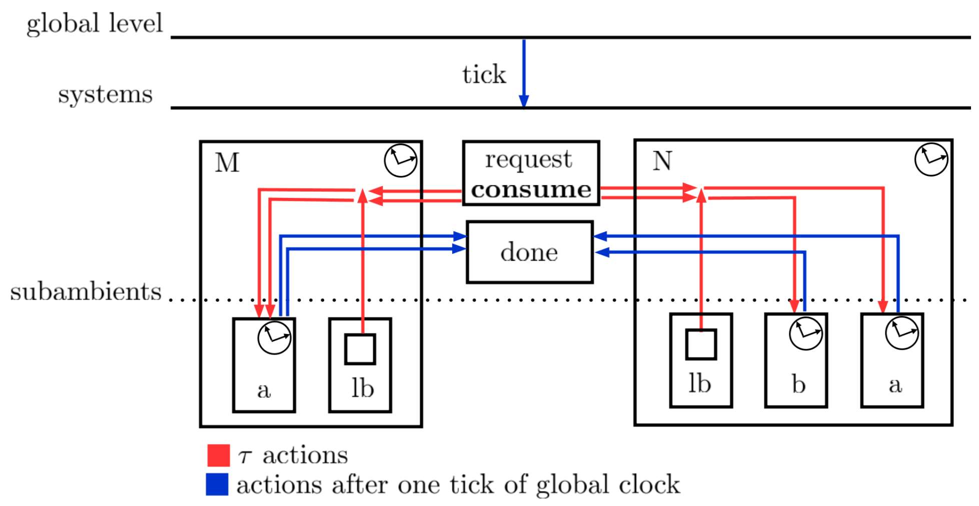


Figure 1: Indistinguishable behavior with round-robin load balancers and requests which need one computing resource to execute.

execute. An example can be seen in Figure 1. Thus, $M \approx^t_{0,\mathcal{P}} N$ for the class $\mathcal{P}$ of processes which need only one computing resource to execute. Since tasks which need more than one resource can not be served by N in one time slice it is easy to see that M will always be faster. This effect can be seen in Figure 2.

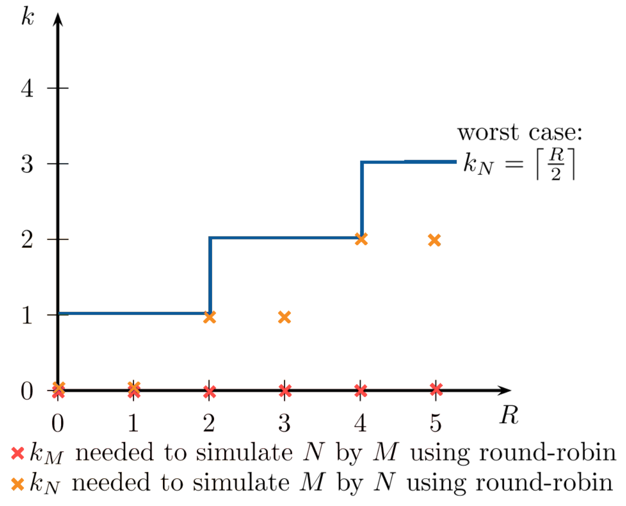


Figure 2: Worst case function for k_N as well as the actual k needed when using round-robin, where R is the maximal number of resources required by a single request.

5.2. Comparing Different Schedulers

In Example 7, round-robin is the best way to schedule the processes for distribution into the virtually timed subambients. But in general round-robin does not guarantee the fastest execution.

Definition 17. *We call the scheduling of processes* perfect *if the processes are always distributed in such a way that the execution takes the least possible amount of time.*

Note that building such a load balancer is a non-trivial optimisation problem. We use this definition to generalize the upper bound for the slack k_N in Example 7. Let $\mathcal{P}_{\varnothing}$ denote the class of all processes which do not require resources and therefore are not time consuming. If $M \approx^t_{0,\mathcal{P}_{\varnothing}} N$, then the systems do not have any internal processes which require a different amount of `tick` steps. Additionally we use the notion of a *statically deployed system.* In a statically deployed system, as for instance in Example 1, the time consuming virtually timed ambients do not move.

Definition 18. *A* statically deployed system *is a timed system where only virtually timed ambients with a speed of* $(0, 0)$ *are permitted to make use of the enter- and exit-capabilities.*

Lemma 5. *Let M and N be statically deployed systems. Assume $M \approx N$ and $M \approx^t_{0,\mathcal{P}_{\varnothing}} N$. Let (a_m, b_m) be the accumulated speed in a virtually timed ambient m, which is used as computing environment. Let $P \in \mathcal{P}$ be a parallel composition of independent requests $p_1, \ldots, p_n$, $n \in \mathbb{N}$, which require a number of computing resources r_{p_i}, $i \in \{1, \ldots n\}$, and result in observable changes to the system. Let $R = \max\{\sum_{i=1}^n r_{p_i} \mid P \in \mathcal{P}\}$ be the maximum of all required computing resources by any request. Then it holds that $M \preceq^t_{k,\mathcal{P}} N$ for some $\mathcal{P}$ and it holds that*

$$k \leq \left\lceil \frac{R - \left\lceil \frac{R}{\sum_{m \in M} \frac{a_m}{b_m}} \right\rceil \cdot \min\{\frac{a_m}{b_m} \mid m \in N\}}{\min\{\frac{a_m}{b_m} \mid m \in N\}} \right\rceil .$$

Proof. As $M \approx N$, it holds that $M \preceq^t_{\infty,\mathcal{P}} N$ for all $\mathcal{P}$. As $M \approx^t_{0,\mathcal{P}_{\varnothing}} N$, there are no internal processes which influence the timing in M and N in different ways. As the scheduling functions of the systems are not known, the worst case scenario in terms of speed occurs if all requests end up in the slowest virtually timed subambient, which is used as a computing environment, in N, while all virtually timed ambients in M can be used perfectly. In this case the number of time steps needed by M to execute all requests is $\left\lceil \frac{R}{\sum_{m\in M}\frac{a_m}{b_m}} \right\rceil$. By the definition of k-timed $\mathcal{P}$-simulation, N can execute k time steps after a process $P \in \mathcal{P}$ has entered the system. To execute all requests in the slowest virtually timed ambient, it has to hold that $R \leq \left(\left\lceil \frac{R}{\sum_{m\in M}\frac{a_m}{b_m}} \right\rceil + k \right) \cdot \min\{\frac{a_m}{b_m} \mid m \in N\}$. Thus, the minimal k in the worst case scenario is

$$k = \left\lceil \frac{R - \left\lceil \frac{R}{\sum_{m\in M}\frac{a_m}{b_m}} \right\rceil \cdot \min\{\frac{a_m}{b_m} \mid m \in N\}}{\min\{\frac{a_m}{b_m} \mid m \in N\}} \right\rceil .$$

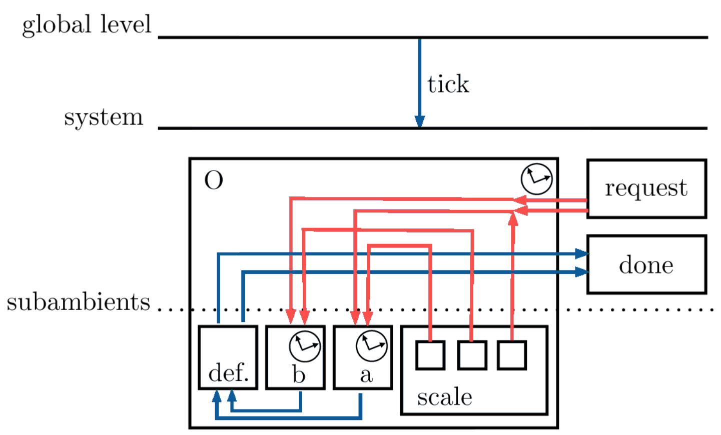
As k can be less for other scheduling functions, the inequation follows. □

Note that if M and N are not statically deployed, a similar estimation can be made by considering the minimum of all possible accumulated speeds during the computation. However, such an estimation is much less accurate.

In the following example, we consider a load balancing strategy which is different from round-robin.

Example 8 (System with elastic scaling)**.** *We modify Example 5 of systems with load balancers by considering an elastically scaling system O which is able to react to the size of the requests.*

$$\begin{aligned}
\textit{system O:}\ & (\nu\ \textit{scale}, \textit{default}, a, b)\ \textit{ess}[\text{CLOCK}^{1,1} \mid \textit{scale} \mid \textit{default}] \\
\textit{request:}\ & \textit{request}[P.\textit{done\_signal} \mid \textbf{\textit{in}}\ \textit{ess}.\textit{size\_signal}.\textbf{\textit{open}}\ \textit{move}] \\
\textit{scale:}\ & \textit{scale}[!(\textit{size\_signal\_a}.\textbf{\textit{open}}\ \textit{lock}_a.a[\cdots] \mid \\
& \quad \textit{size\_signal\_b}.\textbf{\textit{open}}\ \textit{lock}_b.b[\cdots]) \\
& \quad !\textit{lock}_a[x[]] \mid !\textit{lock}_b[y[]] \mid \\
& \quad !(\textbf{\textit{open}}\ x.\textit{move}[\textbf{\textit{out}}\ \textit{scale}.\textbf{\textit{in}}\ \textit{request}.\textbf{\textit{in}}\ a] \mid \\
& \quad\ \textbf{\textit{open}}\ y.\textit{move}[\textbf{\textit{out}}\ \textit{scale}.\textbf{\textit{in}}\ \textit{request}.\textbf{\textit{in}}\ b])] \\
\textit{ambient default:}\ & \textit{default}[!\textbf{\textit{open}}\ a \mid !\textbf{\textit{open}}\ b] \\
\textit{ambient a:}\ & a[\text{CLOCK}^{1,1} \mid \textbf{\textit{out}}\ \textit{scale}.\textbf{\textit{open}}\ \textit{request}.\textit{wait\_for\_done}. \\
& \textbf{\textit{in}}\ \textit{default} \mid \textit{done}[\textbf{\textit{out}}\ \textit{default}.\textbf{\textit{out}}\ \textit{ess}]] \\
\textit{ambient b:}\ & b[\text{CLOCK}^{2,1} \mid \textbf{\textit{out}}\ \textit{scale}.\textbf{\textit{open}}\ \textit{request}.\textit{wait\_for\_done}. \\
& \textbf{\textit{in}}\ \textit{default} \mid \textit{done}[\textbf{\textit{out}}\ \textit{default}.\textbf{\textit{out}}\ \textit{ess}]]
\end{aligned}$$



🟥 τ actions
🟦 actions after first tick of global clock

Figure 3: Example of a scaling system

In this example the ambient scale receives a signal from the request indicating its size. According to that scale releases ambients a or b with different speeds. The request is executed in this ambient and afterwards the ambient is deleted.

Figure 3 exemplifies the behavior of the system when given the requests

r_1*: request*$[$***consume.consume****.done_signal* $|$
in *ess.size_signal_b.****open*** *move*$]$ *and*

r_2*: request*$[$***consume****.done_signal* $|$ ***in*** *ess.size_signal_a.****open*** *move*$]$.

The given system shows the same behavior as the systems with load balancer in Example 5. At first glance the timed behavior seems similar to system M*, but as this system can react to the size of the input while* M *uses a round-robin approach, it holds that* $M \preceq^t_{0,\mathcal{P}} O$ *for all* $\mathcal{P}$*, while* $O \preceq^t_{k,\mathcal{P}} M$ *for some* $k \geq 0$*.*

If two systems release the same combined amount of resources in their virtually timed subambients for the same combined input value of time slices, we say they have the same *combined speed*. We can show that systems with the same combined speed are not distinguishable for small execution requests and perfect scheduling functions.

Lemma 6. *Assume* $M \approx N$ *where* M *and* N *do not have any internal processes which require a different amount of time steps. Assume the schedul-*

ing functions in M and N are perfect and M and N have the same combined speed in the virtually timed subambients which are used as computing environments. Let $P \in \mathcal{P}$ be a parallel composition of independent requests $p_1, \dots, p_n$, $n \in \mathbb{N}$, which require a number of computing resources r_{p_i}, $i \in \{1, \dots n\}$, and result in observable changes to the system. Then $M \approx^t_{0,\mathcal{P}} N$ if $\mathcal{P}$ is such that for all $P \in \mathcal{P}$ it holds that

$$\max\{r_{p_i} \mid i \in \{1, \dots n\}\} \leq 1 \ ,$$

i.e. the maximal requirement of resources in one request is less or equal one.

Proof. As the requests are at most of size one and are scheduled perfectly and both systems have the same combined speed, they will need the same number of time steps. □

In the following example we examine the worst case estimation of Lemma 5 with two different scheduling functions.

Example 9 (Worst case estimations)**.** *Consider two systems M and N, which are build similar to the system in Example 1. System M has a clock with speed $(2, 1)$, some load balancer and the ambients $a[\text{CLOCK}^{4,1} \mid \cdots]$ and $b[\text{CLOCK}^{2,1} \mid \cdots]$ for execution. System N has a clock with speed $(2, 1)$,*

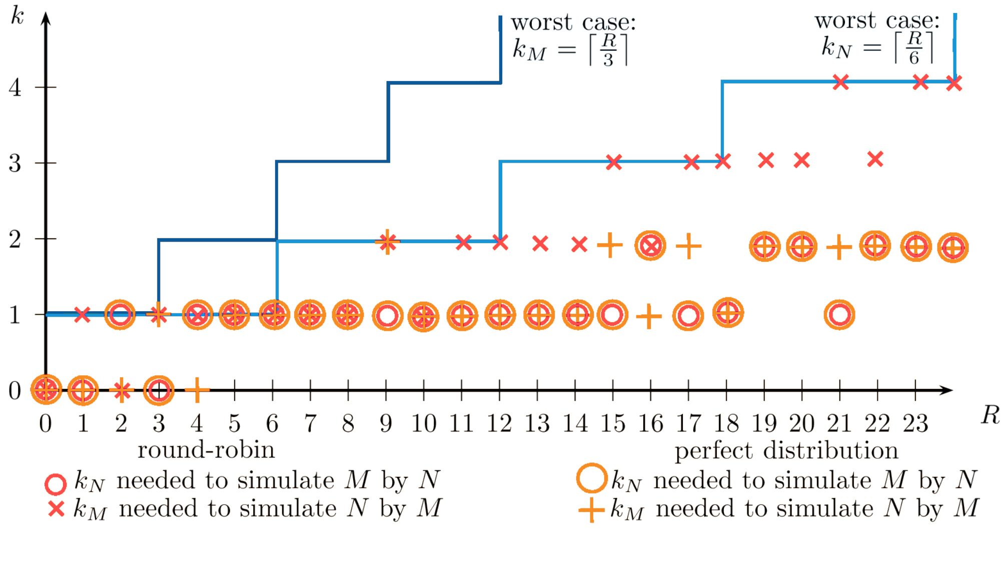


Figure 4: Worst case function and actual k needed by M and N in the case of round-robin and perfect scheduling, respectively.

some load balancer and the computing environments $a[\textsc{Clock}^{3,1} \mid \cdots]$ *and* $a[\textsc{Clock}^{3,1} \mid \cdots]$*. As M and N have the same combined speed by Lemma 6, the systems are not distinguishable for perfect scheduling and requests of maximal size one. With round-robin scheduling , M needs $k = 1$ to simulate N for requests of size one. Figure 4 shows the worst case function as well as the actual slack k needed by the systems to simulate each other for different given scheduling functions and request sizes. The bigger the requests get, the worse the behavior of M with round-robin scheduling becomes. This is due to M containing the virtually timed subambient with the slowest local clock. Additionally we see that the slack k for perfect scheduling stays far behind the worst case estimation.*

6. Related Work

The extension of algebraic concurrency theories such as ACP, CCS and CSP to deal with time-dependent behavior can be done according to different design choices. Time can be absolute or relative. The time domain can be continuous, dense (so-called real time), or discrete. Time can be semantically associated with regular actions, for instance via the use of timers, or it can be represented by special actions. Timed process algebras which originated from ACP and CSP can be found in, e.g., [3, 31, 4]. The model of virtually timed ambients developed in this paper builds on the mobile ambients. Therefore, we focus the discussion of related work on the π-calculus [34], which originated from CCS and which is closely related to the ambient calculus.

An early timed extension of CCS introduced a special action for time, without committing to a discrete or continuous time domain [28]. A related idling action σ is proposed in [17] such that processes in a standard process algebra would need exactly one time unit to process a σ, where time is discrete and processes synchronized via a global clock. A notion of local time for CCS is proposed in [35]; this notion resembles our model of local clocks, but was realized in terms of a timeout oriented model. A simulation-based faster-than preorder is introduced in [23]; this preorder is related to our notion of time relaxation but the faster-than preorder allows a process to delay by at most one time unit. In addition, the high-level idea in these works is very different: All these approaches focus on speed as the *duration* of processes, while in our approach with local clocks speed describes the *processing power* of a virtually timed ambient.

Timers have been studied for both the distributed π-calculus [6, 14] and for mobile ambients [2, 1, 13]. In this line of work, timers, which are introduced to express the possibility of a timeout, are controlled by a global clock. In membrane computing, the execution of each rule similarly takes exactly one time unit, as given by a global clock [32]. To overcome this restriction of membrane computing for modeling actual chemical reactions, each rule in timed P systems [11] has an associated integer representing the time needed to complete the execution of the rule. This resembles the timer approach on mobile ambients [2, 1, 13]. In contrast, the source clocks at the global system level in our work recursively control local clocks which define the execution power of the nested virtually timed ambients. Modeling timeouts is a straightforward extension of our work.

The enhancement of a process algebra with resources as primitives is studied in [22], where priorities are added to make processes sensitive to scheduling. A similar approach with an explicit scheduling concept is studied in [29]. In contrast, the only primitive in our approach is the ambient and the scheduling is fixed in the implementation of the resource distribution. Calculi, which are simpler to implement than the ambient calculus are studied in [7] and [12]. The Kell calculus described in [7] is based on the M-calculus and uses higher order communication. The δ-calculus in [12] uses synchronous movements in order to model distributed mobile real-time business applications. However, both calculi fail to preserve the simplicity of the ambient calculus. CPL [8] is a core language for defining cloud services and their deployment on cloud platforms which aims to enable statically safe service composition and custom implementations of cloud services. In contrast to our work on virtually timed ambients, time and performance are not been considered for CPL.

Cardelli and Gordon defined a labeled transition system for their mobile ambients [10], but no bisimulation. A bisimulation relation for a restricted version of mobile ambients, called mobile safe ambients, is defined in [24] and provides the basis for later work. Barbed congruence for the same fragment of mobile ambients is defined in [36]. It is shown in [15] that name matching reduction barbed congruence and bisimulation coincide in the π-calculus. A bisimulation relation with contextual labels for the ambient calculus is defined in [30], but this approach is not suitable for providing a simple proof method. A labelled bisimulation for mobile ambients is defined by Merro and Nardelli [25], who prove that this bisimulation is equivalent to reduction barbed congruence and develop up-to-proof techniques. The weak timed bisimulation defined in this paper is a conservative extension of this approach.

7. Concluding Remarks

Virtualization opens for new and interesting foundational models of computation by explicitly emphasizing deployment and resource management. This paper introduces virtually timed ambients, a formal model of hierarchical locations of execution with explicit resource provisioning. Resource provisioning for virtually timed ambients is based on virtual time, a local notion of time reminiscent of time slices for virtual machines in the context of nested virtualization. This way, the computing power of a virtually timed ambient depends on its location in the deployment hierarchy. To reason about timed behavior in this setting, we define weak timed bisimulation for virtually timed ambients as a conservative extension of bisimulation for mobile ambients, and show that the equivalence of bisimulation and reduction barbed congruence is preserved by this extension. We define timed relaxation as an "equivalent but slower" simulation relation, allowing speed deviation up to a bounded amount of time.

The calculus of virtually timed ambients opens for further, interesting research questions. One line of research is in statically controlling resource management, for example by means of behavioral types. Another line of research is in dynamically controlling resource management by means of resource awareness. This line of work is suggested by examples in this paper such as load balancers but could be enhanced by reflective resource capabilities allowing a process to influence its own deployment similar to virtualization APIs found in the context of cloud computing.

References

[1] B. Aman and G. Ciobanu. Mobile ambients with timers and types. In C. B. Jones, Z. Liu, and J. Woodcock, editors, *Proc. 4th International Colloquium on Theoretical Aspects of Computing (ICTAC'07)*, volume 4711 of *Lecture Notes in Computer Science*, pages 50–63. Springer, 2007.

[2] B. Aman and G. Ciobanu. Timers and proximities for mobile ambients. In V. Diekert, M. V. Volkov, and A. Voronkov, editors, *Proc. 2nd International Symposium on Computer Science in Russia (CSR'07)*, volume 4649 of *Lecture Notes in Computer Science*, pages 33–43. Springer, 2007.

[3] J. C. M. Baeten and J. A. Bergstra. Real Time Process Algebra. Technical Report CS-R 9053, Centrum voor Wiskunde en Informatica (CWI), 1990.

[4] J. C. M. Baeten and C. A. Middelburg. *Process Algebra with Timing.* Monographs in Computer Science. An EATSC series. Springer, 2002.

[5] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B. Yassour. The Turtles project: Design and implementation of nested virtualization. In *Proc. 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2010)*, pages 423–436. USENIX Association, 2010.

[6] M. Berger. *Towards Abstractions for Distributed Systems.* PhD thesis, University of London, Imperial College, Nov. 2004.

[7] P. Bidinger and J. Stefani. The Kell calculus: Operational semantics and type system. In E. Najm, U. Nestmann, and P. Stevens, editors, *Proc. 6th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2003)*, volume 2884 of *Lecture Notes in Computer Science*, pages 109–123. Springer, 2003.

[8] O. Bračevac, S. Erdweg, G. Salvaneschi, and M. Mezini. CPL: A core language for cloud computing. In *Proc. 15th International Conference on Modularity (MODULARITY 2016)*, pages 94–105. ACM, 2016.

[9] L. Cardelli and A. D. Gordon. Mobile ambients. In M. Nivat, editor, *Proceedings of FoSSaCS '98*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998. An extended abstract appeared in *Proceedings of FoSSaCS '98* (LNCS1378): 140–155.

[10] L. Cardelli and A. D. Gordon. Equational properties of mobile ambients. *Mathematical Structures in Computer Science*, 13(3):371–408, 2003.

[11] M. Cavaliere and D. Sburlan. Time-independent P systems. In G. Mauri, G. Paun, M. J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa, editors, *Proc. 5th International Workshop on Membrane Computing (WMC'04)*, volume 3365 of *Lecture Notes in Computer Science*, pages 239–258. Springer, 2004.

[12] Y. Choe and M. Lee. δ-calculus: Process algebra to model secure movements of distributed mobile processes in real-time business applications. In *23rd European Conference on Information Systems, ECIS 2015, Münster, Germany, May 26-29, 2015*, 2015.

[13] G. Ciobanu. Interaction in time and space. *Electronic Notes in Theoretical Computer Science*, 203(3):5–18, May 2008.

[14] G. Ciobanu and C. Prisacariu. Timers for distributed systems. *Electronic Notes in Theoretical Computer Science*, 164(3):81–99, 2006.

[15] C. Fournet and G. Gonthier. A hierarchy of equivalences for asynchronous calculi. *Journal of Logic and Algebraic Programming*, 63(1):131 – 173, 2005.

[16] R. P. Goldberg. Survey of virtual machine research. *IEEE Computer*, 7(6):34–45, 1974.

[17] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117(2):221–239, 1995.

[18] K. Honda and N. Yoshida. Replication in concurrent combinators. In M. Hagiya and J. C. Mitchell, editors, *TACS*, volume 789 of *Lecture Notes in Computer Science*, pages 786–805. Springer, 1994.

[19] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437–486, 1995.

[20] E. B. Johnsen, R. Schlatte, and S. L. Tapia Tarifa. Integrating deployment architectures and resource consumption in timed object-oriented models. *Journal of Logical and Algebraic Methods in Programming*, 84(1):67–91, 2015.

[21] E. B. Johnsen, M. Steffen, and J. B. Stumpf. A calculus of virtually timed ambients. In Phillip James and Markus Roggenbach editors, *Postproceedings of the 23rd International Workshop on Algebraic Development Techniques (WADT 2016)*, volume 10644 of *Lecture Notes in Computer Science*. Springer Verlag, 2017. To appear.

[22] I. Lee, A. Philippou, and O. Sokolsky. Resources in process algebra. *Journal of Logic and Algebraic Programming*, 72(1):98 –122, 2007.

[23] G. Lüttgen and W. Vogler. Bisimulation on speed: Worst-case efficiency. *Information and Computation*, 191(2):105–144, June 2004.

[24] M. Merro and M. Hennessy. A bisimulation-based semantic theory of safe ambients. *ACM Transactions on Programming Languages and Systems*, 28(2):290–330, 2006.

[25] M. Merro and F. Zappa Nardelli. Behavioral theory for mobile ambients. *Journal of the ACM*, 52(6):961–1023, Nov. 2005.

[26] R. Milner. The polyadic pi-calculus: a tutorial. Technical report, Logic and Algebra of Specification, 1991.

[27] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proceedings of ICALP '92*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer, 1992.

[28] F. Moller and C. Tofts. A temporal calculus of communicating systems. In J. C. M. Baeten and J. W. Klop, editors, *Proc. 1st International Conference on Concurrency Theory (CONCUR'90)*, volume 458 of *Lecture Notes in Computer Science*, pages 401–415. Springer, 1990.

[29] M. R. Mousavi, M. A. Reniers, T. Basten, and M. R. V. Chaudron. PARS: A process algebraic approach to resources and schedulers. In M. Alexander and W. Gardner, editors, *Process Algebra for Parallel and Distributed Processing*. Chapman and Hall/CRC, 2008.

[30] M. Murakami. Congruent bisimulation equivalence of ambient calculus based on contextual transition system. In *Proc. 7th International Symposium on Theoretical Aspects of Software Engineering (TASE 2013)*, pages 149–152. IEEE Computer Society, 2013.

[31] X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: Theory and application. *Information and Computation*, 114(1):131–178, 1994.

[32] G. Paun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.

[33] D. Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012.

[34] D. Sangiorgi and D. Walker. *The Pi-Calculus: A Theory of Mobile Processes.* Cambridge University Press, 2001.

[35] I. Satoh and M. Tokoro. A timed calculus for distributed objects with clocks. In O. M. Nierstrasz, editor, *Proc. 7th European Conference on Object-Oriented Programming (ECOOP'93)*, pages 326–345. Springer, 1993.

[36] M. Vigliotti and I. Phillips. Barbs and congruences for safe mobile ambients. *Electronic Notes in Theoretical Computer Science*, 66(3):37 – 51, 2007.

[37] D. Williams, H. Jamjoom, and H. Weatherspoon. The Xen-Blanket: Virtualize once, run everywhere. In *Proc. 7th European Conference on Computer Systems (EuroSys'12)*, pages 113–126. ACM, 2012.