Measuring security in IoT communications

Chiara Bodei[a], Stefano Chessa[a], Letterio Galletta[b]

[a] *Department of Computer Science. Università di Pisa, Pisa, Italy*
[b] *IMT School for Advances Studies, Lucca, Italy*

Abstract

More smart objects and more applications on the Internet of Things (IoT) mean more security challenges. In IoT security is crucial but difficult to obtain. On the one hand the usual trade-off between highly secure and usable systems is more impelling than ever; on the other hand security is considered a feature that has a cost often unaffordable. Therefore, IoT designers not only need tools to assess possible risks and to study countermeasures, but also methodologies to estimate their costs. Here, we present a methodology, based on the process calculus IoT-LySa, to infer quantitative measures on evolution of systems. The derived quantitative evaluation is exploited to establish the cost of the possible security countermeasures, in terms of time and energy.

Keywords: Internet of Things, Security, Cost evaluation

1. Introduction

We are living the Internet of Things (IoT) revolution, where the *things* we use every day have computational capabilities and are always connected to the Internet. These "smart" devices, which are equipped with sensors, automatically collect a huge amount of data, store them on the cloud or use them to affect the surrounding environment through actuators. This emerging technology provides the momentum for the creation of new societal and economical opportunities, since it is becoming the enabling technology of the future houses, cities, industrial plants, farms, as well as critical infrastructures, i.e. the assets underpinning the functioning of an economy and of a society [1]. As an example of an IoT system, consider a smart street light control system [2] in a smart city scenario. There lamp posts are equipped with sensors to acquire data about the physical environment and regulate the level of illumination according to the hour of day and the detected events in the street like the presence of a walking person. Similarly, a smart lighted pedestrian crossing system is based on warning lights placed ahead of the crosswalk lines that start blinking as pedestrians approach the

Email addresses: chiara.bodei@unipi.it (Chiara Bodei), stefano.chessa@unipi.it (Stefano Chessa), letterio.galletta@imtlucca.it (Letterio Galletta)

zone. Although the benefits of the IoT are undeniable, this paradigm introduces new pressing security challenges, because "important processes that once were performed manually (and thus enjoyed a measure of immunity against malicious cyber activity) are now vulnerable to cyber threats" [3]. Although the precise security guarantees depend on the application, basic security mechanisms are anyway needed to ensure the functionality of the network and to protect the privacy and integrity of the processed and exchanged sensitive data.

However, security in this setting is easy to go wrong, as shown by many examples in news, see [4, 5, 6] to cite only a few. On the one hand, it is extremely hard to guarantee correctness properties of those systems: indeed they are highly distributed and their overall correctness is the resultant of that of its components and their interactions. On the other hand, smart objects have limited computational capabilities and are battery powered. Hence, this shortage of resources heavily constraints the choice of security mechanisms that can be implemented. In particular, in this setting energy can be a precious resource since its lack can affect the whole system lifetime: indeed, it is sufficient a single node that runs out of charge to disconnect all the network. Consequently, the usual trade-off between highly secure and usable systems is more critical than ever. Indeed, IoT designers have to be selective in choosing which security mechanisms adopt in order to optimise computational capabilities and battery power. For this reason, designers not only need tools to assess possible risks and to study countermeasures, but also methodologies and tools to estimate their costs. These costs can be computed in terms of time overhead, energy consumption, bandwidth, and so on. All these factors must be carefully evaluated for achieving an acceptable balance among security, cost and usability.

Contribution. Usually, formal methods provide designers with tools to support the development of systems and to reason about their properties, both qualitative and quantitative. In this paper we advocate formal methods as an enabling technology to support a *security by design* development model. In particular, we present a formal methodology to model an IoT system and to analyse the cost of the adopted security mechanisms. We aim at providing a general framework with a mechanisable procedure (with a small amount of manual tuning), where quantitative aspects are symbolically represented by parameters. Their instantiation is delayed until hardware architectures and cryptographic algorithms are fixed. By only changing these parameters designers could compare different implementations of an IoT system and could choose the better trade-off between security and costs. Carefully evaluating the costs and opting for prudent choices can impact on the overall performance of the network but also on the throughput of the network lifetime.

Technically, our starting point is the formal specification language IoT-LySa, a process calculus recently proposed for describing IoT systems [7, 8, 9, 10, 11, 12]. Designers can model the architecture of a system, the algorithmic behaviour of its smart objects and their interactions through the IoT-LySa primitives. Furthermore, IoT-LySa enables them to reason about qualitative properties as system correctness and robustness by using a static analysis, namely Control Flow

Analysis (CFA). This analysis safely carries out a behavioural forecast, i.e. over-approximation of the system behaviour, at static time, without actually running the system. In practice, it predicts how data from sensors may spread across the system and how objects may interact with each other. To this aim, it "mimics" the evolution of the system, by using abstract values in place of concrete ones and by modelling the consequences of each possible action. From this static "simulation" designers can detect possible flaws and intervene as early as possible during the design phase. In [9] the CFA was used to check both functional and non-functional properties, in particular whether a system complies with some standard security policies as confidentiality, no read-up/no write-down, etc.

In this paper, we define an *enhanced semantics* for IoT-LySa, following the methodology of [13, 14, 15, 16, 17], where each transition is associated with a label recording the actions performed during the transition. These annotations offer the basis for the performance evaluation. In fact, we introduce functions over the enhanced labels to associate costs with transitions. Here, costs are first given in the form of rates: they are specified in terms of the *time* spent for transitions, and depend on the performed action as well as on the involved nodes. In particular, we associate costs with the actions of sensing, storing a value, sending and receiving messages, encryption and decryption, and with the application of aggregation functions. Costs are assigned by using parameters that can be instantiated according to some design choices, e.g. depending on the adopted encryption schema. The accuracy of cost measures depends on the precision of this instantiation. Intuitively, cost functions define exponential distributions, from which we compute the rates at which a system evolves. From rates, we mechanically derive a Continuous-Time Markov Chain (CTMC), which can be analysed using standard techniques and tools [18, 19]. Indeed, to evaluate the performance of the system we calculate the stationary distribution of the derived CTMC and its transition rewards. Furthermore, we define a bisimulation-based equivalence as a further tool for comparing program performances, by combining qualitative and quantitative aspects.

Another critical resource to preserve and carefully use in an IoT system is the energy, since very often smart objects are battery powered. On the one hand, replacing or recharging the battery is usually a costly operation, which may not even be feasible in systems deployed in remote and unreachable areas [20]. On the other hand, the lifetime of the network can be affected by only one single node that stops working by lack of energy. Still starting from the evaluation of the time, we analyse energy consumption. We assume that the energy consumed by a node is proportional to the time spent on carrying its computational and communication activities out. In particular, we introduce a notion of energy budget representing how much energy a node owns and/or may spend. We also define an *energy-sensitive* semantics, on top of the enhanced one, to deal with these budgets. An energy budget can be seen as the initial quantity of energy available for each node and it is consumed during the execution. Thus, a designer can understand if, given a bound on the resources available in terms of an initial quantity of charge (the initial budget), the system is able to complete a certain task. Alternatively, the energy budget can be defined as the energy

a node requires in order to complete a certain sequence of computation and communication tasks. In this second case, a designer can compute the minimal budget required to complete a single duty cycle.

More in general, designers can exploit a cost analysis to tune the trade-off between security and resource usage in order to have a robust system at an affordable price. Furthermore, they can compare different approaches or solutions to implements a system and choose the more convenient or cheaper. Note that our goal is providing a suitable methodology for analysing the cost of the chosen security mechanisms in IoT system. We are not interested here in explicitly modelling possible attacks. Their existence is implicitly assumed and motivates the performance analysis of different designs of the same network, each adopting different countermeasures to prevent or to mitigate the attacks.

Structure of the paper. In Section 2, we introduce and motivate our methodology by using an illustrative, even though realistic, scenario based on a smart storehouse. We briefly recall the process calculus IoT-LySa in Section 3, and we show its enhanced semantics. Section 4 defines the stochastic semantics, by introducing cost functions that assign rates to transitions; we also describe how to obtain the CTMC associated with a given system of nodes and how to extract performance measures from it. In Section 5, we describe our energy-sensitive semantics and show how to use it to reason on the required energy for a system. Our methodology is applied to the simplified version of the systems describing our smart storehouse scenario in Sections 4 and 5. Section 6 discusses related work, while some conclusions are drawn in Section 7.

This article is the full and revised version of the conference paper published in [21], where a subset of the IoT-LySa language was considered with no actuators, no shared store and no asynchronous communications. Here we propose an enhanced semantics for the whole IoT-LySa language as presented in [9], extended with a guarded choice construct. Consequently, we reworked and extended the section concerning the assignment and computation of the costs of transitions. Moreover, we introduce a bisimulation-based equivalence and an energy-sensitive semantics, not present in [21]. Finally, we apply our methodology to a more involved scenario.

2. A storehouse with perishable food

In this section we intuitively present our methodology and the language IoT-LySa (formally introduced in Section 3) by using, as running example, an elaboration of the one presented in [21, 10]. IoT-LySa is a specification language, based on the process algebra theory, that provides designers with linguistic constructs to design and specify a network of smart objects.

2.1. The design of IoT-LySa

IoT-LySa describes systems of nodes that represent smart objects. Intuitively, the definition of a node consists of two interacting parts.

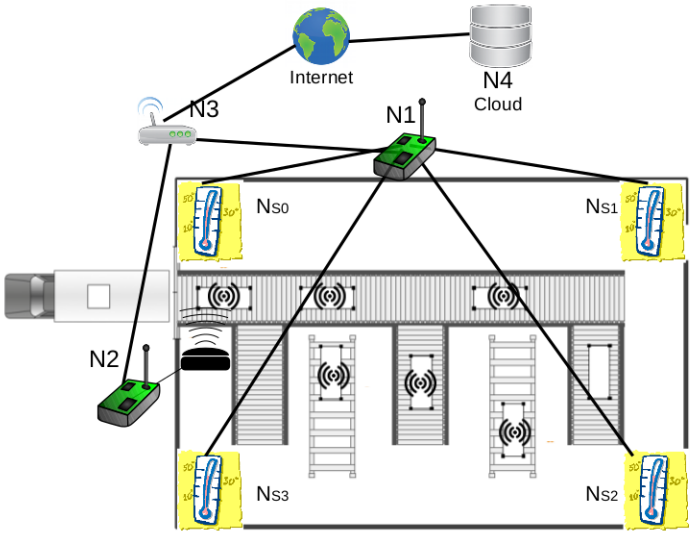

Figure 1: The organisation of nodes in the IoT system of a storehouse with perishable food.

- The first part concerns the logical behaviour of nodes (the software). It is modelled in IoT-LySa as a set of interacting control processes. These processes coordinate communications with the other nodes, manage data gathered from sensors and trigger commands to actuators.
- The second part is cyber-physical and deals with the physical world. It is mainly made of (small pieces of hardware) sensors and actuators that interact with the environment, by collecting data and performing some mechanical actions.

In IoT-LySa, each node is uniquely identified by a label ℓ. The label may abstractly represent information characterising the node (e.g. network address, serial number, and so on). As anticipated, a node is made of control processes, possibly sensors and actuators, and a shared store working as a memory that records the data collected by sensors and the data computed and exchanged by processes.

Since we are mainly interested in modelling the logical behaviour of the system and the way it affects the physical world, sensors and actuators are implemented by abstracting all the low level details of their execution. In particular, we will consider sensors as simple data providers and actuators as passive entities that execute the requested actions when processes ask them.

These two parts are linked together and communicate through the shared store that is assumed to be accessed atomically by sensors and control processes.

We assume indeed that each sensor, identified by an index i, is provided with a reserved location (like a register) in the store where it can record data read from the environment by making them available to the other node components.

Similarly, actuators are provided with a unique index j which identifies them. Finally, in IoT-LySa inter-node communications are based on a message passing mechanism representing the wireless multicast transmission of packets through explicit send/receive constructs.

2.2. A smart storehouse

Consider a smart storehouse, call it SmartStore, storing perishable food, the structure of which is illustrated in Figure 1. To correctly preserve the stocks, the temperature inside the room must be kept under control and regulated depending on the quantity and the kind of food. The system also monitors how long the food is stored inside the storehouse, allowing managers to schedule the order in which the food leaves the warehouse. For instance, they can give priority to food approaching to the expiry-date, to minimise economic loss.

The corresponding network of smart objects is made of eight nodes N_i and N_{s_i} with $i \in [0, 3]$. The four nodes N_{s_i} are located in the corners of the storehouse and play the role of data collectors. In particular, each node N_{s_i} is equipped with a sensor S_{s_i}, which senses the temperature in the area nearby and records it in the shared store. The sensed values are elaborated by the control process of N_{s_i} and sent to the node N_1.

The node N_1 works as data aggregator and temperature regulator. It is equipped with an actuator that allows activating the cooling system of the storehouse. When N_1 receives the temperature data from each node N_{s_i}, computes their average, and sends it to the node N_3. Then, it waits for instructions from N_3, the system controller, on how regulate the temperature in the storehouse.

The node N_2 monitors the stored food and takes care of doing the stocktaking. It is equipped with a RFID reader, located nearby the entrance. We assume that each wood box containing food is equipped with an RFID, read when it is stored. When a wood box enters the storehouse the node N_2 uses its RFID reader to read the box identifier, updates the stocktaking database and sends the updated value to the node N_3.

The node N_3 works as the system controller and as gateway towards the Cloud (i.e. the node N_4). It receives the data on the temperature from N_1, and the data on the stocktaking by N_2. Then, it sends a message to N_4 to log the temperature. Afterwards, it checks whether the temperature is acceptable for the quantity and the kind of the stored food. Depending on this check, it sends a message to N_1 containing information about the proper actions to take on the cooling system.

Finally, the N_4 implements an Internet service that waits for messages from N_3 and handles them.

2.3. Reasoning on the design

Here, we are interested in reasoning on the security of our system. If all communications are in clear then they are not robust: an attacker can easily intercept and manipulate data, especially those sent by nodes N_{s_i}. This means that an attacker may easily violate the confidentiality and integrity of data.

A possible (and very naive) solution consists in using cryptography to protect all communications. Although this approach clearly makes our storehouse system more robust, it may heavily impact on the cost, in terms of speed and of power consumption, as well as, in terms the overall performance and lifetime of the network, critical when e.g. the hardware platforms have limited computational capabilities or when the battery must be preserved. Thus, a good trade-off between the level of security and the resource consumption is required.

A good balance can be obtained by mixing data redundancy and cryptography. Concerning data redundancy, we can exploit the fact that thermometers on the same side of the storehouse should sense the same temperature, with a difference that can be at most a given threshold value ϵ. Thus, the system can easily detect anomalies and discard data tailored by an attacker, by comparing values coming from the nodes that are on the same side of the room. This approach could be sufficient when the attacker can falsify data coming from at most one thermometer. Adding cryptography can help us in making the approach more robust, even when an attacker can intercept and falsify data sent by more than one sensor. However, the designer has to find the right trade-off between the advantages given by redundancy and cryptography and the drawbacks due to their underlying costs. For example, a high redundant network may require more nodes, more messages exchanged or more storage; using cryptography may require powerful processors or non-trivial hardware capabilities.

We would like to formally reason on these possible solutions and understand which one ensures a good level of security but at a reasonable cost in terms of required resources, e.g. battery energy. Once used IoT-LySa to model the architecture of a system, the algorithmic behaviour of its smart objects and their interactions, our methodology consists in associating a cost with each action of the specification representing the time and the energy spent by a node performing that action. Thus, taking the semantics of the specification, we can associate a global cost with each node denoting the time required for running all its activities. As we will see, these costs can help designers to answer questions about the feasibility of their design on a given platform and guide them to modify it, when it does not meet their requirements. Furthermore, they can compare the relative costs of two solutions both in terms of time and of energy and choose the more appropriate one. In practice, the goal of our methodology is to support the automatic selection of a suitable allocation strategy of the security functions that can preserve a certain security level. Thus, a designer can compare different allocation strategies of these functions and select the one that is less expensive in terms of costs, e.g. in terms of energy, but that meets the needed security requirements.

As the example shows, we assume the existence of possible attacks, but we are not interested in explicitly modelling them. We focus on the performance analysis of the different countermeasures that can be adopted to prevent or to mitigate the attacks.

3. IoT-LySa and its Enhanced Semantics

IoT-LySa is an adaption of LySa [22, 23], a process algebra introduced to specify and analyse cryptographic protocols and checking their security properties [24, 25, 26]. Here we introduce the syntax and the semantics of IoT-LySa, by slightly improving the version of [9]. In particular, we introduce an external choice construct and enrich each transition in the semantics with a label θ carrying information about the actions performed during that transition. In the next section, these labels will be associated with a cost and will provide the basis to generate a CTMC to perform our quantitative analysis.

3.1. Syntax

IoT-LySa has a two-level syntax describing the behaviour of nodes and of node components, respectively. An IoT-LySa system $N \in \mathcal{N}$ consists of a fixed number of nodes, each identified by a label $\ell \in \mathcal{L}$, that communicate through message-passing. The label ℓ uniquely identifies a node $\ell : [B]$ and may represent further characterising information (e.g. its location or other contextual information). Each node hosts sensors $S \in \mathcal{S}$, actuators $A \in \mathcal{A}$ and control processes $P \in \mathcal{P}$, all running in parallel and interacting with each other through a shared store. Concerning cryptographic primitives, we assume that there exists a finite set of keys, *a priori* associated to each node.

Let $\mathcal{V}$ be a denumerable set of values (including numbers, booleans etc.), $\mathcal{I}_\ell$ and $\mathcal{J}_\ell$ be identifiers of sensors and actuators, respectively. The syntax of IoT-LySa systems is in Figure 2.

In the syntax of systems, $\mathsf{0}$ denotes the null inactive system; $\ell : [B]$ denotes a single node; and the parallel composition operator $|$ combines nodes. Inside a node there is a parallel composition (by the operator $||$) of components B: control processes P, sensors S, actuators A and a shared store Σ.

We impose that in $\ell : [B]$ there is always a *single* store $\Sigma_\ell : \mathcal{X}_\ell \cup \mathcal{I}_\ell \ \rightarrow \mathcal{V}$. Therefore, a store is essentially an array of fixed dimension, and intuitively a variable $x \in \mathcal{X}_\ell$ and an identifier $i \in \mathcal{I}_\ell$ are interpreted as indexes in the array (no need of α-conversions). We assume that store accesses are atomic, e.g. through CAS instructions [27].

The process $\mathsf{0}$ represents the inactive one. The prefix τ represents the fact that the process performs some internal actions and then it continues as P. The prefix $\langle\!\langle E_1, \cdots, E_m \rangle\!\rangle \triangleright L$ implements a simple form of multicast communication among nodes: the tuple $E_1, \ldots, E_m$ is asynchronously sent to the nodes with labels in L. As we will see below, communications in IoT-LySa are based on a notion of compatibility, defined by different attributes, including a proximity of the interlocutors and the transmission capabilities of the sender. The construct $+$ receives a message by considering two alternatives represented by the two input prefixes. An input prefix $(E_1, \cdots, E_j; x_{j+1}, \cdots, x_m)$ accepts a m-tuple, provided that its first j terms match the input ones (pattern matching is embedded in the input), and then binds the remaining store variables (separated by a ";") to the corresponding values. When the process receives a message, it tries to match the message against the two possible input prefixes. If one of the two prefixes

$\mathcal{N} \ni N ::=$	*systems of nodes*	
	$\mathsf{0}$	empty system
	$\ell : [B]$	single node ($\ell \in \mathcal{L}$, the set of labels)
	$N_1 \mid N_2$	parallel composition of nodes
$\mathcal{B} \ni B ::=$	$n$ode components	
	Σ_ℓ	store of node ℓ
	P	process
	S	sensor, with a unique identifier $i \in \mathcal{I}_\ell$
	A	actuator, with a unique identifier $j \in \mathcal{J}_\ell$
	$B \parallel B$	parallel composition of node components
$\mathcal{P} \ni P ::=$	*control processes*	
	$\mathsf{0}$	inactive process
	$\tau.P$	internal action
	$\langle\!\langle E_1, \cdots, E_r \rangle\!\rangle \triangleright L.\, P$	asynchronous multi-output L$\subseteq \mathcal{L}$
	$(E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_m).\, P +$ $(E'_1, \cdots, E'_j;\ x'_{j+1}, \cdots, x'_m).\, Q$	switch or external choice
	$E?P : Q$	conditional statement
	P_A	constant def.
	decrypt E as $\{E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_m\}_{K_0}$ in P	decryption (with match.)
	$x := E.\, P$	assignment to $x \in \mathcal{X}_\ell$
$\mathcal{E} \ni E ::=$	*terms*	
	v	value ($v \in \mathcal{V}$)
	i	sensor location ($i \in \mathcal{I}_\ell$)
	x	variable ($x \in \mathcal{X}$)
	$\{E_1, \ldots, E_m\}_{k_0}$	encryption with key k_0 ($k \geq 0$)
	$f(E_1, \ldots, E_m)$	function on data

$\mathcal{S} \ni S$	$::=$ *sensors*		$\mathcal{A} \ni A$	$::=$ *actuators*	
	$\mathsf{0}$	inactive sensor		$\mathsf{0}$	inactive actuator
	$\tau.S$	internal action		$\tau.A$	internal action
	$i := v.\, S$	store of $v \in \mathcal{V}$ by the i^{th} sensor		$(\!\lvert j, \Gamma \rvert\!).\, A$	command for actuator j
				$\gamma.A$	triggered action ($\gamma \in \Gamma$)
	S_A	constant def.		A_A	constant def.

Figure 2: The syntax of IoT-LySa.

matches, then the computation continues by selecting the corresponding branch. Otherwise, the m-tuple is not accepted. We assume that the pattern matching expression of the input prefixes in hand are mutually exclusive, so the construct is deterministic. In the rest of the paper we use $(E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_m).P$ without $+$ as a shorthand to denote an external choice whose second element has 0 as continuation. An agent is a static definition of a parametrised process. Each agent identifier P_A has a unique defining equation of the form $P_A = P$. The process $\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j; x_{j+1}, \cdots, x_m\}_{k_0}\ \mathsf{in}\ P$ tries to decrypt an encrypted value using the key k_0, provided that the first j elements of the decrypted term coincide with the terms E_j. We assume a symmetric encryption schema, which is most used in IoT systems because less energy consuming. As done in Section 2, we often use the abbreviation $(\{E_1, \cdots, E_j; x_{j+1}, \cdots, x_m\}_{K_0}).P$ to denote the process $(; x).\mathsf{decrypt}\ x\ \mathsf{as}\ \{E_1, \cdots, E_j; x_{j+1}, \cdots, x_m\}_{k_0}\mathsf{in}\ P$, i.e. the process that receives a term and immediately decrypts it.

A sensor, identified by a unique identifier belonging to the sets $\mathcal{I}_\ell$, can perform an internal action τ, e.g. noise reduction, or store a value v, gathered from the environment, in the shared store and continues as S. We do not provide an explicit operation to read data from the environment but it is implemented as an early input.

An actuator can perform an internal action τ or execute one of its action γ, possibly received from its controlling process. Both sensors and actuators can repeat their behaviour.

Finally, in the syntax of term, a value represents a piece of data (e.g. keys or values read the environment). The encryption function $\{E_1, \ldots, E_m\}_{k_0}$ returns the result of encrypting values E_i for $i \in [1, m]$ with the key k_0. The term $f(E_1, \ldots, E_m)$ is the application of function f to m arguments; we assume given a set of primitive aggregation functions $\mathcal{F}$, typically for aggregating or comparing values, either computed or sampled from the environment.

3.2. Enhanced Operational Semantics

To estimate cost, we give an *enhanced* reduction semantics following [13, 14, 15, 16, 17]. The underlying idea consists in enriching each transition with an *enhanced label* θ that records what happen during the transition. Note that labels only record information but they do not affect the semantics: indeed, the standard semantics can be obtained by simply removing the labels. IoT-LySa has a two-level semantics. The first level describes how a single node internally behaves, i.e. how its components evolve; the second level describes instead how the network globally behaves. Accordingly, we have two kinds of enhanced labels: the labels Θ_B that enrich the transitions of the first level semantics and the labels Θ that enrich the transitions of the second level semantics. They are defined as follows.

Definition 1. Given $\ell, \ell_O, \ell_I \in \mathcal{L}$, the sets of labels Θ_B (ranged over by θ_B), Θ_C (ranged over by θ_C), and the set of *enhanced labels* Θ (ranged over by θ) are defined as:

$$
\begin{array}{lll}
\theta_B ::= & sens(i) & \text{sensing} \\
& store(E) & \text{storing a term} \\
& \langle\!\langle E_1, ..., E_r \rangle\!\rangle & \text{sending a tuple of terms} \\
& do(j, \gamma) & \text{triggering} \\
& act(\gamma) & \text{actuating} \\
& int & \text{internal activity} \\
& dec(E, \{E'_1, \cdots, E'_j; x_{j+1}, \cdots, x_m\}_{k_0}) & \text{decryption of a message} \\
\\
\theta_C ::= & \langle\!\langle E_1, \cdots, E_m \rangle\!\rangle, (E_1, \cdots, E_j; x_{j+1}, \cdots, x_m) & \text{exchanged message} \\
\\
\theta ::= & \ell\ \langle \theta_B \rangle & \text{propagation of actions} \\
& \ell_O \triangleright \ell_I\ \langle \theta_C \rangle & \text{inter-nodes comm.}
\end{array}
$$

The function Λ used to remove the $\mathcal{L}$ component from the enhanced labels is defined as follows

$$
\begin{array}{l}
\Lambda(\ell\ \langle \theta_B \rangle) = \theta_B \\
\Lambda(\ell_O \triangleright \ell_I\ \langle \theta_C \rangle) = \theta_C
\end{array}
$$

Note that labels embody possible encrypted terms: encryption does not corresponds to any prefix of processes.

The labels for nodes components record actions occurred inside a node: sensing from the sensor i ($sens(i)$); evaluating an expression E and storing its result ($store(E)$); sending a message made of r terms ($\langle\!\langle E_1, ..., E_r \rangle\!\rangle$); driving an actuator and affecting the environment ($do(j, \gamma)$ and $act(\gamma)$); performing and internal action (int); performing a decryption ($dec(E, \{E'_1, \cdots, E'_j; x_{j+1}, \cdots, x_m\}_{k_0})$); and exchanging a message $\langle\!\langle E_1, \cdots, E_m \rangle\!\rangle, (E_1, \cdots, E_j; x_{j+1}, \cdots, x_m)$.

There are two kinds of labels for nodes: (i) those with the form $\ell\ \theta_B$, where the label ℓ identifies the node, by lifting a component label to a node one; (ii) those associated with communications transitions that record the label of the sender ℓ_O, the label of the receiver ℓ_I and the exchanged message: $\ell_O \triangleright \ell_I\ \langle \theta_C \rangle$. For the sake of simplicity, in the following we feel free to use the single label

$$\ell_O \triangleright \ell_I\ \langle \langle\!\langle \{E_1, E_2\}_k \rangle\!\rangle, (\{E_1; x_2\}_k) \rangle$$

to endow the complex action of receiving and decrypting at the same time. In [13, 14, 15, 16, 17], labels also record the inference rules used during the deduction of the transitions, in particular those given by the application of parallel composition. The reason is that they encode the locations of the involved processes with the position w.r.t the abstract syntax tree. We do not need to do it because our systems come with labels that explicitly identify their locations.

As usual, our semantics consists of the standard structural congruence $\equiv$ on nodes, processes and sensors and of a set of rules defining the transition relation.

- $(\mathcal{N}/_{\equiv}, |, \mathsf{0})$ and $(\mathcal{B}/_{\equiv}, \|, \mathsf{0})$ are commutative monoids
- $\mu\, h\, .\, X \equiv X\{\mu\, h\, .\, X/h\}$ for $X \in \{P, A, S\}$
- $\langle \langle E_1, \cdots, E_m \rangle \rangle : \emptyset.\ \mathsf{0} \equiv \mathsf{0}$

Our notion of *structural congruence* $\equiv$ is standard except for the last congruence rule for processes that equates a multi-output with empty set of receivers to the inactive process.

We have a two-level *reduction relation* $\rightarrow$ defined on nodes and its components. It is the least relation satisfying the set of inference rules in Table 1. In the definition of the rules, we assume the standard denotational interpretation $[\![E]\!]_\Sigma$ for evaluating terms. We briefly comment them below.

(S-store)
$$\Sigma \parallel i := v.\, S_i \parallel B \xrightarrow{sens(i)} \Sigma\{v/i\} \parallel S_i \parallel B$$

(Asgm)
$$\frac{[\![E]\!]_\Sigma = v}{x := E.\, P \parallel B \xrightarrow{store(E)} \Sigma\{v/x\} \parallel P \parallel B}$$

(Ev-out)
$$\frac{\bigwedge_{i=1}^{m} v_i = [\![E_i]\!]_\Sigma}{\Sigma \parallel \langle\!\langle E_1, \cdots, E_m \rangle\!\rangle \triangleright L.\, P \parallel B \xrightarrow{\langle\!\langle E_1, \cdots, E_m \rangle\!\rangle} \Sigma \parallel \langle\!\langle v_1, \cdots, v_m \rangle\!\rangle \triangleright L.0 \parallel P \parallel B}$$

(Multi-Com)
$$\frac{\ell_2 \in L \wedge\ Comp(\ell_1, \ell_2) \wedge\ \bigwedge_{i=1}^{j} v_i = [\![E_i]\!]_{\Sigma_2}}{\begin{array}{l}\ell_1 : [\langle\!\langle v_1, \cdots, v_m \rangle\!\rangle \triangleright L.\, 0 \parallel B_1] \mid \\ \ell_2 : [\Sigma_2 \parallel (E_1, \cdots, E_j; x_{j+1}, \cdots, x_m).Q + (E'_1, \cdots, E'_j; x_{j+1}, \cdots, x'_m).Q' \parallel B_2] \\ \xrightarrow{\ell_1 \triangleright \ell_2\ \langle \langle\!\langle v_1, \cdots, v_m \rangle\!\rangle, (E_1, \cdots, E_j; x_{j+1}, \cdots, x_m) \rangle} \\ \ell_1 : [\langle\!\langle v_1, \cdots, v_m \rangle\!\rangle \triangleright L \setminus \{\ell_2\}.\, 0 \parallel B_1] \mid \ell_2 : [\Sigma_2\{v_{j+1}/x_{j+1}, \cdots, v_m/x_m\} \parallel Q \parallel B_2]\end{array}}$$

(Decr)
$$\frac{[\![E]\!]_\Sigma = \{v_1, \cdots, v_m\}_{k_0} \ \wedge\ \bigwedge_{i=1}^{j} v_i = [\![E'_i]\!]_\Sigma}{\Sigma \parallel \mathsf{decrypt}\ E\ \mathsf{as}\ \{E'_1, \cdots, E'_j; x_{j+1}, \cdots, x_m\}_{k_0}\ \mathsf{in}\ P \parallel B \xrightarrow{\langle d \rangle} \Sigma\{v_{j+1}/x_{j+1}, \cdots, v_m/x_m\} \parallel P \parallel B}$$

(A-com)
$$\frac{\gamma \in \Gamma}{\langle j, \gamma \rangle.\, P \parallel (\!| j, \Gamma |\!).\, A \parallel B \xrightarrow{do(j,\gamma)} P \parallel \gamma.\, A \parallel B}$$

(Act)
$$\frac{}{\gamma.A \xrightarrow{act(\gamma)} A}$$

(Int)
$$\frac{}{\tau.\, X \xrightarrow{int} X}$$

(Node)
$$\frac{B \xrightarrow{\theta_B} B'}{\ell : [B] \xrightarrow{\ell\langle\theta_B\rangle} \ell : [B']}$$

(ParN)
$$\frac{N_1 \xrightarrow{\theta} N'_1}{N_1 | N_2 \xrightarrow{\theta} N'_1 | N_2}$$

(ParB)
$$\frac{B_1 \xrightarrow{\theta_B} B'_1}{B_1 \| B_2 \xrightarrow{\theta_B} B'_1 \| B_2}$$

(CongrY)
$$\frac{Y'_1 \equiv Y_1 \xrightarrow{\theta_Y} Y_2 \equiv Y'_2}{Y'_1 \xrightarrow{\theta_Y} Y'_2}$$

Table 1: Reduction semantics, where $d = dec(E, \{E'_1, \cdots, E'_j; x_{j+1}, \cdots, x_m\}_{k_0})$, $X \in \{P, S, A\}$, $Y \in \{N, B\}$, and $\theta_Y \in \Theta$ if $Y = N$, $\theta_Y \in \Theta_B$ if $Y = B$.

The first two rules implement the (atomic) asynchronous update of shared variables inside nodes, by using the standard notation $\Sigma\{-/-\}$. According to (S-store), the i^{th} sensor uploads the value v, gathered from the environment, into the store location i. This rule implements a sort of early input in the spirit of the one of the π-calculus [28] and records the sensing action performed by the sensor i, in the label of the transition. According to (Asgm), a control process updates the variable x with the value of E and the label records that during the transition the store is updated by the evaluation of the expression E.

The rules (Ev-out) and (Multi-com) drive asynchronous multi-communications among nodes. In the first rule a node labelled ℓ willing to send a tuple of values $\langle\!\langle v_1, ..., v_m \rangle\!\rangle$, obtained by the evaluation of $\langle\!\langle E_1, ..., E_m \rangle\!\rangle$, spawns a new process.

This process runs in parallel with the continuation P and offers the evaluated tuple to all the receivers in L. As a consequence, transmission is a *non-blocking* action. The spawned process terminates when all receivers have received the message, i.e. when the set L is empty (see the last congruence rule and below). The label generated during the transition records that the node is sending a message whose content is obtained by evaluating the expressions $E_1, \ldots, E_m$.

In the rule (Multi-com), the message coming from ℓ_1 is received by a node labelled ℓ_2. The communication succeeds, provided that (i) ℓ_2 belongs to the set L of possible receivers; (ii) the two nodes are compatible according to the compatibility function $Comp$; and (iii) the first j values of the message match with one of the two branches of the "+" operator, i.e. if they match the evaluations of the first j terms in one of the input prefixes. Moreover, the label ℓ_2 is removed by the set of receivers L of the tuple. The compatibility function $Comp$ defined over node labels is used to model real world constraints on communication, e.g. proximity, with $Comp(\ell_1, \ell_2)$ that yields true only when the two nodes ℓ_1, ℓ_2 are in the same transmission range. Finally, the label of the transition records the occurred communication between the nodes ℓ_1 and ℓ_2 and information about the pattern matching performed, where $x_{j+1}, ..., x_m$ in the transition label are immaterial placeholders recalling the shape of the message.

The inference rule (Decr) expresses the result of decrypting an encrypted term having the form $\{E_1, \cdots, E_m\}_{k_0}$ and of matching it against the pattern $\{E'_1, \cdots, E'_j; x_{j+1}, \cdots, x_m\}_{k_0}$, i.e. the pattern occurring in the corresponding decryption. Similarly, to the communication case, we require that the value v_i of each E_i matches that of the corresponding E'_i for the first j components and that the keys are the same i.e. K_0 (this models *perfect* symmetric cryptography). When successful, the values of the remaining expressions are bound to the corresponding variables. The label of the transition stores the expression to be decrypted and those used during the pattern matching.

A process commands the j^{th} actuator through the rule (A-com), by sending it the pair $\langle j, \gamma \rangle$; γ becomes the first prefix of the actuator, if it is one of its actions. The label records that the identifier of the actuator and the sent action. The rule (Act) says that the actuator performs the action γ, which is recorded also in the label of the transition. Similarly, for the rules (Int) for internal action. The last rules propagate reductions across parallel composition ((ParN) and (ParB)) and nodes (Node), while the (CongrY) are the standard reduction rules for congruence. Note that in the rule (Node), the label θ is endowed with the label ℓ that identifies the node in which the transition occurs.

Hereafter, we assume the standard notion of transition system: intuitively, it is a graph, where the nodes represent systems of nodes and the (labelled) arcs the possible transitions between them. The semantics above describes how a system behaves at run time and, once the semantics of the aggregation function is provided, it also provides the values a node in a system computes and records in their stores. Therefore, the size of the transition system depends strongly on the values read by the sensors and on the content of the stores. Thus, it could be very large or potentially infinite. However, in the analysis of the next section we are not interested in the actual results of a computation as the fact that a sensor

$$
\begin{array}{lcl}
N^p & = & N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1 \mid N_2 \mid N_3 \mid N_4 \\
N_{s_i} & = & \ell_{s_i} : [\Sigma_{s_i} \parallel P_{s_i} \parallel S_{s_i}] \qquad \text{where } S_{s_i} = s_i := v_{s_i}.\tau.S_{s_i} \qquad i \in [0,3] \\
P_{s_i} & = & (; x_{start_i}).\langle\!\langle \ell_{s_i}, s_i \rangle\!\rangle \triangleright \{\ell_1\}.P_{s_i} \qquad \text{where } i \in [0,3]
\end{array}
$$

$$
\begin{array}{lcl}
N_1 & = & \ell_1 : [\Sigma_1 \parallel P_1 \parallel A_0] \qquad \text{where } A_0 = (\!|0,\Gamma|\!).\, A_0 \\
P_1 & = & \langle\!\langle start_0 \rangle\!\rangle \triangleright \{\ell_{s_0}\}.(\ell_{s_0}; z_0).\langle\!\langle start_1 \rangle\!\rangle \triangleright \{\ell_{s_1}\}.(\ell_{s_1}; z_1). \\
 & & \langle\!\langle start_2 \rangle\!\rangle \triangleright \{\ell_{s_2}\}.(\ell_{s_2}; z_2).\langle\!\langle start_3 \rangle\!\rangle \triangleright \{\ell_{s_3}\}.(\ell_{s_3}; z_3). \\
 & & \langle\!\langle avg(z_0, z_1, z_2, z_3) \rangle\!\rangle \triangleright \{\ell_3\}.(; x_{ad}).\langle A_0, x_{ad} \rangle.\, P_1
\end{array}
$$

$$
\begin{array}{lcl}
N_2 & = & \ell_2 : [\Sigma_2 \parallel P_2 \parallel R_0] \qquad \text{where } R_0 = r_0 := w.\tau.R_0 \\
P_2 & = & (; x_{begin_2}).db := update(db, r_0).\langle\!\langle db \rangle\!\rangle \triangleright \{\ell_3\}.(; x_{go_2}).P_2
\end{array}
$$

$$
\begin{array}{lcl}
N_3 & = & \ell_3 : [\Sigma_3 \parallel P_3] \\
P_3 & = & (; x_{avg}).\langle\!\langle begin_2 \rangle\!\rangle \triangleright \{\ell_2\}.(; x_{db}).\langle\!\langle ActDec(x_{avg}, x_{db}) \rangle\!\rangle \triangleright \{\ell_1\}. \\
 & & \langle\!\langle x_{avg} \rangle\!\rangle \triangleright \{\ell_4\}.P_3
\end{array}
$$

$$
\begin{array}{lcl}
N_4 & = & \ell_4 : [\Sigma_4 \parallel P_4] \\
P_4 & = & (; w_{avg}).R_c
\end{array}
$$

Figure 3: An IoT-LySa specification for SmartStore^p (*plain* without security).

read a specific value, e.g. the sensor s_0 of Figure 1 says that the temperature is 22.5 C°, or that an aggregation function resulted in a specific value. We are only interested in the sequence of the operations performed, e.g. the sensor s_0 senses a value or the node ℓ_1 computes an average. For this reason, in the following we consider a symbolic semantics that is defined as the one described above but which differs from it only in the fact that the values read from sensors and computed by aggregation functions are symbolic.

The transition system generated by this enhanced semantics is defined below. Note that this transition system is *finite*, making our analysis easier.

Definition 2. An *(enhanced) transition system* is a quadruple $\langle \mathcal{N}, \Theta, \rightarrow, N_0 \rangle$, where $\mathcal{N}$ is the set of states (systems of nodes), Θ is the set of labels, $\rightarrow$ is the transition relation described in Table 1, and N_0 is the initial state.

3.3. The smart storehouse reloaded

To better clarify the constructs introduced above, we now provide the specification of our motivating example, expressed in IoT-LySa. While in Section 2, we only described the components of nodes, their roles in the network and which messages they exchange, here, we describe each node in details, by specifying their sensors, actuators and control processes.

A first IoT-LySa specification for our storehouse is in Figure 4. We call it SmartStore^p, and denote with N^p the system of its eight nodes. In the data collector nodes N_{S_i} for $i \in [0,3]$, each sensor S_{s_i} periodically senses the temperature and records it into its reserved location s_i in the store Σ_{s_i}, by using an assignment statement. The sensing operation works as an early input where v_{s_i} is the sampled temperature. After sensing, S_{s_i} performs some

internal operations (the τ action), which we are not interested in modelling, e.g. noise reduction of sensed data. Finally, it iterates its behaviour by a recursive invocation. Each process P_{s_i} iteratively performs the following actions:

- it waits for a *start message* from N_1, by using the receive primitive $(; x_{start_i})$, where the variable x_{start_i} will store the received value;
- it collects the data provided by the sensor S_{s_i} by simply accessing the corresponding store location s_i, and
- it sends them together with its label to the node N_1, by using the primitive $\langle\!\langle \ell_{s_i}, s_i \rangle\!\rangle \triangleright \{\ell_1\}$, before repeating its behaviour.

Note also that in the specification we are abusing of notation treating labels as values: actually, the node sends a literal value that denotes the relevant label.

In the node N_1, the process P_1 performs the following actions:

- it sends a *start* message to N_{s_0};
- it waits for its answer containing the sensed value by using the receive primitives $(\ell_{s_0}; z_0)$. The constant ℓ_{s_0} denotes the label of the sender from which the process is waiting for a message. Whereas the variable z_i is used to store the second value extracted from the received pair. Note that the receive primitive carries out pattern matching, where the term on the left of the semicolon is the constant ℓ_{s_0}. In practice, through this receive operation P_1 only accepts pairs whose first element is ℓ_{s_0}, when this is the case it takes the second element of the message and stores it in the variable z_0.
- P_1 interacts with N_{s_1}, N_{s_2}, and N_{s_3} by following the same communication schema. Note that we are modelling here a polling protocol similar to those used in master-slave communications, e.g. in Bluetooth [29].
- When all the messages are received, P_1 computes the average of the storehouse temperature with the function avg and sends it to the node N_3.
- Then, it waits for messages coming from the node N_3 (see below) and indicating how to adjust the temperature of the storehouse through the cooling system. As expected, $(; x_{ad})$ means that we perform no pattern matching (the part on the left of the semicolon is empty) and that the received data are stored in the variable x_{ad}.
- Finally, before repeating its behaviour, P_1 commands the actuator A_0 to execute the action just received.

In the specification of the node N_2, the sensor R_0 is defined following the same schema of the sensors S_i above, where r_0 is its reserved location.

- The control process P_2 waits for a *begin* message from N_3;
- it reads a value from R_0;

- it updates the stocktaking *db* with the function *update*;
- finally, before repeating its behaviour, it sends the updated stocktaking to the node N_3.

The node N_3 consists of a single control process P_3 that

- receives the data on the temperature from N_1;
- it sends a *begin* message to N_2 in order to receive data on the stocktaking;
- then, it sends a message to N_4 to log the temperature;
- afterwards, it checks whether the temperature is acceptable for the quantity and the kind of the stored food by using the function *ActDec*;
- depending on the result of this check, it sends a message to N_1 containing information about the proper actions to take on the cooling system.

We do not detail the specification of the node N_4: it is made of a single control process P_4 that receives data from N_3 and stores them into the Cloud running the continuation R_c, left abstract.

Of course, the specification SmartStore^p is not robust against an attacker since all communications are in clear. In our second scenario we assume to enable the nodes N_{s_1} and N_{s_3} to use cryptography for obtaining reliable data and to use the redundancy schema sketched in Section 2. The specification implementing this approach, call it SmartStore^c (the corresponding system of nodes is N^c), is shown in Figure 4. There are new versions of processes P_{s_i} for $i \in \{1, 3\}$, P_1 and P_3. In the specification, all the prefixes are enriched by tags (blue in the pdf), which are not part of the IoT-LySa language, but that will make the development of Sections 3 and 4 easier and more understandable. We assume that the cryptographic keys k_{s_1} and k_{s_3} are exchanged once and for all at deploy time and that they do not change during the execution, as it is often the case.

Now the processes P_{s_i} (with $i \in \{1, 3\}$) encrypt the data through the construct $\{\ell_{s_i}, s_i\}_{k_{s_i}}$, just before sending the message. Consequently, the new process P_1 of the node N_1 receives messages, decrypts them and performs consistency checks on the received data. The construct $(\{\ell_{s_i}, z_i\}_{k_{s_i}})$ is a shorthand meaning that the process receives an encrypted message, decrypts it by using the key k_{s_i} and if the pattern matching on ℓ_{s_i} succeeds the value extracted from the message is assigned to the variable z_i. Consistency checks are implemented by the function *cmp*. It compares temperatures coming from secure sensors with the those from insecure ones and returns `true` if data are as expected, and `false` otherwise. Then, the process sends the returned boolean together with the average results to the node N_3. The system is therefore designed in such a way that it can tolerate that part of the communications are compromised.

In turn, the specification of the process P_3 of N_3 is updated as follows. It receives a message consisting of two values and behaves differently depending on the first value, i.e. the result of consistency check made by N_1. The choice between the different behaviour is expressed by the construct $+$. This construct

means that the process receives a message, and then carries out pattern matching with the two guards ($\texttt{true}; x_{avg}$) and ($\texttt{false}; x_{avg}$) selecting the successful one. In our case, if the consistency check on the temperature failed the process P_3 sends a command γ_{check} to N_1 and raises an alarm to the Cloud (`false` branch) before sending the message to N_2, otherwise it behaves as in the previous version (`true` branch), sending an OK message to the Cloud.

Finally, another possible specification able to overcome the problem of an attacker intercepting and manipulating messages could be the following (call it $\widehat{N}^c$). Assume to use cryptography just in one sensor node, say N_{s_1}, and to perform consistency checks similar to the previous ones. In particular, the comparison function is now *halfcmp* and behaves as *cmp* above but it uses only two arguments. The specification of the new variant of our systems becomes $\widehat{N}^c$, where the modified components come with a hat ($\widehat{\ }$):

$$
\begin{array}{lcl}
\widehat{N}^c & = & N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid \widehat{N}_{s_3} \mid \widehat{N}_1 \mid N_2 \mid N_3 \mid N_4 \\
\widehat{N}_{s_3} & = & \ell_{s_i} : [\Sigma_{s_3} \parallel \widehat{P}_{s_3} \parallel S_{s_3}] \\
\widehat{P}_{s_3} & = & (; x_{start_3})^{s30}.\langle\!\langle \ell_{s_3}, s_3 \rangle\!\rangle \triangleright \{\ell_1\}^{s31}.\widehat{P}_{s_3} \\
\widehat{N}_1 & = & \ell_1 : [\Sigma_1 \parallel \widehat{P}_1 \parallel A_0] \\
\widehat{P}_1 & = & \langle\!\langle start_0 \rangle\!\rangle \triangleright \{\ell_{s_0}\}^{10}.(\ell_{s_0}; z_0)^{11}.\langle\!\langle start_1 \rangle\!\rangle \triangleright \{\ell_{s_1}\}^{12}.(\{\ell_{s_1}; z_1\}_{k_{s_1}})^{13}. \\
 & & \langle\!\langle start_2 \rangle\!\rangle \triangleright \{\ell_{s_2}\}^{14}.(\ell_{s_2}; z_2)^{15}.\langle\!\langle start_3 \rangle\!\rangle \triangleright \{\ell_{s_3}\}^{16}.(\ell_{s_3}; z_3)^{17}. \\
 & & \langle\!\langle halfcmp(z_0, z_1), avg(z_0, ..., z_3) \rangle\!\rangle \triangleright \{\ell_3\}^{18}. \\
 & & (; x_{ad})^{19}.\langle A_0, x_{ad} \rangle.\ ^{110}\widehat{P}_1
\end{array}
$$

3.4. Running the smart storehouse

We now show how the SmartStore^c system described in Figure 5 runs. To recall the association between transitions and involved prefixes, enhanced labels are indexed with the tags of the corresponding prefixes.

For brevity, we ignore the sensors actions, and put together some consecutive steps (this should be clear since we put multiple labels on the relation transition).

- The first block of transitions concerns N_1 that collects data about the temperature in the room by interacting with the sensor nodes N_{s_i}, receives the messages and decrypts the encrypted ones.

- The second block describes N_1 that compares the received values (we assume they are OK), computes their average and sends the results of its computations to N_3. Since the result of the checks done by N_1 is OK, the first branch of N_3 is selected. Node N_3 then asks N_2 for information on the stocktaking database. The sensor r_0 of the node N_2 senses a food box entering the storehouse, thus the control process P_2 updates its database *db* and sends it to the node N_3.

- In the third block, node N_3 receives information from N_2 concerning the updates of the stocktaking database. Then, it computes the actuation decision and sends it to N_1. The process P_1 of the node N_1 receives the decision and drives the actuator A_0 to carry out the right actuation.

$$
\begin{array}{lcl}
N^c & = & N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1 \mid N_2 \mid N_3 \mid N_4 \\
N_{s_i} & = & \ell_{s_i} : [\Sigma_{s_i} \parallel P_{s_i} \parallel S_{s_i}] \qquad \text{where } S_{s_i} = s_i := v_{s_i}.\tau.S_{s_i} \qquad i \in [0,3] \\
P_{s_i} & = & (; x_{start_i})^{si0}.\langle\!\langle \ell_{s_i}, s_i \rangle\!\rangle \triangleright \{\ell_1\}^{si1}.P_{s_i} \qquad i \in [0,2] \\
P_{s_i} & = & (; x_{start_i})^{si0}.\langle\!\langle \{\ell_{s_i}, s_i\}_{k_{s_i}} \rangle\!\rangle \triangleright \{\ell_1\}^{si1}.P_{s_i} \qquad i \in [1,3]
\end{array}
$$

$$
\begin{array}{lcl}
N_1 & = & \ell_1 : [\Sigma_1 \parallel P_1 \parallel A_0] \\
P_1 & = & \langle\!\langle start_0 \rangle\!\rangle \triangleright \{\ell_{s_0}\}^{10}.(\ell_{s_0}; z_0)^{11}.\langle\!\langle start_1 \rangle\!\rangle \triangleright \{\ell_{s_1}\}^{12}.(\{\ell_{s_1}; z_1\}_{k_{s_1}})^{13}. \\
 & & \langle\!\langle start_2 \rangle\!\rangle \triangleright \{\ell_{s_2}\}^{14}.(\ell_{s_2}; z_2)^{15}.\langle\!\langle start_3 \rangle\!\rangle \triangleright \{\ell_{s_3}\}^{16}.(\{\ell_{s_3}; z_3\}_{k_{s_3}})^{17}. \\
 & & \langle\!\langle cmp(z_0,z_1,z_2,z_3), avg(z_0,z_1,z_2,z_3) \rangle\!\rangle \triangleright \{\ell_3\}^{18}. \\
 & & (; x_{ad})^{19}.\langle A_0, x_{ad} \rangle.\ {}^{110}P_1
\end{array}
$$

$$
\begin{array}{lcl}
N_2 & = & \ell_2 : [\Sigma_2 \parallel P_2 \parallel R_0] \qquad \text{where } R_0 = r_0 := w.\tau.R_0 \\
P_2 & = & (; x_{begin_2})^{20}.db := update(db, r_0)^{21}.\langle\!\langle db \rangle\!\rangle \triangleright \{\ell_3\}^{22}.P_2
\end{array}
$$

$$
\begin{array}{lcl}
N_3 & = & \ell_3 : [\Sigma_3 \parallel P_3] \\
P_3 & = & (\texttt{true}; x_{avg})^{300}.\langle\!\langle begin_2 \rangle\!\rangle \triangleright \{\ell_2\}^{310}.(; x_{db})^{320}.\langle\!\langle actDec(x_{avg}, x_{db}) \rangle\!\rangle \triangleright \{\ell_1\}^{330}. \\
 & & \langle\!\langle \texttt{ok}, x_{avg} \rangle\!\rangle \triangleright \{\ell_4\}^{340}.P_3 \ + \\
 & & (\texttt{false}; x_{avg})^{301}.\langle\!\langle begin_2 \rangle\!\rangle \triangleright \{\ell_2\}^{311}.(; x_{db})^{321}.\langle\!\langle \gamma_{check} \rangle\!\rangle \triangleright \{\ell_1\}^{331}. \\
 & & \langle\!\langle \texttt{alarm}, x_{avg} \rangle\!\rangle \triangleright \{\ell_4\}^{341}.P_3
\end{array}
$$

$$
\begin{array}{lcl}
N_4 & = & \ell_4 : [\Sigma_4 \parallel P_4] \\
P_4 & = & (; w_{res}, w_{avg})^{40}.R_c.P_4
\end{array}
$$

Figure 4: An IoT-LySa specification for SmartStore^c (with security).

- Finally, in the fourth block, N_3 sends an ok message to N_4. Note that the initial system evolves but it will always finally return to its initial form.

Consider the second version of our storehouse system. The run corresponding to the one above is similar except for the following actions concerning the first part where the node N_1 collects data:

$$
\begin{array}{l}
\widehat{N}^c \xrightarrow{\hat{\theta}_{10}} \xrightarrow{\hat{\theta}_{S00}} \ldots \xrightarrow{\hat{\theta}_{S31}} \ldots \xrightarrow{\hat{\theta}_{17}} \xrightarrow{\hat{\theta}_{18}} \\
N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_3
\end{array}
$$

In particular,

$$
\begin{array}{lcl}
\hat{\theta}_{11} & = \hat{\theta}_{15} = \hat{\theta}_{17} = \ell_{s_i} \triangleright \ell_1 \langle \langle\!\langle \ell_{s_i}, v_i \rangle\!\rangle, (\ell_{s_i}; z_i) \rangle & i \in \{0,2,3\} \\
\hat{\theta}_{18} & = \ell_1 \langle \langle\!\langle halfcmp(v_0, v_1), avg(v_0, \cdots, v_3) \rangle\!\rangle \rangle & \\
\hat{\theta}_{s00} & = \hat{\theta}_{s20} = \hat{\theta}_{s30} = \ell_1 \triangleright \ell_{s_i} \langle \langle\!\langle start_i \rangle\!\rangle, (; x_{start_i}) \rangle & \\
\hat{\theta}_{tag} & = \theta_{tag} \text{ in all the other cases} &
\end{array}
$$

4. Stochastic Semantics

We now show how to generate a Continuous Time Markov Chain (CTMC) from a transition system (see [15] for more details). First, we introduce functions

$$
\begin{aligned}
N^c = {} & N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1 \mid N_2 \mid N_3 \mid N_4 \\
& \xrightarrow{\theta_{10}}\xrightarrow{\theta_{S00}}\xrightarrow{\theta_{S01}}\xrightarrow{\theta_{11}}\xrightarrow{\theta_{12}}\xrightarrow{\theta_{S10}}\xrightarrow{\theta_{S11}}\xrightarrow{\theta_{13}}\xrightarrow{\theta_{14}}\xrightarrow{\theta_{S20}}\xrightarrow{\theta_{S21}}\xrightarrow{\theta_{15}}\xrightarrow{\theta_{16}}\xrightarrow{\theta_{S30}}\xrightarrow{\theta_{S31}}\xrightarrow{\theta_{17}} \\
& N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{VIII} \mid N_2 \mid N_3 \mid N_4 \\
& \xrightarrow{\theta_{18}}\xrightarrow{\theta_{30i}}\xrightarrow{\theta_{31i}}\xrightarrow{\theta_{20}}\xrightarrow{\theta_{21}}\xrightarrow{\theta_{22}} \\
& N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_2 \mid N_{30}^{III} \mid N_4 \\
& \xrightarrow{\theta_{320}}\xrightarrow{\theta_{330}}\xrightarrow{\theta_{340}} \\
& N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_2 \mid N_3 \mid N_4 \\
& \xrightarrow{\theta_{19}}\xrightarrow{\theta_{20}} \\
& N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1 \mid N_2 \mid N_3 \mid N_4 \\
& \xrightarrow{\theta_{40}} \\
& N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1 \mid N_2 \mid N_3 \mid N_4^I
\end{aligned}
$$

where

$$
\begin{aligned}
\theta_{10} &= \theta_{12} = \theta_{14} = \theta_{16} = \ell_1\langle\langle\langle start_i\rangle\rangle\rangle && i \in [0,3] \\
\theta_{si0} &= \ell_1 \triangleright \ell_{s_i}\langle\langle\langle start_i\rangle\rangle, (; x_{start_i})\rangle && i \in [0,3] \\
\theta_{si1} &= \ell_{s_i}\langle\langle \ell_{s_i}, s_i\rangle\rangle && i \in \{0,2\} \\
\theta_{si1} &= \ell_{s_i}\langle\langle \{\ell_{s_i}, s_i\}_{k_{s_i}}\rangle\rangle && i \in \{1,3\} \\
\theta_{11} &= \theta_{15} = \ell_{s_i} \triangleright \ell_1\langle\langle\langle \ell_{s_i}, v_i\rangle\rangle, (\ell_{s_i}; z_i)\rangle && i \in \{0,2\} \\
\theta_{13} &= \theta_{17} = \ell_{s_i} \triangleright \ell_1\langle\langle\langle \{\ell_{s_i}, v_i\}_{k_{s_i}}\rangle\rangle, (\{\ell_{s_i}; z_i\}_{k_{s_i}})\rangle && i \in \{1,3\} \\
&\quad \text{where } v_i = [\![s_i]\!]_{\Sigma_{s_i}} &&
\end{aligned}
$$

$$
\begin{aligned}
\theta_{18} &= \ell_1\langle\langle\langle cmp(v_0, \cdots, v_3), avg(v_0, \cdots, v_3)\rangle\rangle\rangle \\
\theta_{300} &= \ell_1 \triangleright \ell_3\langle\langle\langle \texttt{true}, v_m\rangle\rangle, (\texttt{true}; x_{avg})\rangle \\
&\quad \text{where } \texttt{true} = [\![cmp(z_0, ..., z_3)]\!]_{\Sigma_1^{VIII}} \wedge v_m = [\![avg(z_0, ..., z_3)]\!]_{\Sigma_1^{VIII}}
\end{aligned}
$$

$$
\begin{aligned}
\theta_{310} &= \ell_3\langle\langle\langle begin_2\rangle\rangle\rangle \\
\theta_{20} &= \ell_3 \triangleright \ell_2\langle\langle\langle begin_2\rangle\rangle, (; x_{begin_2})\rangle \\
\theta_{21} &= \ell_2\langle store(update(db, r_0))\rangle \\
\theta_{22} &= \ell_2\langle\langle\langle db\rangle\rangle\rangle
\end{aligned}
$$

$$
\begin{aligned}
\theta_{320} &= \ell_2 \triangleright \ell_3\langle\langle\langle v_{db}\rangle\rangle, (x_{db})\rangle \\
&\quad \text{where } v_{db} = [\![db]\!]_{\Sigma_2^{II}} \\
\theta_{330} &= \ell_3\langle\langle\langle ActDec(x_{avg}, x_{db})\rangle\rangle\rangle \\
\theta_{340} &= \ell_3\langle\langle\langle \texttt{ok}, x_{avg}\rangle\rangle\rangle \\
\theta_{19} &= \ell_3 \triangleright \ell_1\langle\langle\langle v_{ad}, (; x_{ad})\rangle\rangle\rangle \\
&\quad \text{where } v_{ad} = [\![ActDec(x_{avg}, x_{db})]\!]_{\Sigma_3'} \\
\theta_{110} &= \ell_1\langle do(A_0, v_{ad})\rangle
\end{aligned}
$$

$$
\begin{aligned}
\theta_{40} &= \ell_3 \triangleright \ell_4\langle\langle\langle \texttt{ok}, v_{avg}\rangle\rangle, (; w_{res}, w_{avg})\rangle \\
&\quad \text{where } v_{avg} = [\![x_{avg}]\!]_{\Sigma_3^I}
\end{aligned}
$$

Figure 5: A run of the SmartStore^c system.

on the enhanced labels to associate costs with transitions. In general costs can be measured in terms of the resources of interest. Here, we focus on the time. In particular, the cost of a system is specified in terms of the *time* spent on transitions, and it depends on the performed action as well as on the involved nodes. Intuitively, cost functions define exponential distributions, from which we compute the rates at which a system evolves and the corresponding CTMC. Then, to evaluate the performance we calculate the stationary distribution of the CTMC and the transition rewards. Moreover, we define a bisimulation-based equivalence that can be exploited to compare the performance of different specifications. In the next section, we will consider the energy.

4.1. Cost Functions

Our cost functions assign a *rate* to each transition with label $\theta \in \Theta$. To define this rate, we suppose to execute each action on a dedicated architecture that only performs that action, and we estimate the corresponding duration. To model the performance degradation due to the run-time support, we introduce a scaling factor for each routine called by the implementation under consideration. Here, we just propose a format for these functions, with parameters that depend on the nodes to be instantiated on need. For instance, in a node where the cryptographic operations are performed at very high speed (e.g. by a cryptographic accelerator), but with a slow link (low bandwidth), the time will be low for encryptions and high for communication; vice versa, in a node offering a high bandwidth, but poor cryptography resources, the required time will be high for encryptions and low for communication.

Technically, we interpret costs as parameters of exponential distributions $F(t) = 1 - e^{-rt}$, with *rate* r and t as time parameter (general distributions are also possible see [30]): the transition rate r is the parameter that identifies the exponential distribution of the duration times of the transition, as usual in stochastic process algebras (e.g. [31]). The shape of $F(t)$ is a curve that grows from 0 asymptotically approaching 1 for positive values of its argument t. The parameter r determines the slope of the curve: the greater the rate r, the faster the time in which $F(t)$ approaches its asymptotic value, and therefore the faster the speed of the corresponding transition. The exponential distributions that we use enjoy the *memoryless property*, i.e. the occurrence of a new transition does not depend on when the last transition occurred. We also assume that transitions are *time homogeneous*, i.e. that the corresponding rates do not depend on the time at which they occur.

We first associate costs with terms, and then we define the functions that associates rates with the (enhanced labels of) transitions. Even though we are mainly interested in the costs of communications and in the computational ones due to the application of aggregation functions and of encryption and decryption, we initially associate costs with every possible activity. We need the auxiliary

function $f_E \colon \mathcal{E} \to \mathbb{R}^+$ that estimates the effort needed to manipulate terms.

- $f_E(v) = f_{load}(v)$
- $f_E(i) = f_{load}(i)$
- $f_E(x) = f_{load}(x)$
- $f_E(fun(E_1, ..., E_m)) = f_{fun}(f_E(E_1), ..., f_E(E_m))$
- $f_E(\{E_1, \ldots, E_m\}_{k_0}) = f_{enc}(f_E(E_1), ..., f_E(E_m), crypto\_system, kind(k_0))$

Manipulating a non composed term has the cost of loading it, while the cost of a composed term depends on the cost of its components and on the particular function applied to it. In the case of encryption, we use the function f_{enc}, which also depends on the used crypto-system and on the kind (short/long, short-term/long-term) of the key.

We now present the functions $\$_\alpha \colon \Theta_B \to \mathbb{R}^+$ and $\$_t \colon \Theta \to \mathbb{R}^+$ to assign costs to enhanced labels.

- $\$_\alpha(sens(i)) = f_{sens}(i)$
- $\$_\alpha(store(E)) = f_{store}(E)$
- $\$_\alpha(\langle\!\langle E_1, ..., E_m \rangle\!\rangle) = f_{out}(f_E(E_1), ..., f_E(E_m), bw)$
- $\$_\alpha(do(j, \gamma)) = f_{trig}(j)$
- $\$_\alpha(act(\gamma)) = f_{act}(\gamma)$
- $\$_\alpha(int) = \lambda$
- $\$_\alpha(dec(E, \{E'_1, \cdots, E'_j; x_{j+1}, \cdots, x_m\}_{k_0})) =$
  $f_{dec}(f_E(E), f_E(E_1), ..., f_E(E_j), crypto\_system, kind(K_0), patternMatch(j))$

- $\$_t(\ell\ \theta_B) = f_{()}(\ell) \cdot \$_\alpha(\theta_B)$
- $\$_t(\ell_O \triangleright \ell_I\ \langle\langle\!\langle E_1, ..., E_m \rangle\!\rangle, (E_1, \cdots, E_j; x_{j+1}, \cdots, x_m)\rangle) =$
  $f_{<>}(\ell_O, \ell_I) \cdot f_{in}(f_E(E_1), ..., f_E(E_j), patternMatch(j), bw)$

where the functions f_{sens}, f_{store}, f_{trig}, f_{act} define the costs of the routines that implement sensing, storing, triggering, and actuating; while λ is the cost of an internal action. The functions f_{out} and f_{in} define the costs of the routines that implement the send and receive primitives. Besides the implementation cost due to their own algorithms, the functions above depend on the bandwidth of the channel (represented by bw), on the cost of the exchanged terms (computed by f_E), and on the nodes involved in the communication. Also, the cost of an input depends on the number of required matchings (represented by $match(j)$). The function f_{dec} defines the cost of the decryption routine: its cost is similar to the one for encryption, with the additional cost of matchings.

The costs associated to labels θ_B only take into account the low-level operations corresponding to the actions, independently of the context, i.e. the node in which they are performed. The cost associated to a label $\ell\ \theta_B$ also considers in which node the action is performed, by using the function $f_{()}(\ell)$. Nodes can indeed have different costs for the same activity.

Finally, in the inter-node communication the two partners independently perform some low-level operations locally to their nodes, labelled ℓ_O and ℓ_I. Hence, the cost also depends on the nodes involved in the communication, represented

$$
\begin{array}{lcl}
N^s & = & N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1 \mid N_2 \mid N_3 \mid N_4 \\
N_{s_i} & = & \ell_{s_i} : [\Sigma_{s_i} \parallel P_{s_i} \parallel S_{s_i}] \qquad \text{where } S_{s_i} = s_i := v_{s_i}.\tau.S_{s_i} \qquad i \in [0,3] \\
P_{s_i} & = & (;x_{start_i})^{si0}.\langle\!\langle \ell_{s_i}, s_i \rangle\!\rangle \triangleright \{\ell_1\}^{si1}.P_{s_i} \qquad i \in [0,2] \\
P_{s_i} & = & (;x_{start_i})^{si0}.\langle\!\langle \{\ell_{s_i}, s_i\}_{k_{s_i}} \rangle\!\rangle \triangleright \{\ell_1\}^{si1}.P_{s_i} \qquad i \in [1,3]
\end{array}
$$

$$
\begin{array}{lcl}
N_1 & = & \ell_1 : [\Sigma_1 \parallel P_1 \parallel A_0] \\
P_1 & = & \langle\!\langle start_0 \rangle\!\rangle \triangleright \{\ell_{s_0}\}^{10}.(\ell_{s_0}; z_0)^{11}.\langle\!\langle start_1 \rangle\!\rangle \triangleright \{\ell_{s_1}\}^{12}.(\{\ell_{s_1}; z_1\}_{k_{s_1}})^{13}. \\
 & & \langle\!\langle start_2 \rangle\!\rangle \triangleright \{\ell_{s_2}\}^{14}.(\ell_{s_2}; z_2)^{15}.\langle\!\langle start_3 \rangle\!\rangle \triangleright \{\ell_{s_3}\}^{16}.(\{\ell_{s_3}; z_3\}_{k_{s_3}})^{17}. \\
 & & \langle\!\langle cmp(z_0,z_1,z_2,z_3), avg(z_0,z_1,z_2,z_3) \rangle\!\rangle \triangleright \{\ell_3\}^{18}. \\
 & & (;x_{ad})^{19}.\langle A_0, x_{ad}\rangle.{}^{110}P_1
\end{array}
$$

$$
\begin{array}{lcl}
N_3 & = & \ell_3 : [\Sigma_3 \parallel P_3] \\
P_3 & = & (\texttt{true}; x_{avg})^{300}.\langle\!\langle actDec(x_{avg}) \rangle\!\rangle \triangleright \{\ell_1\}^{330}.P_3 + \\
 & & (\texttt{false}; x_{avg})^{301}.\langle\!\langle \gamma_{check} \rangle\!\rangle \triangleright \{\ell_1\}^{331}.P_3
\end{array}
$$

Figure 6: The simplified version of IoT-LySa specification for SmartStore^c (with security).

here by the function $f_{<>}(\ell_O, \ell_I)$. In particular, it may depend on the level of interference in the neighbourhood of the nodes that, in turn, affects the duration of the channel arbitration at the MAC layer of the communication subsystem. It could also depend on other parameters such as the communication medium, latency and so on.

These costs represent the time spent performing the corresponding action, with the intuition that the lower the rate, the greater the time needed to complete an action and therefore the slower the speed of the transition.

For simplicity, we ignore the costs for other primitives, e.g. constant invocation, parallel composition, summation (see [15] for a complete treatment).

Note that we do not fix the actual cost function: we only propose for it a set of parameters to reflect some features of an idealised architecture. Although very abstract, this suffices to make our point. A precise instantiation comes with the refinement steps from specification to implementations as soon as actual parameters become available.

4.2. Smart storehouse example (cont'd)

In order to make our performance analysis easier to be presented, we will consider the simplified version of our system SmartStore^c (still denoted with N^c for simplicity), as described in Figure 6, where the tags coincide with the ones in the corresponding prefixes in Figure 4. The whole enhanced transition system follows, where transitions with tags $s30i$ and $s33i$ succinctly represent the transitions tagged with $s300$, $s330$, and $s301$, $s331$, respectively.

$$
\begin{aligned}
&N^c \xrightarrow{\theta_{10}}\xrightarrow{\theta_{S00}}\xrightarrow{\theta_{S01}}\xrightarrow{\theta_{11}}\xrightarrow{\theta_{12}}\xrightarrow{\theta_{S10}}\xrightarrow{\theta_{S11}}\xrightarrow{\theta_{13}}\xrightarrow{\theta_{14}}\xrightarrow{\theta_{S20}}\xrightarrow{\theta_{S21}}\xrightarrow{\theta_{15}}\xrightarrow{\theta_{16}}\xrightarrow{\theta_{S30}}\xrightarrow{\theta_{S31}}\xrightarrow{\theta_{17}}\xrightarrow{\theta_{18}}\\
&N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_3 \xrightarrow{\theta_{30i}} \begin{cases} N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_{30}^{I} & \text{if } i = 0 \\ N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_{31}^{I} & \text{if } i = 1 \end{cases}\\
&\xrightarrow{\theta_{33i}} N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_3 \qquad i \in \{0,1\}\\
&\xrightarrow{\theta_{19}}\xrightarrow{\theta_{110}}\\
&N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1 \mid N_3 = N^c
\end{aligned}
$$

Analogously, we consider the simplified version of the second process $\widehat{N}^c$. Its evolution is similar to the one of N^c. In particular:

$$
\begin{aligned}
\hat{\theta}_{s31} &= \ell_{s_3}\langle\langle \ell_{s_3}, s_3 \rangle\rangle \\
\hat{\theta}_{17} &= \ell_{s_1} \triangleright \ell_1\langle\langle\{\ell_{s_1}, v_1\}_{k_{s_1}}\rangle\rangle, (\{\ell_{s_1}; z_1\}_{k_{s_1}})\rangle \\
\hat{\theta}_{18} &= \ell_1\langle\langle cmp(v_0, \cdots, v_3), avg(v_0, \cdots, v_3)\rangle\rangle\rangle \\
\hat{\theta}_{tag} &= \theta_{tag} \quad \text{in all the other cases}
\end{aligned}
$$

We now associate a rate with each transition in the transition system of N^c. To illustrate our methodology, we assume that the coefficients due to the nodes amount to 1, i.e. $f_{()}(\ell) = 1$ and $f_{<>}(\ell, \ell') = 1$ for each ℓ, ℓ' in $\mathcal{L}$.

We instantiate the cost functions defined above, in terms of the following symbolic parameters: (i) `lo` (`st`) for the loading (storing) of a simple, i.e. not composed term E_{simple}; (ii) `f` for the application of the aggregate function fun, whose cost is proportional to the number of its arguments; (iii) `e` and `d` for encrypting and for decrypting; (iv) `s` and `r` for sending and for receiving; (v) `ma` for pattern matching; and (vi) `t` for commanding an action to an actuator; (vii) σ and α for sensing and actuating. Given these parameters, the instantiation of the cost functions is the following.

- $f_E(E_{simple}) = \texttt{lo}$
- $f_E(fun(E_1, \ldots, E_m) = \sum_{i=1}^{m} f_E(E_i) + m \cdot \texttt{f}$
- $f_E(\{E_1, \ldots, E_m\}_{k_0}) = \sum_{i=1}^{m} f_E(E_i) + m \cdot \texttt{e}$

- $\$_t(\ell\ sens) = \sigma$
- $\$_t(\ell\ act(x_{ad})) = \alpha$
- $\$_t(\ell\ do(j, \gamma)) = \texttt{t}$
- $\$_t(\ell\ store(E)) = f_E(E) + \texttt{st}$
- $\$_t(\ell\ \langle\langle E_1, \ldots, E_m\rangle\rangle) = \sum_{i=1}^{m} f_E(E_i) + m \cdot \texttt{s}$
- $\$_t(\ell_O \triangleright \ell_I\ \langle\langle v_1, \ldots, v_m\rangle\rangle, (E_1, \cdots, E_j; x_{j+1}, \cdots, x_m)\rangle) =$
  $\sum_{i=1}^{j} f_E(E_i) + m \cdot \texttt{r} + j \cdot \texttt{ma}$

- $\$_t(\ell\ \langle\langle\{E_1, E_2\}_k)\rangle\rangle) = f_E(\{E_1, E_2\}_k) + \texttt{s}$
- $\$_t(\ell_O \triangleright \ell_I\ \langle\langle\{E_1, E_2\}_k\rangle\rangle, (\{E_1; x_2\}_k)\rangle) = f_E(E_1) + \texttt{r} + 2 \cdot \texttt{d} + \texttt{ma}$

For the sake of simplicity, we ignore the costs of sensing from the environment and the cost of taking actions of actuators. Both costs in our scenario can be covered by the remaining computational costs.

The rates of the transitions of N^c and $\widehat{N}^c$ are $c_{tag} = \$_t(\theta_{tag})$ and $\hat{c}_{tag} = \$_t(\hat{\theta}_{tag})$ and are described below. Recall that the greater the rate, the faster the speed of the transition.

$$
\begin{array}{lll}
c_{10} = c_{12} = c_{14} = c_{16} = \$_t(\ell_1\langle\langle\langle start_i\rangle\rangle\rangle) = \frac{1}{\texttt{lo+s}} & & i \in [0,3] \\
c_{si0} = \$_t(\ell_1 \triangleright \ell_{s_i}\langle\langle\langle start_i\rangle\rangle, (; x_{start_i})\rangle) = \frac{1}{\texttt{lo+r}} & & i \in [0,3] \\
c_{si1} = \$_t(\ell_{s_i}\langle\langle\ell_{s_i}, s_i\rangle\rangle) = \frac{1}{2\cdot\texttt{(lo+s)}} & & i \in \{0,2\} \\
c_{si1} = \$_t(\ell_{s_i}\langle\langle\{\ell_{s_i}, s_i\}_{k_{s_i}}\rangle\rangle) = \frac{1}{2\cdot\texttt{(lo+e)+s}} & & i \in \{1,3\} \\
c_{11} = c_{15} = \$_t(\ell_{s_i} \triangleright \ell_1\langle\langle\langle\ell_{s_i}, v_i\rangle\rangle, (\ell_{s_i}; z_i)\rangle) = \frac{1}{\texttt{lo+2·r+ma}} & & i \in \{0,2\} \\
c_{13} = c_{17} = \$_t(\ell_{s_i} \triangleright \ell_1\langle\langle\langle\{\ell_{s_i}, v_i\}_{k_{s_i}}\rangle\rangle, (\{\ell_{s_i}; z_i\}_{k_{s_i}})\rangle) = \frac{1}{\texttt{lo+r+2·d+ma}} & & i \in \{1,3\} \\
c_{18} = \$_t(\ell_1\langle\langle\langle cmp(z_0,\cdots,z_3), avg(z_0,\cdots,z_3)\rangle\rangle\rangle) = \frac{1}{8\cdot\texttt{(lo+f)+2·s}} & &
\end{array}
$$

$$
\begin{array}{l}
c_{30i} = \$_t(\ell_1 \triangleright \ell_3\langle\langle\langle\texttt{bool}_\texttt{i}, v_m\rangle\rangle, (bool_i; x_{avg})\rangle) = \frac{1}{\texttt{lo+m+2·r}} \\
c_{330} = \$_t(\ell_3\langle\langle\langle ActDec(x_{avg})\rangle\rangle\rangle) = \frac{1}{\texttt{lo+f+s}} \\
c_{331} = \$_t(\ell_3\langle\langle\langle\gamma_{check}\rangle\rangle\rangle) = \frac{1}{\texttt{lo+s}}
\end{array}
$$

$$
\begin{array}{l}
c_{19} = \$_t(\ell_3 \triangleright \ell_1\langle\langle\langle v_{ad}\rangle\rangle, (; x_{ad})\rangle) = \frac{1}{\texttt{lo+r}} \\
c_{110} = \$_t(\ell_1\langle do(A_0, v_{ad})\rangle) = \frac{1}{\texttt{t}}
\end{array}
$$

$$
\begin{array}{l}
\hat{c}_{s31} = \$_t(\ell_{s_3}\langle\langle\ell_{s_3}, s_3\rangle\rangle) = \frac{1}{2\cdot\texttt{(lo+s)}} \\
\hat{c}_{17} = \$_t(\ell_{s_1} \triangleright \ell_1\langle\langle\langle\{\ell_{s_1}, v_1\}_{k_{s_1}}\rangle\rangle, (\{\ell_{s_1}; z_1\}_{k_{s_1}})\rangle) = \frac{1}{\texttt{lo+r+2·d+ma}} \\
\hat{c}_{18} = \$_t(\ell_1\langle\langle\langle halfcmp(z_0, z_1), avg(z_0,\cdots,z_3)\rangle\rangle\rangle) = \frac{1}{6\cdot\texttt{(lo+f)+2·s}} \\
\hat{c}_{tag} = c_{tag} \quad \text{in all the other cases}
\end{array}
$$

4.3. Stochastic Analysis

By using the above rates, we can now transform the transition system N into its corresponding $CTMC(N)$. We recall that the exponential distributions we use enjoy the memoryless property and are time-homogeneous. Afterwards, we can calculate the actual performance measures, by using reward structures (see [32] for more details on the theory of stochastic processes).

Markov Chains. First, we summarise some notions used in the CMTC derivation. We start by the definition of *transition rate*, i.e. the rate at which a system evolves from N_i to N_j that amounts to the sum of the single rates θ_k of all the possible transitions from N_i to N_j (if N_j cannot be reached in one step, the rate is 0).

Definition 3. The *transition rate* $q(N_i, N_j)$ between two systems N_i and N_j is defined as follows

$$q(N_i, N_j) = \sum_{\theta_k} \$_t(\theta_k) \text{ where } N_i \xrightarrow{\theta_k} N_j$$

Given a transition system N, the corresponding CTMC has a state for each node in N, and the arcs between states are obtained by coalescing all the arcs with the same source and target in N. The rates at which the system evolves from one state to another can be arranged in a matrix $\mathbf{Q}$, called the *generator*

matrix. Apart from its diagonal, it corresponds to the adjacency matrix of the graph that represents the CTMC considered. Note that $q(N_i, N_j)$ coincides with the off-diagonal element q_{ij} of the generator matrix $\mathbf{Q}$. Hence, hereafter we will use both CTMC and its corresponding $\mathbf{Q}$ to denote a Markov chain.

Definition 4. Given a system N and its transition system. Then, the *generator matrix* Q of the continuous time Markov chain of N (called $CTMC(N)$) is a $[n \times n]$ square matrix with elements q_{ij} defined as follows.

$$q_{ij} = \begin{cases} q(N_i, N_j) = \sum\limits_{\theta_k} \$_t(\theta_k) & \text{if } i \neq j \ \wedge \ N_i \xrightarrow{\theta_k} N_j \\ -\sum\limits_{j=0, j\neq i}^{n} q_{ij} & \text{if } i = j \end{cases}$$

Each entry q_{ij} defines the instantaneous transition rate from N_i to N_j in terms of the transitions outgoing from N_i. The total transition rate between two states is the sum of the rates of the transitions connecting the corresponding nodes in the transition system. The diagonal entries are such that the sum of all rows is zero. Hence, we can directly define the graph CTMC associated with a system.

Definition 5. Given a transition system associated with a system N, we define the *continuous time Markov chain* $CTMC(N)$ of N as the labelled transition system, whose transition relation is the minimal relation defined as follows

$$\frac{N_i \xrightarrow{\theta_k} N_j}{N_i \xrightarrow{q_{ij}} N_j}$$

where q_{ij} is defined as in Definition 4.

It is possible to show (see [15]) that the definitions above are correct, i.e. that they coincide (apart from the diagonal) and that $CTMC(N)$ is actually a continuous time Markov chain.

The conditional transition rate from N_i to N_j, via an action labelled θ, is the sum of the activity rates decorating arcs that connect the corresponding nodes in the derivation graph which are also labelled by the action of kind θ. It is the rate at which a system that behaves as component N_i evolves to behaving as component N_j as the result of completing an activity of kind θ.

Definition 6. The *conditional transition* rate $q(N_i, \theta, N_j)$ between two systems N_i and N_j is defined as follows

$$q(N_i, \theta, N_j) = \sum_{N_i \xrightarrow{\theta} N_j} \$_t(\theta)$$

Evaluating the Performance. Performance analysis usually has to do with the behaviour of systems over long periods of time, necessary for the system to reach a sort of regular pattern of behaviour. Statistically speaking, the system reaches the equilibrium. For this reason, to evaluate the performance of N, we need to obtain the stationary distribution for the $CMTC(N)$.

Definition 7. A Continuous Time Markov Chain has a *stationary probability distribution* $\Pi = (X_0, \ldots, X_{n-1})$ if

$$\Pi^T \mathbf{Q} = \mathbf{0} \quad \wedge \quad \sum_{i=0}^{n} X_i = 1$$

The stationary probability distribution can be found by using the normalisation condition $\sum_{i=0}^{n} X_i = 1$ and the global balance equations $\Pi^T \mathbf{Q} = \mathbf{0}$ (the rate of flow coming out of each state is balanced by the rate of flow entering the state).

To have a *unique* stationary distribution $CTMC(N)$ must be time homogeneous and irreducible (i.e. all states can be reached from all other states). The first condition is clearly satisfied. The second condition is fulfilled if all the states of $CTMC(N)$ are positive recurrent (a state is positive recurrent if the probability of the system to return to the state is 1 and the number of the needed steps is finite). Since a derivative N' of N corresponds to a state of $CTMC(N)$, it suffices then that N' is reachable by any of its derivatives through a finite sequence of transitions because we only consider finite state processes. So, we will restrict our attention to systems in this form.

4.4. Smart storehouse example (cont'd)

Back to both the systems of our running example, we first consider the transition system corresponding to N^c that is, as required above, finite and with a cyclic initial state. Therefore, it has stationary distribution.

We derive the generator matrix $\mathbf{Q_1}$ of $CTMC(N^c)$, described in Figure 7, where, for simplicity, we use the parameters $a, b, c, ..., w$ to refer to the inverses of the rates of our system (in the transition system below, we put the parameters as indexes of the arrows with the corresponding rates), e.g. $c_{10} = \frac{1}{\mathtt{1o+s}} = \frac{1}{a}$.

$$N^c \xrightarrow{c_{10}}_a \xrightarrow{c_{S00}}_b \xrightarrow{c_{S01}}_c \xrightarrow{c_{11}}_d \xrightarrow{c_{12}}_e \xrightarrow{c_{S10}}_f \xrightarrow{c_{S11}}_g \xrightarrow{c_{13}}_h \xrightarrow{c_{14}}_i \xrightarrow{c_{S20}}_j \xrightarrow{c_{S21}}_k \xrightarrow{c_{15}}_l \xrightarrow{c_{16}}_m \xrightarrow{c_{S30}}_n \xrightarrow{c_{S31}}_o \xrightarrow{c_{17}}_p \xrightarrow{c_{18}}_q$$

$$N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_3 \xrightarrow{c_{30i}}_{r/s} \begin{cases} N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_{30}^{I} & \text{if } i = 0 \\ N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_{31}^{I} & \text{if } i = 1 \end{cases}$$

$$\xrightarrow{c_{33i}}_{t/u} N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_3$$

$$\xrightarrow{c_{19}}_v \xrightarrow{c_{110}}_w$$

$$N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1 \mid N_3 = N^c$$

The corresponding stationary distribution Π_1 is the solution of the following system of symbolic linear equations,

$$
\begin{array}{rr}
\frac{1}{w}x_{21} - \frac{1}{a}x_0 = 0; & \frac{1}{k}x_{10} - \frac{1}{l}x_{11} = 0; \\
\frac{1}{a}x_0 - \frac{1}{b}x_1 = 0; & \frac{1}{l}x_{11} - \frac{1}{m}x_{12} = 0; \\
\frac{1}{b}x_1 - \frac{1}{c}x_2 = 0; & \frac{1}{m}x_{12} - \frac{1}{n}x_{13} = 0; \\
\frac{1}{c}x_2 - \frac{1}{d}x_3 = 0; & \frac{1}{n}x_{13} - \frac{1}{o}x_{14} = 0; \\
\frac{1}{d}x_3 - \frac{1}{e}x_4 = 0; & \frac{1}{o}x_{14} - \frac{1}{p}x_{15} = 0; \\
\frac{1}{e}x_4 - \frac{1}{f}x_5 = 0; & \frac{1}{p}x_{15} - \frac{1}{q}x_{16} = 0; \\
\frac{1}{f}x_5 - \frac{1}{g}x_6 = 0; & \frac{1}{q}x_{16} - (\frac{1}{r} + \frac{1}{s})x_{17} = 0; \\
\frac{1}{g}x_6 - \frac{1}{h}x_7 = 0; & \frac{1}{r}x_{17} - \frac{1}{t}x_{18} = 0; \\
\frac{1}{h}x_7 - \frac{1}{i}x_8 = 0; & \frac{1}{s}x_{17} - \frac{1}{u}x_{19} = 0; \\
\frac{1}{i}x_8 - \frac{1}{j}x_9 = 0; & \frac{1}{t}x_{18} + \frac{1}{u}x_{19} - \frac{1}{v}x_{20} = 0; \\
\frac{1}{j}x_9 - \frac{1}{k}x_{10} = 0; & \frac{1}{v}x_{20} - \frac{1}{w}x_{21} = 0;
\end{array}
\qquad
\begin{array}{l}
\sum_{i=0}^{21} x_i = 1; \\
a, b, c, ..., w \neq 0;
\end{array}
$$

computed by exploiting the computer algebra package Mathematica [35]. The stationary distribution Π_1 is

$$
\begin{array}{[ccccccccccc]}
\frac{a}{C}, & \frac{b}{C}, & \frac{c}{C}, & \frac{d}{C}, & \frac{e}{C}, & \frac{f}{C}, & \frac{g}{C}, & \frac{h}{C}, & \frac{i}{C}, & \frac{j}{C}, & \frac{k}{C}, \\
\frac{l}{C}, & \frac{m}{C}, & \frac{n}{C}, & \frac{o}{C}, & \frac{p}{C}, & \frac{q}{C}, & \frac{rs}{(r+s)C}, & \frac{st}{(r+s)C}, & \frac{ru}{(r+s)C}, & \frac{v}{C}, & \frac{1}{C}
\end{array}
$$

where

$$
C = 1 + a + b + c + d + e + f + g + h + i + j + k + l + m + n + o + p + q + \frac{rs}{r+s} + \frac{st}{r+s} + \frac{ru}{r+s} + v
$$

We can also note that, because of the chosen instantiatiation of the cost functions above, some of the obtained rates coincide, thus possibly simplifying our computations.

$$
\begin{array}{ll}
c_{10} = c_{12} = c_{14} = c_{16} = c_{331} & \\
c_{19} = c_{si0} & i \in [0, 3] \\
c_{si1} & i \in \{1, 3\} \\
c_{si1} & i \in \{0, 2\}
\end{array}
$$

Similarly, we can derive the generator matrix $\widehat{Q}_1'$ and the corresponding stationary distribution $\widehat{\Pi}_1$ for the transition system corresponding to $\widehat{N}^c$.

$$
\begin{array}{[ccccccccccc]}
\frac{a}{\hat{C}}, & \frac{b}{\hat{C}}, & \frac{c}{\hat{C}}, & \frac{d}{\hat{C}}, & \frac{e}{\hat{C}}, & \frac{f}{\hat{C}}, & \frac{g}{\hat{C}}, & \frac{h}{\hat{C}}, & \frac{i}{\hat{C}}, & \frac{j}{\hat{C}}, & \frac{k}{\hat{C}}, \\
\frac{l}{\hat{C}}, & \frac{m}{\hat{C}}, & \frac{n}{\hat{C}}, & \frac{o}{\hat{C}}, & \frac{p}{\hat{C}}, & \frac{q}{\hat{C}}, & \frac{rs}{(r+s)\hat{C}}, & \frac{st}{(r+s)\hat{C}}, & \frac{ru}{(r+s)\hat{C}}, & \frac{v}{\hat{C}}, & \frac{1}{\hat{C}}
\end{array}
$$

$$
\hat{C} = 1 + a + b + c + d + e + f + g + h + i + j + k + l + m + n + o' + p' + q' + \frac{rs}{r+s} + \frac{st}{r+s} + \frac{ru}{r+s} + v
$$

Note that the only differences are in the rates o', p', q' corresponding to the rates of the transitions that differ from the ones in N^c:

$$o' = \hat{c}_{s31}$$
$$p' = \hat{c}_{17}$$
$$q' = \hat{c}_{18}$$

Also note that it is easy to study the impact of variations on rate parameters on the performance analysis by resorting to a computer algebra package like Mathematica that is able to solve symbolic equations.

4.5. Reward structures

To define performance measures for a system N, we define the corresponding *reward structure*, following [36, 31]. Usually, a reward structure is a function that associates a reward with any state passed through a computation of N, according to the measures of interest. For instance, for computing the amount of usage of a resource, a reasonable choice is to associate a nonzero reward (e.g. 1) with any state in which the resource can be used and zero to the others.

As in [15], since we are in a process algebraic setting, we compute rewards of states, starting from transition rates.

Definition 8. Given a function ρ associating a transition reward with each transition θ in a transition system, the reward of a state N is

$$\rho_N = \sum_{N \xrightarrow{\theta} N'} \rho(\theta)$$

The reward structure of a system N is a vector of rewards, whose elements correspond to the states reachable from N, i.e. to its derivatives $d(N)$. By looking at the stationary distribution and varying the reward structure, we can compute different performance measures. The *total reward* is obtained by combining the values of the stationary distribution Π with rewards.

Definition 9. Given a system N, let $\Pi = (X_0, \ldots, X_{n-1})$ be its stationary distribution. The *total reward* of N is computed as $R(N) = \sum_{N_i \in d(N)} \rho_{N_i} \cdot X_i$, where $d(N)$ is the set of derivatives of N.

4.6. Smart storehouse example (cont'd)

We now apply the method above, to analyse the performance of both systems of our running example, to compare their relative efficiency. We can consider different performance measures, depending on particular aspects of system behaviour, by suitably associating rewards with the set of activities of interest.

A possible measure can describe the number of actuation commands decided per time unit. Thus, we associate a nonzero reward to the transitions that represent the sending of an actuation command performed by node N_3 and $\hat{N}_3$. The corresponding transitions have labels θ_{330} and θ_{331}, $\hat{\theta}_{330}$ and $\hat{\theta}_{331}$. Now, we assign nonzero rewards with the nodes from which those transitions come.

$$\rho_{N_{18}} 1; \rho_{N_{19}} = 1; \rho_{N_i} = 0 \quad i \notin \{18, 19\}$$
$$\rho_{\hat{N}_{18}} 1; \rho_{\hat{N}_{19}} = 1; \rho_{\hat{N}_i} = 0 \quad i \notin \{18, 19\}$$

The total reward $R(N^c)$ of the system then amounts to

$$\frac{st}{(r+s)C} + \frac{ru}{(r+s)C}$$

while $R(\widehat{N}^c)$ to

$$\frac{st}{(r+s)\hat{C}} + \frac{ru}{(r+s)\hat{C}}$$

By comparing the two throughputs, since $C > \hat{C}$, it is straightforward to obtain that $R(N^c) < R(\widehat{N}^c)$, i.e. that, as expected, $\widehat{N}^c$ performs better than N^c. To use this measure, it is necessary to instantiate our parameters under various hypotheses, depending on several factors, such as the network load, the packet size, and so on. Furthermore, we need to consider the costs of cryptographic algorithms and how changing their parameters impact on the guaranteed security level (see e.g. [37]).

Our running example on concrete devices. To make our analysis concrete, we instantiate our example on real devices (note that the average of the density function is the inverse of the rate). We assume to have a mote-class hardware platform, say MicaZ [38] and that our communication primitives are implemented by a typical radio sub-system for sensor networks, for instance based on IEEE 802.15.4 [38], with a bandwidth of 256 *kbps*. Hence, we set the time of sending/receiving a message to $\texttt{se} \backsimeq \texttt{re} \backsimeq 2\ ms$. The cost of cryptographic primitives depends on the adopted solution: software or hardware. In our case, we take the cost from [39]. In the first solution, the time for encrypting is $\texttt{en} \backsimeq 1.5\ ms$ and for decrypting is $\texttt{de} \backsimeq 1.5\ ms$. In the second solution, we assume to have a chip radio like CC2420. Although this chip does not provide decryption via hardware [39], we assume that the decryption cost is equivalent to that of the encryption, which is $\texttt{en} \backsimeq \texttt{de} \backsimeq 30\ \mu s$. We also consider a third case in which the encryption is performed at the hardware level and the decryption at the software level. For the sake of simplicity, suppose to further simplify the costs of the transitions of the system of N^c, by only considering the costs of sending, receiving, encrypting and decrypting, and by neglecting constants. The new costs are therefore as follows.

$$\begin{array}{ll} a = \texttt{se}; & l = \texttt{re}; \\ b = \texttt{re}; & m = \texttt{se}; \\ c = \texttt{se}; & n = \texttt{re}; \\ d = \texttt{re}; & o = (\texttt{se} + \texttt{en}); \\ e = \texttt{se}; & p = (\texttt{re} + \texttt{de}); \\ f = \texttt{re}; & q = \texttt{se}; \\ g = (\texttt{se} + \texttt{en}); & r = \texttt{re}; \\ h = (\texttt{re} + \texttt{de}); & s = \texttt{re}; \\ i = \texttt{se}; & t = \texttt{se}; \\ j = \texttt{re}; & u = \texttt{se}; \\ k = \texttt{se}; & v = \texttt{re}; \\ & w = \texttt{se} \end{array}$$

When $\mathtt{se} = \mathtt{re} = 2\ ms$ and $\mathtt{en} = \mathtt{de} = 1.5\ ms$, we have the following stationary distribution Π_1

$$[\tfrac{1}{23}, \tfrac{1}{23}, \tfrac{1}{23}, \tfrac{1}{23}, \tfrac{1}{23}, \tfrac{1}{23}, \tfrac{7}{92}, \tfrac{7}{92}, \tfrac{1}{23}, \tfrac{1}{23}, \tfrac{1}{23}, \tfrac{1}{23}, \tfrac{1}{23}, \tfrac{1}{23}, \tfrac{7}{92}, \tfrac{7}{92}, \tfrac{1}{23}, \tfrac{1}{46}, \tfrac{1}{46}, \tfrac{1}{46}, \tfrac{1}{23}, \tfrac{1}{46}]$$

whereas for $\widehat{N}^c$, we would have (with $o' = \mathtt{se}$ and $p' = \mathtt{re}$) the following stationary distribution $\widehat{\Pi}_1$:

$$[\tfrac{2}{43}, \tfrac{2}{43}, \tfrac{2}{43}, \tfrac{2}{43}, \tfrac{2}{43}, \tfrac{2}{43}, \tfrac{7}{86}, \tfrac{7}{86}, \tfrac{2}{43}, \tfrac{2}{43}, \tfrac{2}{43}, \tfrac{2}{43}, \tfrac{2}{43}, \tfrac{2}{43}, \tfrac{2}{43}, \tfrac{2}{43}, \tfrac{1}{43}, \tfrac{1}{43}, \tfrac{1}{43}, \tfrac{2}{43}, \tfrac{1}{43}]$$

When $\mathtt{se} = \mathtt{re} = 2\ ms$ and $\mathtt{en} = \mathtt{de} = 0.03\ ms$, we have the following stationary distributions Π_1 and $\widehat{\Pi}_1$:

$$[\tfrac{50}{1003}, \tfrac{50}{1003}, \tfrac{50}{1003}, \tfrac{50}{1003}, \tfrac{50}{1003}, \tfrac{50}{1003}, \tfrac{203}{4012}, \tfrac{203}{4012}, \tfrac{50}{1003}, \tfrac{50}{1003}, \tfrac{50}{1003},$$
$$\tfrac{50}{1003}, \tfrac{50}{1003}, \tfrac{50}{1003}, \tfrac{203}{4012}, \tfrac{203}{4012}, \tfrac{50}{1003}, \tfrac{25}{1003}, \tfrac{25}{1003}, \tfrac{25}{1003}, \tfrac{50}{1003}, \tfrac{25}{1003}]$$

$$[\tfrac{100}{2003}, \tfrac{100}{2003}, \tfrac{100}{2003}, \tfrac{100}{2003}, \tfrac{100}{2003}, \tfrac{100}{2003}, \tfrac{203}{4006}, \tfrac{203}{4006}, \tfrac{100}{2003}, \tfrac{100}{2003}, \tfrac{100}{2003},$$
$$\tfrac{100}{2003}, \tfrac{100}{2003}, \tfrac{100}{2003}, \tfrac{100}{2003}, \tfrac{100}{2003}, \tfrac{100}{2003}, \tfrac{50}{2003}, \tfrac{50}{2003}, \tfrac{50}{2003}, \tfrac{100}{2003}, \tfrac{50}{2003}]$$

If $\mathtt{se} = \mathtt{re} = 2\ ms$ and $\mathtt{en} = 0.03\ ms$ and $\mathtt{de} = 1.5\ ms$, we have the following stationary distributions Π_1 and $\widehat{\Pi}_1$:

$$[\tfrac{100}{2153}, \tfrac{100}{2153}, \tfrac{100}{2153}, \tfrac{100}{2153}, \tfrac{100}{2153}, \tfrac{100}{2153}, \tfrac{203}{4306}, \tfrac{175}{2153}, \tfrac{100}{2153}, \tfrac{100}{2153}, \tfrac{100}{2153},$$
$$\tfrac{100}{2153}, \tfrac{100}{2153}, \tfrac{100}{2153}, \tfrac{203}{4306}, \tfrac{175}{2513}, \tfrac{100}{2153}, \tfrac{50}{2153}, \tfrac{50}{2153}, \tfrac{50}{2153}, \tfrac{100}{2153}, \tfrac{50}{2153}]$$

$$[\tfrac{200}{4153}, \tfrac{200}{4153}, \tfrac{200}{4153}, \tfrac{200}{4153}, \tfrac{200}{4153}, \tfrac{200}{4153}, \tfrac{203}{4153}, \tfrac{350}{4153}, \tfrac{200}{4153}, \tfrac{200}{4153}, \tfrac{200}{4153},$$
$$\tfrac{200}{4153}, \tfrac{200}{4153}, \tfrac{200}{4153}, \tfrac{200}{4153}, \tfrac{200}{4153}, \tfrac{200}{4153}, \tfrac{100}{4153}, \tfrac{100}{4153}, \tfrac{100}{4153}, \tfrac{200}{4153}, \tfrac{100}{4153}]$$

From these distributions, for instance in the first case, we can obtain that

$$R(N^c) = \frac{2}{46} < \frac{2}{43} = R(\widehat{N}^c).$$

4.7. Performance Bisimulation

In this subsection, we define a bisimulation-based equivalence that represents a further tool to compare program performance. We adapt to IoT-LySa the notion of late bisimulation and of *performance* bisimulation, introduced in [33] for π-calculus to cope with exponential transition distributions. As in [33], these notions are parametric as they depend on a given function that applied to enhanced labels, extracts the part of information of interest there encoded, which in our setting can be information on the performed actions independently from the labels of the involved nodes.

Definition 10. Given a function f and a binary relation $\mathcal{S}$ on systems of nodes N, it is an *f-simulation* if $N_1 \mathcal{S} N_2$ implies that

- if $N_1 \xrightarrow{\ell_1 \langle \theta_B \rangle} N_1'$, then for some N_2', $N_2 \xrightarrow{\ell_2 \langle \theta_B \rangle} N_2'$, $f(\ell_1 \langle \theta_B \rangle) = f(\ell_2 \langle \theta_B \rangle)$, and $N_1' \mathcal{S} N_2'$;
- if $N_1 \xrightarrow{\ell_O \triangleright \ell_I \langle \theta_C \rangle} N_1'$, then for some N_2', $N_2 \xrightarrow{\ell_O' \triangleright \ell_I' \langle \theta_C \rangle} N_2'$, $f(\ell_O \triangleright \ell_I \langle \theta_C \rangle) = f(\ell_O' \triangleright \ell_I' \langle \theta_C \rangle)$, and $N_1' \mathcal{S} N_2'$.

The relation $\mathcal{S}$ is an *f-bisimulation* if both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are f-simulations. We write $N_1 \approx_f N_2$, i.e. N_1 is bisimilar to N_2, if there exists an f-bisimulation $\mathcal{S}$ such that $N_1 \mathcal{S} N_1'$.

If we put $f = \Lambda$ we obtain a bisimulation, reminiscent of the classical late bisimulation of the π-calculus, that allows us to investigate the qualitative properties of systems of nodes, since we are only focussing on the kind of the performed actions.

To deal with the stochastic information implicitly carried by transition labels, we extend the conditions given in Definition 10 on transitions between states to conditions on transitions between equivalence classes. Indeed in the qualitative setting, bisimulation equivalence of two states requires that any transition of one state has at least one matching transition of the other state, while in the quantitative one bisimulation takes the "quantity" (e.g. rates) of transitions into account. To this aim we resort to the function γ_0 in the following proposition that gives an alternative definition of $\approx_f$, as done in [33, 34].

Proposition 1. *Given a function f, a binary relation $\mathcal{S}$ on systems of nodes N is an f-*simulation *if $N_1 \mathcal{S} N_2$ implies that for any equivalence class C originated from $\mathcal{S}$*

$$\begin{aligned} \forall \theta_1.\ \gamma_0(N_1, \theta_1, C) &= \gamma_0(N_2, \theta_2, C), \\ f(\theta_1) &= f(\theta_2), \end{aligned}$$

where

$$\gamma_0(N, \theta, C) = \begin{cases} 1 & \text{if } \exists N' \in C.N \xrightarrow{\theta} N' \\ 0 & \text{otherwise} \end{cases}$$

Proof.

- (Case $\theta = \ell \langle \theta_B \rangle$). According to Definition 10, we have to prove that $N_1 \mathcal{S} N_1'$ implies that if $N_1 \xrightarrow{\theta_1} N_1'$, then for some N_2', $N_2 \xrightarrow{\theta_2} N_2'$, $f(\theta_1) = f(\theta_2)$ and $N_1' \mathcal{S} N_2'$. By hypothesis, we know that $\gamma_0(N_1, \theta_1, C) = \gamma_0(N_2, \theta_2, C)$. The case when $\gamma_0(N_1, \theta_1, C) = 0$ is trivial. When $\gamma_0(N_1, \theta_1, C) = 1$ we know that there exist $N_1', N_2' \in C$ such that $N_1 \xrightarrow{\theta_1} N_1'$ and $N_2 \xrightarrow{\theta_2} N_2'$. Furthermore, it holds that $N_1' \mathcal{S} N_2'$ because C is the equivalent class originated by $\mathcal{S}$. Since we know $f(\theta_1) = f(\theta_2)$ from the hypotheses, we can conclude that $\mathcal{S}$ is an f-simulation.

  Repeating the same pattern of proof starting with $N_2 \xrightarrow{\theta_2} N_2'$, we obtain that also $\mathcal{S}^{-1}$ is an f-simulation, and therefore $\mathcal{S}$ is an f-bisimulation.

- (Case $\theta = \ell_O \triangleright \ell_I \ \langle \theta_C \rangle$). The proof is similar and therefore omitted. □

We now instantiate the f-bisimulation by modifying the definition of the function γ_0, in order to consider the duration of transitions, according to exponential distributions. To this aim, we use the function $\hat{\gamma}$, where the cost function $\$_t$ is applied to enhanced labels for extracting the corresponding transition rates. Given an equivalence class C, $\hat{\gamma}(N, \theta, C)$ is given by the sum of the conditional transition rates $q(N_i, N_j, \theta)$ (see Definition 6) between N and any $N_i \in C$, via a given action labelled θ_i, with $\Lambda(\theta_i) = \Lambda(\theta)$.

Note that, in the exponential case, equivalence classes are just singletons N. Furthermore, we can use Λ as function f.

Definition 11. The exponential function $\hat{\gamma}$ is defined as

$$\hat{\gamma}(N, \theta, C) = \sum_{N_i \in C, \Lambda(\theta_i) = \Lambda(\theta)} q(N_i, \theta, N_j) = \sum_{N_i \in C, \Lambda(\theta_i) = \Lambda(\theta)} r_i,$$

where $r_i = \$_t(\theta_i)$ are the exponential distributions associated with a $N \xrightarrow{\theta_i} N_i$ transition, where $\Lambda(\theta_i) = \Lambda(\theta)$.

Note that $\hat{\gamma}$ is the *total conditional rate* from N to C. Two nodes are performance bisimilar if there is an equivalence relation such that, for any action labelled θ, the total conditional transition rates originating from those components to any equivalence class, via activities of the same kind, coincide.

Definition 12. Given a function $\$_t$, a binary relation $\mathcal{S}$ on systems of nodes N, is a *performance simulation* if $N_1 \mathcal{S} N_2$ implies that for any equivalence class C originating from $\mathcal{S}$

$$\forall \theta_1.\ \hat{\gamma}(N_1, \theta_1, C) = \hat{\gamma}(N_2, \theta_2, C), \quad \Lambda(\theta_1) = \Lambda(\theta_2).$$

The relation $\mathcal{S}$ is a *performance bisimulation* if both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are performance simulations. We write $N_1 \approx_p N_2$, i.e. N_1 is bisimilar to N_2, if there exists a performance bisimulation $\mathcal{S}$ such that $N_1 \mathcal{S} N_1'$.

Example. We apply our notion of performance bisimulation to our case study. In the version of Figure 6, we have a node N_{s_i} for each sensor S_{s_i} (we omit here the other nodes and the communications with other nodes).

$$\begin{array}{lcll} N^s & = & N_{s_0} \mid N_1 \mid N_2 \mid N_3 & \\ N_{s_i} & = & \ell_{s_i} : [\Sigma_{s_i} \parallel P_{s_i} \parallel S_{s_i}] & \text{where } S_{s_i} = s_i := v_{s_i}.\tau.S_{s_i} \qquad i \in [0,3] \\ P_{s_i} & = & (; x_{start_i})^{si0}.\langle\!\langle \ell_{s_i}, s_i \rangle\!\rangle \triangleright \{\ell_1\}^{si1}.P_{s_i} & i \in [0,2] \\ P_{s_i} & = & (; x_{start_i})^{si0}.\langle\!\langle \{\ell_{s_i}, s_i\}_{k_{s_i}} \rangle\!\rangle \triangleright \{\ell_1\}^{si1}.P_{s_i} & i \in [1,3] \end{array}$$

$$\begin{array}{lcl} N_1 & = & \ell_1 : [\Sigma_1 \parallel P_1 \parallel A_0] \\ P_1 & = & \langle\!\langle start_0 \rangle\!\rangle \triangleright \{\ell_{s_0}\}^{10}.(\ell_{s_0}; z_0)^{11}.\langle\!\langle start_1 \rangle\!\rangle \triangleright \{\ell_{s_1}\}^{12}.(\{\ell_{s_1}; z_1\}_{k_{s_1}})^{13}. \\ & & \langle\!\langle start_2 \rangle\!\rangle \triangleright \{\ell_{s_2}\}^{14}.(\ell_{s_2}; z_2)^{15}.\langle\!\langle start_3 \rangle\!\rangle \triangleright \{\ell_{s_3}\}^{16}.(\{\ell_{s_3}; z_3\}_{k_{s_3}})^{17}... \end{array}$$

Suppose that, for optimisation reasons, we are interested in using fewer nodes, but keeping a similar behaviour. Thus, we define a new specification and we exploit our performance bisimulation to check that it has the same behaviour of the previous one.

We illustrate the case in which there are two nodes $N_{s_{01}}$ and $N_{s_{23}}$, where the first one collects the data coming from the sensors S_{s_0} and S_{s_1}, while the second node collects the data coming from the sensors S_{s_2} and S_{s_3}.

$$
\begin{array}{lcl}
N^s & = & N_{s_{01}} \mid N_{s_{23}} \mid N_1 \mid ... \\
N_{s_{i,i+1}} & = & \ell_{s_i} : [\Sigma_{s_{i,i+1}} \parallel P_{s_{i,i+1}} \parallel S_{s_i} \parallel S_{s_{i+1}}] \text{ where} \\
 & & S_{s_k} = s_k := v_{s_k}.\tau.S_{s_k} \;\; k \in [i, i+1], i \in [0,2] \\
P_{s_{i,i+1}} & = & (; x_{start_i})^{si0}.\langle\!\langle \ell_{s_i}, s_i \rangle\!\rangle \triangleright \{\ell_1\}^{si1}. \\
 & & (; x_{start_{i+1}})^{s(i+1)0}.\langle\!\langle \{\ell_{s_i}, s_i\}_{k_{s_i}} \rangle\!\rangle \triangleright \{\ell_1\}^{s(i+1)1}.P_{s_{i,i+1}} \quad i \in \{0,2\} \\
N_1 & = & \ell_1 : [\Sigma_1 \parallel P_1 \parallel A_0] \\
P_1 & = & \langle\!\langle start_0 \rangle\!\rangle \triangleright \{\ell_{s_{01}}\}^{10}.(\ell_{s_0}; z_0)^{11}.\langle\!\langle start_1 \rangle\!\rangle \triangleright \{\ell_{s_{01}}\}^{12}.(\{\ell_{s_1}; z_1\}_{k_{s_1}})^{13}. \\
 & & \langle\!\langle start_2 \rangle\!\rangle \triangleright \{\ell_{s_{23}}\}^{14}.(\ell_{s_2}; z_2)^{15}.\langle\!\langle start_3 \rangle\!\rangle \triangleright \{\ell_{s_{23}}\}^{16}.(\{\ell_{s_3}; z_3\}_{k_{s_3}})^{17} ...
\end{array}
$$

It can be easily verified that the two systems $N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1$ and $N_{s_{01}} \mid N_{s_{23}} \mid N_1$ are performance bisimilar, since they match each other's moves.

5. Measuring the energy

In this section we show how to extend our analysis to measure also the energy depleted by a network of nodes. Here, we assume that the energy consumed by a node during its execution is proportional to computational and communication activities and to the time spent to perform them (that is, in turn, inversely proportional to the corresponding rates).

We then introduce a cost function $\$_e \colon \Theta \to \mathbb{R}^+$, by extending the one $\$_t$ defined in Section 4, defined as follows

$$
\$_e(\theta) = \begin{cases} \frac{e_c}{\$_t(\theta)} & \text{if } \theta = \ell\langle\theta_b\rangle \text{ for some } \ell \text{ and } \theta_b \\ \frac{e_r}{\$_t(\theta)} & \text{otherwise} \end{cases}
$$

where the factors e_c and e_r are the coefficients for computation and communication activities, respectively. Note that when the transition is a communication of computed values, we assume that the prevalent energy cost is the one of the transmission.

We differentiate between the computation and communication costs because the energy required for processing is different from that for transmission/reception. Furthermore, concerning the latter cost, we assume that this is proportional to the time spent in send/receive operations. Note that, in general, the energy cost on the communication sub-system is mostly due to the length of its period of activity (regardless it is in receive, send or listening state). However, if the MAC protocol used in the communication sub-system is based on low power listening mechanisms (for example B-MAC [40] or X-MAC [41]), then the costs are actually bound to the number of send and receive operations, as

defined in our model. We plan to extend this energy cost model to other MAC protocols not based on low power listening in our future work.

Note that the time, as measured in Section 4, can be considered a global resource that each node can consume. Thus, it is natural to consider its consumption by taking into account the whole behaviour of nodes and processes. The energy is instead local, and each node consumes independently of the other nodes. On the one hand, each node has its own energy supply as a battery, which is finite; on the other hand, the lifetime of the network can be affected by a single node that stops working due to the lack of energy. In this last case we are interested in looking, steps by steps, whether the energy budget is sufficient to go on. For this reason, we focus on the energy efficiency of each node and not on the one of the whole system. To do that, we introduce a notion of energy budget representing how much energy a node owns and may spend and, we define a new semantics, call it *energy-sensitive*, on top of the enhanced one of Section 3.

Here we consider two ways to deal with this energy budget. In the first case, a budget represents the maximal quantity of energy that a node may use during its execution, e.g. its initial battery charge. A node can complete a transition if it has enough budget and each action decrements this quantity. In the second case, a budget calculates how much charge a node requires in order to complete its computation and communication tasks. Thus, each action that a node performs increments the required budget.

For the new semantics we introduce some notations. Let $\Gamma\colon \mathcal{L} \to \mathbb{R}^+$ be the *energy function* from node labels to positive numbers, mapping each node to its energy budget. To uniformly represent the two ways of using the budget we use the symbol $\otimes$ that stands for the classical operation $+$ or $-$ on positive numbers. We denote with $\Gamma[\ell \otimes m]$ the function update defined as follows

$$\Gamma[\ell \otimes m](\ell') = \begin{cases} \Gamma(\ell) \otimes m & \text{for } \ell = \ell' \\ \Gamma[\ell \otimes m](\ell') = \Gamma(\ell') & \text{otherwise} \end{cases}$$

Furthermore, given an enhanced label θ, we use the shorthand $\theta.\ell$ to denote the label of the node carrying the transition out, i.e. $\theta.\ell = \ell$, when $\theta = \ell\langle\theta_b\rangle$, or $\theta.\ell = \ell_I$ when $\ell_O \triangleright \ell_I\ \langle\langle\!\langle E_1, \cdots, E_m \rangle\!\rangle, (E_1, \cdots, E_j; x_{j+1}, \cdots, x_m)\rangle$.

Given a system of nodes N the energy-sensitive semantics is characterised by transitions of the form $\Gamma \vdash N \xrightarrow{\theta} \Gamma' \vdash N'$ meaning that the system of nodes with an energy budget Γ performs a step with enhanced label θ and becomes N' with the energy budget updated to Γ', provided that the condition (see below the predicate $\diamond$) on budgets is respected. The whole semantics can be totally specified by the following rule:

$$\frac{N \xrightarrow{\theta} N' \qquad \diamond\,(\Gamma(\theta.\ell), \$_e(\theta))}{\Gamma \vdash N \xrightarrow{\theta} \Gamma[\theta.\ell \otimes \$_e(\theta)] \vdash N'}$$

where $\diamond$ is a predicate that says if the transition is energy affordable for the node with label ℓ with the current energy budget $\Gamma(\theta.\ell)$. For example, if we consider the budget as the upper bound a node may spend, a naive instantiation consists

in taking $\diamond$ equal to $\geq$. Instead, we can define the predicate as the one always true when we consider the budget as a counter of the required energy.

Note that the model where the budget is consumed can be used to capture the notion of lifetime of a network. This is defined in literature as the time at which the first network node depletes its battery (i.e. its energy battery becomes 0); alternatively as the time for a number or a percentage of nodes to use up the energy supplies. More precise characterisations can be found in [42], where lifetime is considered an application-specific notion, and therefore it should take into account not only the number of nodes, but also the importance of the single nodes for the application.

In our framework, we can compute the lifetime value, by analysing the transition graph of our system of nodes. It is sufficient to look for the shortest path reaching a vertex where one of the nodes has just depleted its battery. Similarly, we can look for a vertex where a given number (say k) of nodes deplete their batteries, or at least one of a given number (say k) of nodes depletes its battery.

5.1. Smart storehouse example (cont'd)

Back to our storehouse system, we apply the second kind of energy-sensitive semantics to measure the energy required by each node in order to complete a run of its duty cycle. In particular, for each node of the system N^c, the vector Γ calculates how much charge the node requires completing its task. Thus, each transition increments the required consumption of the node. By using the computed Γ we can decide the energy supply to assign to each node in order for the system to work for at least a certain number of iterations.

To this aim, we set the initial energy consumption Γ_0 to

Γ_0	ℓ_{s_0}	ℓ_{s_1}	ℓ_{s_2}	ℓ_{s_3}	ℓ_1	ℓ_3
	0	0	0	0	0	0

while the subsequent vectors, which in this case express the energy budget required to reach the respective state, are derived considering the following parametric runs.

$$\Gamma_0 \vdash N^c \xrightarrow{c_{10}} a \xrightarrow{c_{S00}} b \xrightarrow{c_{S01}} c \xrightarrow{c_{11}} d \xrightarrow{c_{12}} e \xrightarrow{c_{S10}} f \xrightarrow{c_{S11}} g \xrightarrow{c_{13}} h \xrightarrow{c_{14}} i \xrightarrow{c_{S20}} j \xrightarrow{c_{S21}} k \xrightarrow{c_{15}} l \xrightarrow{c_{16}} m \xrightarrow{c_{S30}} n \xrightarrow{c_{S31}} o \xrightarrow{c_{17}} p \xrightarrow{c_{18}} q$$

$$\Gamma' \vdash N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_3 \xrightarrow{c_{30i}}_{r/s} \begin{cases} \Gamma_0'' \vdash N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_{30}^{I} \text{ if } i = 0 \\ \Gamma_1'' \vdash N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_{31}^{I} \text{ if } i = 1 \end{cases}$$

$$\Gamma_i''' \vdash \xrightarrow{c_{33i}}_{t/u} N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1^{IX} \mid N_3$$

$$\xrightarrow{c_{19}} v \xrightarrow{c_{110}} w$$

$$\Gamma_i'''' \vdash N_{s_0} \mid N_{s_1} \mid N_{s_2} \mid N_{s_3} \mid N_1 \mid N_3 = N^c$$

where

Γ'	ℓ_{s_0}	ℓ_{s_1}	ℓ_{s_2}	ℓ_{s_3}	ℓ_1	ℓ_3
Γ''	$\frac{(c_{s00}+c_{s01})}{e_r}$ ℓ_{s_0}	$\frac{(c_{s10}+c_{s11})}{e_r}$ ℓ_{s_1}	$\frac{(c_{s20}+c_{s21})}{e_r}$ ℓ_{s_2}	$\frac{(c_{s30}+c_{s31})}{e_r}$ ℓ_{s_3}	$\frac{(c_{10}+\ldots+c_{18})}{e_r}$ ℓ_1	0 ℓ_3
Γ_i'''	$\frac{(c_{s00}+c_{s01})}{e_r}$ ℓ_{s_0}	$\frac{(c_{s10}+c_{s11})}{e_r}$ ℓ_{s_1}	$\frac{(c_{s20}+c_{s21})}{e_r}$ ℓ_{s_2}	$\frac{(c_{s30}+c_{s31})}{e_r}$ ℓ_{s_3}	$\frac{(c_{10}+\ldots+c_{18})}{e_r}$ ℓ_1	$\frac{c_{3i0}}{e_r}$ ℓ_3
Γ_i''''	$\frac{(c_{s00}+c_{s01})}{e_r}$ ℓ_{s_0}	$\frac{(c_{s10}+c_{s11})}{e_r}$ ℓ_{s_1}	$\frac{(c_{s20}+c_{s21})}{e_r}$ ℓ_{s_2}	$\frac{(c_{s30}+c_{s31})}{e_r}$ ℓ_{s_3}	$\frac{(c_{10}+\ldots+c_{18})}{e_r}$ ℓ_1	$\frac{c_{3i0}+c_{33i}}{e_r}$ ℓ_3
	$\frac{(c_{s00}+c_{s01})}{e_r}$	$\frac{(c_{s10}+c_{s11})}{e_r}$	$\frac{(c_{s20}+c_{s21})}{e_r}$	$\frac{(c_{s30}+c_{s31})}{e_r}$	$\frac{(c_{10}+\ldots+c_{110})}{e_r}$	$\frac{c_{3i0}+c_{33i}}{e_r}$

Therefore, the last energy consumption Γ_i'''' is also the energy budget required to complete an entire cycle.

Γ_0	ℓ_{s_0}	ℓ_{s_1}	ℓ_{s_2}	ℓ_{s_3}	ℓ_1	ℓ_3
	Γ_{s_0}	Γ_{s_1}	Γ_{s_2}	Γ_{s_3}	Γ_1	Γ_3

6. Related Work

Our approach follows the well-established line of research about performance evaluation through process calculi and probabilistic model checking (see [43, 44] for a survey). To the best of our knowledge, the application of formal methods to IoT systems or to wireless sensor networks is still in its youth, and only a limited number of papers in the literature addressed the problem from a process algebras perspective, e.g. [45, 46, 47, 48]. In [49] the problem of modelling and estimating the communication cost in an IoT scenario is tackled through Stochastic Petri Nets. Their approach is similar to ours: they derive a CTMC from a Petri Net describing the system and proceed with the performance evaluation by using standard tools. Differently from us, they focus not on the cost of security but only on the one of communication (they do not use cryptographic primitives). In [50] a performance comparison between the security protocols IPSec and DTLS is presented, in particular by considering their impact on the resources of IoT devices with limited computational capabilities. They modified protocols implementations to make them properly run on the devices. An extensive experimental evaluation study on these protocols shows that both their implementations ensure a good level of end-to-end security. In [51] the authors propose a methodology similar to ours that combines a security analysis for non-interference with an evaluation of the performance. As in our case, they use the same formal system to reason on both aspects.

Another line of research related to our is that of [52, 53, 54], where the authors investigate security issues arising in IoT and cyber-physical systems (CPSs) in a process algebraic setting. In particular, in [52] the hybrid process calculus `CCPSA` is presented for modelling cyber-physical system and cyber-physical attacks. The attacks taken into account concern physical layer, i.e. an attacker can tamper with both the sensors and actuators. In [53] is presented a bisimulation metric to compare the behaviour of an IoT system with the behaviour of the same system under attack. This metric aims at evaluating not only the vulnerability of a IoT system to a certain attack, but also the impact of a successful attack in terms of the deviation introduced in the behaviour of the target system. Finally, in [54] `pCCPS`, a hybrid probabilistic process calculus with a discrete notion of time, is proposed for specifying and reasoning on CPSs. Cyber-physical systems are represented by a physical components describing the physical processes and a cyber component that govern sensors and actuators, and that is responsible for the communication with other cyber components. Furthermore, `pCCPS` is equipped with a bisimulation metric that formalise a notion of behavioural distance between two systems.

7. Conclusions

In the IoT scenario security is critical but it is hard to address in an affordable way due to the limited computational capabilities of smart objects. We have presented a formal framework that supports designers in specifying an IoT system and in estimating the cost of security mechanisms. Using a process algebraic language for modelling both the behaviour and the performance of systems allows us to incorporate performance analysis into the design and to obtain a qualitative and quantitative model, starting from the same system specification. A key feature of our approach is that quantitative aspects are symbolically represented by parameters. Actual values are obtained as soon as the designer provides some additional information about the hardware and the network architecture and the cryptographic algorithms relative to the system in hand. By abstractly reasoning about these parameters designers can compare different implementations of the same IoT system, and choose the one that ensures the best trade-off between security guarantees and their price. In practice, we considered the process algebra IoT-LySa [7] and we adapted the technique of [16] to determine the costs of using/not using cryptographic measures in communications and to reason about the cost-security trade-offs. In particular, we defined an enhanced semantics, where each system transition is associated with a rate in the style of [14, 15]. From the rates we derive a CTMC, through which we could perform cost evaluation, by using standard techniques and tools [18, 19].

As future work, we plan to improve our proposal by including asymmetric cryptography into the picture and to introduce mechanisms to consider context-aware security in the line of [55, 56]. Moreover, we plan to realise a tool for the design and development of large-scale sensor networks. Finally, we will extend the current model for energy efficiency to adapt to different settings, MAC-layers and multi-hop networks.

Acknowledgements We thank Francesco Romani for his help in the algebraical treatment of our case study data.

References

[1] T. Simon, Critical infrastructure and the Internet of Things.
URL https://www.cigionline.org/publications/critical-infrastructure-and-internet-things-0

[2] CISCO, IoE-Driven Smart Street Lighting Project Allows Oslo to Reduce Costs, Save Energy, Provide Better Service.
URL http://www.cisco.com/c/dam/m/en_us/ioe/public_sector/pdfs/jurisdictions/Oslo_Jurisdiction_Profile_051214REV.pdf

[3] U.S. Department of Homeland Security, Strategic principles for securing the Internet of Things.
URL https://www.dhs.gov/securingtheIoT

[4] P. Oltermann, German parents told to destroy doll that can spy on children.
URL https://www.theguardian.com/world/2017/feb/17/german-parents-told-to-destroy-my-friend-cayla-doll-spy-on-children

[5] K. Zetter, Everything We Know About Ukraine's Power Plant Hack.
URL https://www.wired.com/2016/01/everything-we-know-about-ukraines-power-plant-hack/

[6] A. Greenberg, Hackers Remotely Kill a Jeep on the Highway—With Me in It.
URL http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

[7] C. Bodei, P. Degano, G.-L. Ferrari, L. Galletta, Where do your IoT ingredients come from?, in: Procs. of Coordination 2016, Vol. 9686 of LNCS, Springer, 2016, pp. 35–50.

[8] C. Bodei, P. Degano, G.-L. Ferrari, L. Galletta, A step towards checking security in IoT, in: Procs. of ICE 2016, Vol. 223 of EPTCS, 2016, pp. 128–142.

[9] C. Bodei, P. Degano, G.-L. Ferrari, L. Galletta, Tracing where IoT data are collected and aggregated, Logical Methods in Computer Science 13 (3:5) (2017) 1–38.

[10] C. Bodei, L. Galletta, Tracking sensitive and untrustworthy data in IoT, in: Procs.of the First Italian Conference on Cybersecurity (ITASEC 2017), CEUR Vol-1816, 2017, pp. 38–52.

[11] C. Bodei, P. Degano, L. Galletta, E. Tuosto, Tool supported analysis of IoT, in: Procs. of ICE 2017, Vol. 261 of EPTCS, 2017, pp. 37–56.

[12] C. Bodei, P. Degano, G.-L. Ferrari, L. Galletta, Sustainable precision agriculture from a process algebraic perspective: a smart vineyard, Atti Soc. Tosc. Sci. Nat.e Mem., Suppl. 125 (2018) 39–44.

[13] P. Degano, C. Priami, Non interleaving semantics for mobile processes, Theoretical Computer Science 216.

[14] P. Degano, C. Priami, Enhanced operational semantics, ACM Computing Surveys 33 (2) (2001) 135 – 176.

[15] C. Nottegar, C. Priami, P. Degano, Performance evaluation of mobile processes via abstract machines, Transactions on Software Engineering 27 (10).

[16] C. Bodei, M. Buchholtz, M. Curti, P. Degano, F. Nielson, H. R. Nielson, C. Priami, On evaluating the performance of security protocols, in: Proc. of PaCT 2005, Vol. 3606 of LNCS, Springer, 2005, pp. 1 – 15.

[17] C. Bodei, M. Curti, P. Degano, C. Priami, A quantitative study of two attacks, Electr. Notes Theor. Comput. Sci. 121 (2005) 65–85.

[18] A. Reibnam, R. Smith, K. Trivedi, Markov and Markov reward model transient analysis: an overview of numerical approaches, European Journal of Operations Research 40 (2) (1989) 257–267.

[19] W. J. Stewart, Introduction to the numerical solutions of Markov chains, Princeton University Press, 1994.

[20] I. Dietrich, F. Dressler, On the lifetime of wireless sensor networks, ACM Trans. Sen. Netw. 5 (1) (2009) 5:1–5:39.

[21] C. Bodei, L. Galletta, The cost of securing IoT communications?, in: Procs.of Italian Conference on Theoretical Computer Science, CEUR Vol-1720, 2016, pp. 163–176.

[22] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, H. R. Nielson, Automatic validation of protocol narration, in: Computer Security Foundations Workshop (CSFW-16 2003), IEEE Computer Society, 2003, pp. 126–140.

[23] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, H. R. Nielson, Static validation of security protocols, Journal of Computer Security 13 (3) (2005) 347–390.

[24] H. Gao, C. Bodei, P. Degano, H. Nielson, A formal analysis for capturing replay attacks in cryptographic protocols., in: Proc. of ASIAN'07, LNCS 4846, Springer, 2007, pp. 150–165.

[25] H. Gao, C. Bodei, P. Degano, A formal analysis of complex type flaw attacks on security protocols, in: Proc. of AMAST'08, LNCS 5140, Springer, 2008, pp. 167–183.

[26] C. Bodei, L. Brodo, P. Degano, H. Gao, Detecting and preventing type flaws at static time, Journal of Computer Security 18 (2) (2010) 229–264.

[27] M. Herlihy, Wait-free synchronization, ACM Trans. Program. Lang. Syst. 13 (1) (1991) 124–149.

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

[29] B. Technology, Bluetooth core specification version 4.2.
URL https://www.bluetooth.com/specifications/bluetooth-core-specification

[30] C. Priami, Language-based performance prediction of distributed and mobile systems, Information and Computation 175 (2002) 119–145.

[31] J. Hillston, A Compositional Approach to Performance Modelling, Cambridge University Press, 1996.

[32] R. Nelson, Probability, Stochastic Processes and Queeing Theory, Springer, 1995.

[33] C. Priami, Language-based performance prediction for distributed and mobile systems, Inf. Comput. 175 (2) (2002) 119–145.

[34] J. Hillston, A compositional approach to performance modelling, Ph.D. thesis, University of Edinburgh, UK (1994).
URL http://hdl.handle.net/1842/15027

[35] Wolfram Mathematica, https://www.wolfram.com/mathematica/.

[36] R. Howard., Dynamic Probabilistic Systems: Semi-Markov and Decision Systems, Vol. Volume II, Wiley, 1971.

[37] J. Lee, K. Kapitanova, S. Son, The price of security in wireless sensor networks, Computer Networks 54 (17) (2010) 2967–2978.

[38] P. Baronti, P. Pillai, V. W. C. Chook, S. Chessa, A. Gotta, Y. Hu, Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards, Computer Communications 30 (7) (2007) 1655–1695.

[39] M. Healy, T. Newe, E. Lewis, Efficiently securing data on a wireless sensor network, Journal of Physics: Conference Series 76 (1) (2007) 012063.

[40] J. Polastre, J. Hill, D. Culler, Versatile low power media access for wireless sensor networks, in: Proc. of the 2nd International Conference on Embedded Networked Sensor Systems, SenSys 2004, 2004, pp. 95–107.

[41] M. Buettner, G. V. Yee, E. Anderson, R. Han, X-MAC: A short preamble mac protocol for duty-cycled wireless sensor networks, in: Proc. of the 4th International Conference on Embedded Networked Sensor Systems, SenSys '06, 2006, pp. 307–320.

[42] I. Dietrich, F. Dressler, On the lifetime of wireless sensor networks, TOSN 5 (1) (2009) 5:1–5:39.

[43] M. Kwiatkowska, G. Norman, D. Parker, Stochastic model checking, in: Procs. of Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM 2007), Vol. 4486 of LNCS, 2007, pp. 220–270.

[44] M. Kwiatkowska, D. Parker, Advances in probabilistic model checking, in: Procs. of Software Safety and Security - Tools for Analysis and Verification, Vol. 33, IOS Press, 2012, pp. 126–151.

[45] I. Lanese, D. Sangiorgi, An operational semantics for a calculus for wireless systems, Theor. Comput. Sci. 411 (19) (2010) 1928–1948.

[46] I. Lanese, L. Bedogni, M. D. Felice, Internet of things: a process calculus approach, in: Procs of Symp. on Applied Computing (SAC 2013), ACM, 2013, pp. 1339–1346.

[47] R. Lanotte, M. Merro, A semantic theory of the Internet of Things., in: Procs. of Coordination 2016, Vol. 9686 of LNCS, Springer, 2016, pp. 157–174.

[48] A. Singh, C. R. Ramakrishnan, S. Smolka, A process calculus for mobile ad hoc networks, Sci. Comput. Program. 75 (6) (2010) 440–469.

[49] L. Chen, L. Shi, W. Tan, Modeling and performance evaluation of Internet of Things based on Petri Nets and behavior expression, Research Journal of Applied Sciences, Engineering and Technology 4 (18) (2012) 3381–3385.

[50] A. D. Rubertis, L. Mainetti, V. Mighali, L. Patrono, I. Sergi, M. Stefanizzi, S. Pascali, Performance evaluation of end-to-end security protocols in an Internet of Things, in: Proc. of (SoftCOM) 2013, IEEE, 2013, pp. 1–6.

[51] A. Aldini, M. Bernardo, An integrated view of security analysis and performance evaluation: Trading qos with covert channel bandwidth, in: M. Heisel, P. Liggesmeyer, S. Wittmann (Eds.), Computer Safety, Reliability, and Security, 23rd International Conference, Vol. 3219 of LNCS, Springer, 2004, pp. 283–296.

[52] R. Lanotte, M. Merro, R. Muradore, L. Viganò, A formal approach to cyber-physical attacks, in: 30th IEEE Computer Security Foundations Symposium, CSF 2017, 2017, pp. 436–450.

[53] R. Lanotte, M. Merro, S. Tini, Towards a formal notion of impact metric for cyber-physical attacks, in: Proc. of the 14th International Conference on integrated Formal Methods (IFM 2018), Vol. 11023 of LNCS, 2018, pp. 296–315.

[54] R. Lanotte, M. Merro, S. Tini, A probabilistic calculus of cyber-physical systems, CoRR abs/1707.02279.

[55] C. Bodei, P. Degano, L. Galletta, F. Salvatori, Linguistic mechanisms for context-aware security, in: Proc. of 11th International Colloquium on Theoretical Aspects of Computing, LNCS 8687, Springer, 2014, pp. 61–79.

[56] C. Bodei, P. Degano, L. Galletta, F. Salvatori, Context-aware security: Linguistic mechanisms and static analysis, Journal of Computer Security 24 (4) (2016) 427–477.

$$\begin{bmatrix}
-\frac{1}{a} & \frac{1}{a} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -\frac{1}{b} & \frac{1}{b} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -\frac{1}{c} & \frac{1}{c} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -\frac{1}{d} & \frac{1}{d} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -\frac{1}{e} & \frac{1}{e} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -\frac{1}{f} & \frac{1}{f} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{g} & \frac{1}{g} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{h} & \frac{1}{h} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{i} & \frac{1}{i} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{j} & \frac{1}{j} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{k} & \frac{1}{k} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{l} & \frac{1}{l} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{m} & \frac{1}{m} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{n} & \frac{1}{n} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{o} & \frac{1}{o} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{p} & \frac{1}{p} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{q} & \frac{1}{q} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -(\frac{1}{r}+\frac{1}{s}) & \frac{1}{r} & \frac{1}{s} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{t} & 0 & \frac{1}{t} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{u} & \frac{1}{u} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{v} & \frac{1}{v} \\
\frac{1}{w} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{w}
\end{bmatrix} \quad (1)$$

Figure 7: A run of the SmartStore^c system