Securing IoT communications: at what cost?*

Chiara Bodei and Letterio Galletta

Dipartimento di Informatica, Università di Pisa
{chiara,galletta}@di.unipi.it

Abstract. IoT systems use wireless links for local communication, where locality depends on the transmission range and include many devices with low computational and battery power such as sensors. In IoT systems, security is a crucial requirement, but difficult to obtain, because standard cryptographic techniques have a cost that is usually unaffordable. We resort to an extended version of the process calculus LySA, called IoT-LySA, to model the patterns of communication of IoT devices. Moreover, we assign rates to each transition to infer quantitative measures on the specified systems. The derived performance evaluation can be exploited to establish the cost of the possible security countermeasures.

1 Introduction

Nowadays, "software is eating the world", i.e. software is pervading our everyday life and the objects we use such as webTV, coffeemakers, cars, smartphones, ebook readers, and Smart Cities, on a broader scale [3].

The main distinguishing feature of this scenario, called Internet of Things (IoT), is that in principle objects are always connected to the Internet and are equipped with different kinds of sensors, e.g. accelerometers, light sensors, microphone, and so on. These smart devices automatically collect information of various kinds and store them on the cloud or use them to directly operate on the surrounding environment through actuators. For instance, our smart alarm clock can drive our heating system to find a warm bathroom in the morning, while an alarm sensor in our place can directly trigger an emergency call to the closest police station. As a further example consider a storehouse stocking perishable food equipped with sensors to determine the internal temperature and other relevant attributes. The refrigeration system can automatically adapt the temperature according to the information collected by sensors.

The IoT paradigm introduces new pressing security challenges. Although security should be co-designed with the system and not just added

* Work partly supported by project PRA_2016_64 "Through the fog" funded by the University of Pisa.

on as an optional equipment, this is not easy in a setting where the usual trade-off between highly secure and system usability is more critical than ever. For instance, sensors are plagued by several security vulnerabilities: e.g. an attacker can easily intercept sensor communications and manipulate and falsify data. Nevertheless, security costs are high for devices with limited computational and communication capabilities and with limited battery power, and designers must be selective in choosing security mechanisms. Back to the refrigerator system above, to protect against falsification of data by an attacker, it is possible to resort to cryptography and to consistency data checks to prevent and detect anomalies. But is it affordable to secure all the communications? Can we obtain an affordable solution still having a good level of security by protecting only part of communications?

To relieve this kind of problems, IoT designers not only need tools to assess possible risks and to study countermeasures, but also methodologies to estimate their costs. The cost of security can be computed in terms of time overhead, energy consumption, bandwidth, and economic, and so on. All these factors must be carefully evaluated to achieve an acceptable balance between security, cost and usability of the system.

Usually, formal methods provide designers with tools to support the development of systems; also, they support them in proving properties, both qualitative and quantitative. The application of formal methodologies to securing IoT systems is still in its infancy. We would like to contribute by proposing IoT-LySA, an extension of the process calculus LySA [7] (a close relative of the π-[25] and Spi-calculus [1]) to model the patterns of communication of IoT devices, and to infer quantitative measures of the costs the specified systems bear for security mechanisms.

Here, we present preliminary steps, based on [6], towards the development of a formal methodology that supports designers in analysing the cost of security in IoT systems. Our long term goal is to provide a general framework with a mechanisable procedure (with a small amount of manual tuning), where quantitative aspects are symbolically represented by parameters. The instantiation of parameters is delayed until hardware architectures and cryptographic algorithms are fixed. By changing only these parameters designers could compare different implementations of IoT systems and could choose among different alternatives by selecting the better tradeoffs among security and costs. Technically, we extend LySA (Sect. 2) with suitable primitives for describing the activity of sensors and of sensor nodes, and for managing the coordination and communication capabilities of smart objects. The calculus is then given an

enhanced semantics, following [15], where each transition is associated to a rate in the style of [27]. It suffices to have information about the activities performed by the components of a system in isolation, and about some features of the network architecture. Starting from rates, it is possible to mechanically derive Markov chains. The actual cost of security can be carried out using standard techniques and tools [33,30,32]. Note that here quantitative measures can live together with the usual qualitative semantics, where instead these aspects are usually abstracted away. For the sake of simplicity, we do not consider actuators and temporal concerns.

The paper is organised as follows. In Sect. 2, we briefly introduce the process calculus IoT-LySA. In Sect. 3, we present a sample function that assigns rates to transitions, and we show how to obtain the CTMC associated with a given system of nodes and how to extract performance measures from it. Concluding remarks and related work are in Sect. 4.

2 IoT-LySA and its Enhanced Semantics

The original LySA calculus [5,7] is based on the π-[25] and Spi-calculus [1]. The main differences are: (i) the absence of channels, there is only one global communication medium to which all processes have access; (ii) the pattern matching tests associated with received and decrypted values are incorporated into inputs and into decryptions. Below, we assume that the reader is familiar with the basics of process calculi. We extend LySA to model IoT communications by introducing: (i) systems of nodes, consisting of (a representation of the) physical components, i.e. sensors, and of software control processes for specifying the *logic* of the node; (ii) primitives for reading from sensors also with cryptographic protection; (iii) global variables, i.e. whose scope is the whole node, to store data sent by sensors; (iv) a multi-communication modality among nodes (communications are subject to various constraints mainly about proximity); (v) functions to process and aggregate data, in particular the sensor's ones.

Syntax. As shown below, IoT-LySA systems have a two-level structure and consist of a fixed number of uniquely labelled nodes $N \in \mathcal{N}$ that host control processes $P \in \mathcal{P}$, and indexed sensor processes $S_i \in \mathcal{S}$ with $i \in \mathcal{I}_\ell$. Here $\mathcal{V}$ denotes the set of values, while $\mathcal{X}$ and $\mathcal{Z}$ and the local and the global variables, respectively. Finally, $\mathcal{L}$ denotes the set of node labels.

$N ::=$	*systems of nodes*	
	$\mathsf{0}$	nil
	$\ell : [P \parallel S]$	single node ($\ell \in \mathcal{L}$)
	$N_1 \mid N_2$	parallel composition of nodes

Intuitively, the null inactive system of nodes is denoted by $\mathsf{0}$ (*nil*); in a single node $\ell : [P \parallel S]$ the label ℓ uniquely identifies the node and represents further characterising information (e.g. its location or other contextual information if needed). Node components are obtained by the parallel composition (through the operator $|$) of control processes P, and of a fixed number of (less than $\#(\mathcal{I}_\ell)$) sensors S. The syntax of control processes follows.

$P ::=$	*control processes*	
	$\mathsf{0}$	nil
	$\langle E_1, \cdots, E_k \rangle . P$	intra-node output
	$\langle\!\langle E_1, \cdots, E_k \rangle\!\rangle \triangleright L . P$	multi-output L$\subseteq \mathcal{L}$
	$(E_1, \cdots, E_j ; x_{j+1}, \cdots, x_k) . P$	input (with match.)
	$P_1 \parallel P_2$	parallel composition of processes
	$P_1 + P_2$	summation
	$(\nu\, n)P$	restriction
	$A(y_1, \ldots, y_n)$	recursion
	$\mathsf{decrypt}\ E\ \mathsf{as}$	decryption (with match.)
	$\{E_1, \cdots, E_j ; x_{j+1}, \cdots, x_k\}_{E_0}\ \mathsf{in}\ P$	
	$(i ; z_i).P$	clear input from sensor i
	$(\{i ; z_i\}_K).P$	crypto input from sensor i

The process $\mathsf{0}$ represents the *inactive* process. The process $\langle E_1, \cdots, E_k \rangle . P$ sends the tuple $E_1, \cdots, E_k$ to another process in the same node and then continues like P. The process $\langle\!\langle E_1, \cdots, E_k \rangle\!\rangle \triangleright L.P$ sends the tuple $E_1, \ldots, E_k$ to the nodes whose labels are in L and evolves as P. Only nodes that are "compatible" (according, among other attributes, to a proximity-based notion) can communicate. The process $(E_1, \cdots, E_j ; x_{j+1}, \cdots, x_k) . P$ is willing to receive a tuple $E'_1, \cdots, E'_k$ with the same arity. The input primitive tests the first j terms of the received message; if they pairwise match with the first j terms of the input tuple, then the variables $x_{j+1}, \cdots, x_k$, occurring in the input tuple are bound to the corresponding terms $E_{j+1}, \cdots, E_k$ in the output tuple. The continuation is therefore $P\{E_{j+1}/x_{j+1}, \ldots, E_k/x_k\}$, where $\{-/-\}$ denotes, as usual, the standard substitution. Otherwise, the received tuple is not accepted. Note that for simplicity, all the pattern matching values precede all the binding variables: syntactically, a semi-colon separates the two components (see [9,4] for a more flexible choice). To better understand this construct, suppose to have a process P waiting for a message that P knows to include the value v and a value that P still does not know. The input pattern tuple would be: $(v ; x'_v)$. If P receives the matching tuple $\langle v, v' \rangle$, the pattern matching

succeeds and the variable x'_v is bound to v'. The operator $\|$ describes parallel composition of processes, while $+$ denotes non-deterministic choice. The operator (νa) acts as a static declaration for the name a in the process P the restriction prefixes. Restriction can be used to create new values, e.g. keys. An agent is a static definition of a parameterised process. Each agent identifier A has a unique defining equation of the form $A(y_1, \ldots, y_n) = P$, where $y_1, \ldots, y_n$ are distinct names occurring free in P. The process $\mathsf{decrypt}\ E\ \mathsf{as}\ E_1, \cdots, E_j; x_{j+1}, \cdots, x_k\}_{E_0}\ \mathsf{in}\ P$ receives an encrypted message. Also in this case we use the pattern matching but additionally the message $E = \{E'_1, \cdots, E'_k\}_{E'_0}$ is decrypted with the key E_0. Hence, whenever $E_i = E'_i$ for all $i \in [0, j]$, the receiving process behaves as $P\{E_{j+1}/x_{j+1}, \ldots, E_k/x_k\}$.

Sensors have the form:

$S ::=$	*sensor processes*	
	$\tau.S$	internal action
	$\langle i, v\rangle.\, S$	output of the i^{th} sensor
	$\langle\{i, v\}_K\rangle.\, S$	encrypted output of the i^{th} sensor
	$S_1 \parallel S_2$	parallel composition of sensors
	$A(y_1, \ldots, y_n)$	recursion

A sensor can perform an internal action τ or send a (encrypted) value v, gathered from the environment, to its controlling process and continues as S. We do not provide an explicit operation to read data from the environment but it can be easily implemented as an internal action.

Finally, the syntax of term is:

$E ::=$	*terms*	
	v	value ($v \in \mathcal{V}$)
	x	variable ($x \in \mathcal{X}$)
	z	sensor's variable ($z \in \mathcal{Z}$)
	$\{E_1, \cdots, E_k\}_{E_0}$	encryption ($k \geq 0$)
	$f(E_1, \cdots, E_n)$	function application ($n \geq 0$)

A value represents a piece of data, in particular we use them for keys, integers and values read from the environment. As said above, we have two kinds of disjoint variables: x are standard variables, i.e. as used in π-calculus; sensor variables z belong to a node and are globally accessible within it. As usual, we require that variables and names are disjoint. The encryption function $\{E_1, \cdots, E_k\}_{E_0}$ returns the result of encrypting values E_i for $i \in [1, k]$ with the key E_0. The term $f(E_1, \cdots, E_n)$ is the application of function f to n arguments; we assume given a set of primitive

aggregation functions, e.g. functions for comparing values or computing some metrics.

Working Example. Consider the scenario presented in the Introduction and illustrated in Fig. 1. We want to set a simple IoT system up in order to keep the temperature under control inside a storehouse with perishable food (a big quadrangular room). We plan to install four sensors: one for each corner of the storehouse. We assume that each sensor S_i periodically senses the temperature and sends it with a wireless communication to a control unit P_c in the same node, which aggregates the read values and checks if the average temperature is within accepted bounds. If this is not the case, the control unit sends an alarm throught other nodes and the Cloud. We assume that an attacker can intercept and manipulate data sent by sensors. A possible countermeasure is to exploit the fact that sensors on the same side should sense the same temperature, with a difference that can be at most a given value ϵ. The control unit can indeed easily detect anomalies and discard a piece of data manipulated by an attacker, by comparing it with values coming from the other sensor on the same side. But what if the attacker falsifies the data sent by more than one sensor? A possible solution consists in enabling a part of the sensors (in our example e.g. S_1 and S_3) to use cryptography in order to have at least two reliable data. Nevertheless, before adopting it or evaluating further solutions we would like to estimate the overhead cost. Sensors can be modelled in IoT-LySA as follows.

$$\begin{array}{ll} S_m = \langle m, sense_j = m() \rangle . \, \tau . S_m & m = 0, 2 \\ S_j = \langle \{ j, sense_j() \}_{K_j} \rangle . \, \tau . S_j & j = 1, 3 \end{array}$$

The control process P_c of the first node reads from sensors and then aggregates and compares the sensed values, in order to check them and compute their average. The control process Q_c of the second node verifies the result of the comparison and of the average functions and decides if sending an `alarm` or an `ok` message to the third node, together with the average value. The control process R_c of the third node represents an Internet service where the control process waits for the message of the second node and handles it (through the internal action τ). The specifications of the control processes follow.

$$\begin{array}{l} P_c = (0;\, z_0) . \, \tau . (\{ 1; z_1 \}_{K_1}) . \tau . (2;\, z_2) . \, \tau . (\{ 3; z_3 \}_{K_3}) . \tau . \\ \qquad \langle\!\langle cmp(z_0, ..., z_3), avg(z_0, ..., z_3) \rangle\!\rangle \triangleright \{ \ell_2 \} . \tau . P_c \\ Q_c = (\texttt{true}; x_{avg}) . \langle\!\langle \texttt{ok}, x_{avg} \rangle\!\rangle \triangleright \{ \ell_3 \} . \tau . Q_c \, + \\ \qquad (\texttt{false}; x_{avg}) . \langle\!\langle \texttt{alarm}, x_{avg} \rangle\!\rangle \triangleright \{ \ell_3 \} . \tau . Q_c \\ R_c = (; w_{res}, w_{avg}) . \tau . R_c \end{array}$$

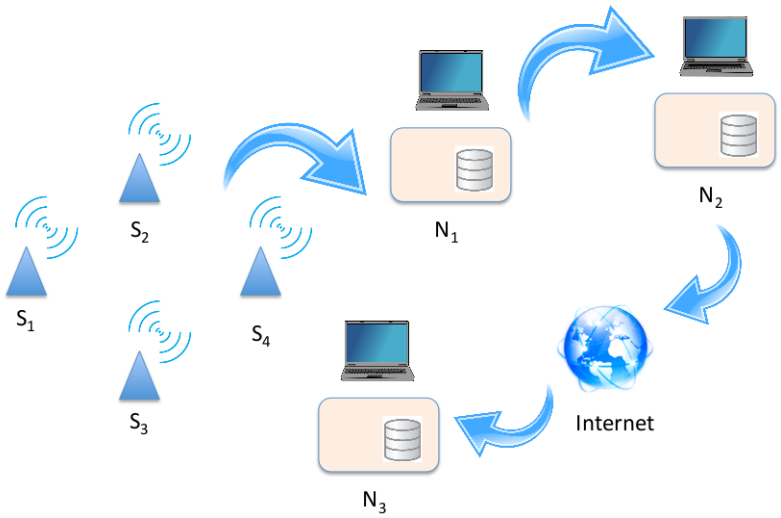

Fig. 1. The organisation of nodes in our refrigerator system.

The aggregation function *cmp* on the collected data perform consistency checks, by comparing data coming from insecure sensors with data coming from sensor endowed with encrypted communication; if the first data are out of bounds, the result is `true` otherwise is `false`. The function *avg* computes the average of its arguments. We suppose that processes and sensors perform some internal activities (denoted by τ-actions). The whole IoT-LySA node system, which includes the node N_1 (composed by the control unit and sensors) and the nodes N_2 and N_3, is specified as follows.

$$N = N_1 \mid N_2 \mid N_3 \qquad N_2 = \ell_2 : [Q_c \parallel 0]$$
$$N_1 = \ell_1 : [P_c \parallel (S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \qquad N_3 = \ell_3 : [R_c \parallel 0]$$

Another solution consists in enabling just one sensor on four to use cryptography. In the new system of nodes $\hat{N}$, the only difference is the following specification of the control process $\hat{P}_c$ of the first node.

$$\hat{P}_c = (0;\, z_0).\,\tau.(\{1; z_1\}_{K_1}).\tau.(2;\, z_2).\,\tau.(3;\, z_3).\,\tau.$$
$$\langle\!\langle halfcmp(z_0, z_1), avg(z_0, ..., z_3)\rangle\!\rangle \triangleright \{\ell_2\}.\tau.\hat{P}_c$$

Note that the required comparison function *halfcmp* is simpler, since it uses only two arguments. We expect that this second solution is less expensive. Our methodology allows us to formally compare the relative costs of the two solutions.

Enhanced Operational Semantics. Here, we give a reduction semantics in the style of the one of LySA [7]. It is an *enhanced* semantics, because following [14,15], transitions are annotated with labels used to estimate costs (actually, ours is a simplified version of the one in [14,15]).

The underlying idea is that each transition is enriched with an *enhanced label* θ, which records both the actions related to the transition and the possible nodes involved. Actually, there is a label for transitions involving communications and decryptions. More in detail, both point to point communications and multi-communications records the two actions (input and output) that lead to the transition, together with the labels of the corresponding nodes. Decryption actions store the label of the node performing the operation. Note that in the following definition and in the semantic rules, we use the abbreviations *out*, *in*, *dec*, for denoting the communication prefixes, the decryption constructs and, inside them, the possible function calls f, of the considered transition. The standard semantics can be obtained by simply removing the transition labels.

Definition 1. *(Transition labels) Given $\ell_O, \ell_I, \ell_D \in \mathcal{L}$, the set $\Theta \ni \theta$ of* enhanced labels *is defined as follows.*

$$
\begin{array}{lll}
\theta ::= & & \\
 & \langle \ell\ \{out\}, \ell\ \{in\} \rangle & \textit{internal secure communication} \\
 & \langle \ell\ out, \ell\ in \rangle & \textit{internal communication} \\
 & \langle \ell_O\ out, \ell_I\ in \rangle & \textit{intra-nodes communication} \\
 & \{\ell_D\ dec\} & \textit{decryption of a message}
\end{array}
$$

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

$$\langle\langle E_1, \cdots, E_k \rangle\rangle \triangleright \emptyset.\mathsf{0} :\equiv P$$

As usual, our *reduction relation* $\xrightarrow{\theta} \subseteq \mathcal{N} \times \mathcal{N}$ is defined as the least relation on closed nodes, processes and sensors that satisfies a set of inference rules. Our rules are quite standard apart from the five rules for communications shown in Tab. 1 and briefly commented below. We assume the standard denotational interpretation for evaluating terms $[\![E]\!]$.

- the rule (*Sens-Com*) is used for communications between sensors and processes. The used variables are assumed to be *global*. i.e. shared

among the process of the same node. The idea is that sensors contribute to a sort of shared data structure $z_1, \cdots, z_n$. Therefore the substitution is performed on all the control processes in the node. The transition label $\langle \ell \ out, \ell \ in \rangle$ records the fact that an internal communication occurred inside the node ℓ;

- similarly, the rule (*Crypto-Sens-Com*) is used for protected communications between sensors and processes: the value sensed by the sensor is encrypted before being sent to the process and is received if successfully decrypted. Also in this case the transition label $\langle \ell \ \{out\}, \ell \ \{in\} \rangle$ records information about the internal protected communication;
- the rule (*Intra-Com*) is used for communications internal to a node. The communication succeeds, provided that the first j values match with the evaluations of the first j terms in the input. When these comparisons are successful each E_i is bound to each x_i. The transition label $\langle \ell \ out, \ell \ in \rangle$ records the fact that an internal communication occurred inside the node ℓ;
- the rule (*Point2Point-Com*) is used for point to point communications between nodes. The communication succeeds, provided that (i) the labels of the two nodes are compatible according to the compatibility function *Comp*; and (ii) the first j values match with the evaluations of the first j terms in the input. When these comparisons are successful each E_i is bound to each x_i. The transition label $\langle \ell_1 \ out, \ell_2 \ in \rangle$ records the fact that an inter-node communication occurred between the nodes labelled by ℓ_1 and ℓ_2;
- the rule (*Multi-Com*) is used for multi-communications among nodes. The communication between the node labelled ℓ and the node ℓ' succeeds, provided that (i) ℓ' belongs to the set L of possible receivers, (ii) the two nodes are compatible according to the compatibility function *Comp*, and (iii) that the first j values match with the evaluations of the first j terms in the input. When these comparisons are successful, the first node spawns a new process, running in parallel with the continuation P, whose task is to offer the output tuple to all its receivers L, except for ℓ', which is removed, while in the second node each E_i is bound to each x_i. Outputs terminate when all the receivers in L have received the message (see the congruence rule). The transition label $\langle \ell_1 \ out, \ell_2 \ in \rangle$ records the fact that an inter-node communication occurred between the nodes ℓ_1 and ℓ_2 (with $\ell_2 \in L$).

The role of the compatibility function *Comp* is crucial in modelling real world constraints on communication. A basic requirement is that inter-node communications are mainly proximity-based, i.e. that only nodes

(Sensor-Com)

$$\ell : [\langle i, v_i\rangle.\, S_i \| \; S \parallel (i;\, z_i).\, P \mid Q] \xrightarrow{\langle \ell \; out, \ell \; in\rangle} \ell : [S_i \parallel S \| \; P\{v_i/z_i\} \mid Q\{E_i/z_i\}]$$

(Crypto-Sensor-Com)

$$\ell : [\langle \{i, v_i\}_K\rangle.\, S_i \| \; S \parallel (\{i; z_i\}_K).P \mid Q] \xrightarrow{\langle \ell \; \{out\}, \ell \; \{in\}\rangle} \ell : [S_i \parallel S \| \; P\{v_i/z_i\} \mid Q\{E_i/z_i\}]$$

(Intra-Com)

$$\frac{\bigwedge_{i=1}^{k} v_i = [\![E_i]\!] \;\wedge\; \bigwedge_{i=1}^{j} [\![E_i]\!] = [\![E'_i]\!]}{\ell : [\langle E_1, \cdots, E_k\rangle.\, P \;\mid\; (E'_1, \cdots, E'_j;\, x_{j+1}, \cdots, x_k).\, Q \| \; S]}$$

$$\xrightarrow{\langle \ell \; out, \ell \; in\rangle}$$

$$\ell : [P \;\mid\; Q\{v_{j+1}/x_{j+1}, \cdots, v_k/x_k\} \| \; S]$$

(Point2Point-Com)

$$\frac{Comp(\ell_1, \ell_2) \;\wedge\; \bigwedge_{i=1}^{k} v_i = [\![E_i]\!] \;\wedge\; \bigwedge_{i=1}^{j} [\![E_i]\!] = [\![E'_i]\!]}{\ell_1 : [\langle E_1, \cdots, E_k\rangle.\, P_{11} \mid P_{12} \parallel S_P] \mid \ell_2 : [(E'_1, \cdots, E'_j;\, x_{j+1}, \cdots, x_k).\, Q_{11} \mid Q_{12} \| \; S_Q]}$$

$$\xrightarrow{\langle \ell_1 \; out, \ell_2 \; in\rangle}$$

$$\ell_1 : [\langle E_1, \cdots, E_k\rangle.\, P_{11} \mid P_{12} \parallel S_P] \mid \ell_2 : [Q_{11}\{v_{j+1}/x_{j+1}, \cdots, v_k/x_k\} \mid Q_{12} \| \; S_Q]$$

(Multi-Com)

$$\frac{\ell_2 \in L \;\wedge\; Comp(\ell_1, \ell_2) \;\wedge\; \bigwedge_{i=1}^{k} v_i = [\![E_i]\!] \;\wedge\; \bigwedge_{i=1}^{j} [\![E_i]\!] = [\![E'_i]\!]}{\ell_1 : [\langle\!\langle E_1, \cdots, E_k\rangle\!\rangle \triangleright L.P_{11} \mid P_{12} \parallel S_P] \mid \ell_2 : [((E'_1, \cdots, E'_j; x_{j+1}, \cdots, x_k)).Q_{11} \mid Q_{12} \parallel S_Q]}$$

$$\xrightarrow{\langle \ell_1 \; out, \ell_2 \; in\rangle}$$

$$\ell_1 : [\langle\!\langle E_1, \cdots, E_k\rangle\!\rangle \triangleright L \setminus \{\ell_2\}.P_{11} \mid P_{12} \parallel S_P] \;||\; \ell_2 : [Q_{11}\{v_{j+1}/x_{j+1}, \cdots, v_k/x_k\} \mid Q_{12} \| \; S_Q]$$

Table 1. Operational semantic rules for communication.

that are in the same transmission range can directly exchange messages. This is easily encoded here by defining a predicate (over node labels) yielding true only when two nodes are in the same transmission range. Of course, this function could be enriched in order to consider finer notions of compatibility expressing various policies, e.g. topics for event notification.

Hereafter, we assume the standard notion of transition system. Intuitively, a transition system is a graph, in which systems of nodes form the nodes and arcs represent the possible transitions between them (in our cases arcs come with labels). For technical reasons, which will be clear in the next section, hereafter, we will restrict ourselves to *finite* state systems, i.e. whose corresponding transition systems have a finite set of states. Note that this does not mean that the behaviour of such processes is finite, because their transition systems may have loops.

Example (cont'd) Consider our simple running example and the single run of the first system (the one of the second system is similar) where the four sensors of node ℓ_1 send a message to their control process P_c and P_c checks the collected data and sends the checking result to the node with label ℓ_2. For brevity, we ignore their internal actions τ. We denote with P'_c (Q'_c, R'_c, resp.) the continuations of P_c (Q_c, R_c, resp.) after the first input prefixes, with v_{comp} the value $cmp(v_0, ..., v_3)$, with v_{avg} the value $avg(v_0, ..., v_3)$, and with v_{res_i} (with $i = 0, 1$) the value $\texttt{ok}$ ($\texttt{alarm}$ respectively), depending on which branch of the summation is chosen.

$$
\begin{array}{lcl}
N & = & \ell_1 : [(0;\, z_0).\, P'_c \mid P \parallel (\langle 0, sense_0() \rangle).\, \tau.S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid N_2 \mid N_3 \\
 & \xrightarrow{\theta_0} & \\
N' & = & \ell_1 : [P'_c\{0/z_0\} \parallel (\tau.S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid N_2 \mid N_3 \\
 & \xrightarrow{\theta_1}\xrightarrow{\theta_2}\xrightarrow{\theta_3} & \\
N'''' & = & \ell_1 : [P'_c\{0/z_0, 1/z_1, 2/z_2, 3/z_3\} \parallel \\
 & & (S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid N_2 \mid N_3 \\
 & = & \\
 & & \ell_1 : [\langle\!\langle v_{comp}, v_{avg} \rangle\!\rangle \triangleright \{\ell_2\}.\tau.P_c \parallel (S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid \\
 & & \ell_2 : [(\texttt{true}; x_{avg}).\langle\!\langle \texttt{ok}, x_{avg} \rangle\!\rangle \triangleright \{\ell_3\}.\tau.Q_c + \\
 & & \quad (\texttt{false}; x_{avg}).\langle\!\langle \texttt{alarm}, x_{avg} \rangle\!\rangle \triangleright \{\ell_3\}.\tau.Q_c] \mid N_3 \\
 & \xrightarrow{\theta_{4i}} & \\
N''''_i & = & \ell_1 : [P_c \mid P \parallel (S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid \\
 & & \ell_2 : [Q'_c\{v_{avg}/x_{avg}\}) \parallel \mathsf{0}] \mid \\
 & & \ell_3 : [(; w_{res}, w_{avg}).\tau.R_c \parallel \mathsf{0}] \\
 & = & \\
 & & \ell_1 : [P_c \mid P \parallel (S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid \\
 & & \ell_2 : [\langle\!\langle v_{res_i}, v_{avg} \rangle\!\rangle \triangleright \{\ell_3\}.\tau.Q_c \parallel \mathsf{0}] \mid \\
 & & \ell_3 : [(; w_{res}, w_{avg}).\tau.R_c \parallel \mathsf{0}] \\
 & \xrightarrow{\theta_{5i}} & \\
N & = & \ell_1 : [P_c \mid P \parallel (S_0 \parallel S_1 \parallel S_2 \parallel S_3)] \mid \\
 & & \ell_2 : [Q_c \parallel \mathsf{0}] \mid \\
 & & \ell_3 : [R'_c\{v_{res_i}/w_{res}, v_{avg}/w_{avg}\} \parallel \mathsf{0}] \\
 & = & \\
 & & N_1 \mid N_2 \mid N_3
\end{array}
$$

The whole sequence of transitions with source N is as follows.

$$
N \xrightarrow{\theta_0} N' \xrightarrow{\theta_1} N'' \xrightarrow{\theta_2} N''' \xrightarrow{\theta_3} N'''' \xrightarrow{\theta_{4i}} \begin{cases} N''''_0 \xrightarrow{\theta_{50}} N \text{ if } i = 0 \\ N''''_1 \xrightarrow{\theta_{51}} N \text{ if } i = 1 \end{cases}
$$

where $N', N'', N''', N'''', N'''''$ represent the derivatives of N (i.e. the node systems reached in the computation) and the label θ_j, which denotes the

label of the j^{th} transition (θ_{ji} depending on the branch of the summation), are as follows.

$$\theta_0 = \theta_2 = \langle \ell_1\langle j, v_j\rangle, \ell_1(j; z_i)\rangle,$$
$$\theta_1 = \theta_3 = \langle \ell_1\langle \{j, v_j\}_{K_i}\rangle, \ell_1(\{j; z_j\}_{K_i})\rangle,$$
$$\theta_{4i} = \langle \ell_1\langle\langle cmp(v_0, \cdots, v_3), avg(v_0, \cdots, v_3)\rangle\rangle, \ell_2(v_{bool}; x_{avg})\rangle$$
$$\theta_{5i} = \langle \ell_2\ \langle\langle v_{res_i}, v_{avg}\rangle\rangle, \ell_3(w_{res}, w_{avg})\rangle$$

The evolution of the second system $\hat{N}$ is similar and its transition labels $\hat{\theta}_i$ are as follows.

$$\hat{\theta}_0 = \hat{\theta}_2 = \hat{\theta}_3 = \langle \ell_1\langle j, v_j\rangle, \ell_1(j; z_i)\rangle,$$
$$\hat{\theta}_1 = \langle \ell_1\langle \{j, v_j\}_{K_i}\rangle, \ell_1(\{j; z_j\}_{K_i})\rangle,$$
$$\hat{\theta}_{4i} = \langle \ell_1\langle\langle halfcmp(v_0, v_1), avg(v_0, \cdots, v_3)\rangle\rangle, \ell_2(v_{bool}; x_{avg})\rangle$$
$$\hat{\theta}_{5i} = \langle \ell_2\ \langle\langle v_{res_i}, v_{avg}\rangle\rangle, \ell_3(w_{res}, w_{avg})\rangle$$

Note that in both cases, the transition systems loop, but they are *finite* as required.

3 Stochastic Semantics

We now show how to extract quantitative information from a transition system by transforming it in a Continuous Time Markov Chains (CTMC) (see [27] for a more detailed description of this process). First, we introduce functions that associate costs to single transitions, by inspecting their enhanced labels. This information is sufficient to extract the necessary quantitative information to obtain the Continuous Time Markov Chains (CTMC). In general, by "cost" we mean any measure that affects quantitative properties of transitions: here, we intend the time the system is likely to remain within a given transition. We specify the cost of a system in terms of the time overhead due to its primitives. The cost of (the component of) the transition depends on both the current action and on its context of executions, in our case, on the nodes involved. Intuitively, cost functions define exponential distributions of transitions. Starting from them it is possible to compute the rates at which a system evolves and therefore the corresponding CTMC. Finally, to evaluate the system performance we need to compute the (unique) stationary distribution of the CTMC and the transition rewards.

3.1 Cost Functions

First, we intuitively present the main factors that influence the costs of actions and those due to their context. For the sake of simplicity, here

we ignore the costs for other primitives, e.g. restriction, constant invocation, parallel composition, summation, and internal actions (see [27] for a complete treatment).

- The cost of a *communication* depends on the costs of the input and output components. In particular, the cost of an (i) *output* depends on the size of the message and on the cost of each term of the message sent, in particular on its encryptions; (ii) *input* depends on the size of the message and on the cost of checks needed to accept the message. Actually, the two partners independently perform some low-level operations locally to their environment, each of which leads to a delay. Since communication is synchronous and handshaking, the overall cost corresponds to the cost paid by the slower partner.
- The cost of both *encryption* and *decryption* depends on the sizes of the cleartext and ciphertext, respectively, the complexity of the algorithm that implements it, the cipher mode adopted, and the kind of the key (short/long, short-term/long-term). The length of the key is important: usually, the longer the key, the greater the computing time. In addition, the cost for *decryption* depends on the cost of the checks needed to accept the decryption.

To define a cost function, we start by considering the execution of each action on a dedicated architecture that only has to perform that action, and we estimate the corresponding duration with a fixed rate r. Then we model the performance degradation due to the run-time support. To do that, we introduce a scaling factor for r in correspondence with each routine called by the implementation of the transition θ under consideration. Here, we just propose a format for these functions, with parameters to be instantiated on need. Note that these parameters depend on the node, e.g. 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.

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 [29]). The *rate* r associated with the transition is the parameter that identifies the exponential distribution of the duration times of the transition, as usual in stochastic process algebras (e.g. [18,17]). 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 r, the faster $F(t)$ approaches its asymptotic value. The probability of performing an action with parameter r within time x is $F(x) = 1 - e^{-rx}$, so r determines the time, Δt, needed to have a probability near to 1. The exponential distributions that we use enjoy the *memoryless property*, i.e. the occurrence of a new transition does not depend on when the previous transitions occurred. We also assume that transitions are *time homogeneous*, i.e. the corresponding rates do not depend on the time in which the transitions are fired.

We define in a few steps the function that associates rates with communication and decryption transitions, or, more precisely, with their enhanced labels. For simplicity, we assume the sensor cost of sensing from the environment as non-significant. We first give the auxiliary function $f_E : \mathcal{E} \to \mathbb{R}^+$ that estimates the effort needed to manipulate terms $E \in \mathcal{E}$.

- $f_E(a) = size(a)$
- $f_E(\{E_1, \ldots, E_k\}_{E_0}) = f_{enc}(f_E(E_1), ..., f_E(E_1), kind(E_0))$

The size of a name a ($size(a)$) matters. For an encrypted term, we use the function f_{enc}, which in turn depends on the estimate of the terms to encrypt and on the kind of the key (represented by $kind(E_0)$), i.e. on its length and on the corresponding crypto-system. Then we assign costs to communication and decryption actions.

- $\$_\alpha(\langle E_1, \ldots, E_k \rangle) = f_{out}(f_E(E_1), ..., f_E(E_1), bw)$
- $\$_\alpha((E_1, \ldots, E_j; x_{j+1}, \ldots, x_k)) = f_{in}(f_E(E_1), ..., f_E(E_j), match(j), bw)$
- $\$_\alpha(\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j; x_{j+1}, \cdots, x_k\}_{E_0}) =$
  $f_{dec}(f_E(E), kind(E_0), match(j))$

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 communication channel (represented by bw) and the cost of the exchanged terms, which is computed by the auxiliary function f_E. They in turn depend on the nodes where the communication occurs. Moreover, the inter-node communication depends on the proximity-relationship between the nodes, represented here by the function $F(\ell_O, \ell_I)$. Also, the cost of an input depends on the number of tests or matchings required (represented by $match(j)$). Finally, the function f_{dec} represents the cost of a decryption. It depends on the manipulated terms ($f_E(E)$), on the kind of key ($kind(E_0)$), on the number of matchings ($match(j)$), and on the cryptographic features of the node that performs the decryption.

Finally, the function $\$: \Theta \to \mathbb{R}^+$ associates rates with enhanced labels.

- $\$(\langle \ell_O\ out, \ell_I\ in \rangle) = F(\ell_O, \ell_I) \cdot \min\{\$_\alpha(out, \ell_O), \$_\alpha(in, \ell_I)\}$
- $\$\langle \ell\ dec \rangle = \$_\alpha(dec, \ell)$

As mentioned above, the two partners independently perform some low-level operations locally to their nodes, represented by the two node labels ℓ_O and ℓ_I. Each label leads to a delay in the rate of the corresponding action. Thus, the cost of the slower partner corresponds to the minimum cost of the operations performed by the participants, in isolation. Indeed the lower the cost, i.e. the rate, the greater the time needed to complete an action and hence the slower the speed of the transition occurring. The smaller r, the slower $F(t) = 1 - e^{-rt}$ approaches its asymptotic value.

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.

Example (cont'd) We now associate a rate to each transition in the transition system of the system of nodes N, called N for brevity. To illustrate our methodology, we make some simplifying assumptions: we assume that τ transitions have no cost and that the coefficients due to the nodes amount to 1. We instantiate the cost functions given above, by using the following parameters each used to compute the rate corresponding to a particular action (sending, receiving and decryption) or a part of it, such as an encryption or a pattern matching: (i) `e` and `d` for encrypting and for decrypting; (ii) `s` and `r` for sending and for receiving; (iii) `m` for pattern matching; and (iv) `f` for the application of the aggregate function f, whose cost is proportional to the number of their arguments. The functions are:

- $f_E(a) = 1$
- $f_E(\{E_1, \ldots, E_k\}_{E_0}) = \frac{\mathtt{e}}{\mathtt{s}} \cdot \sum_{i=1}^{k} f_E(E_i) + 1$
- $\$_\alpha(\langle E_1, \ldots, E_k \rangle) = \frac{1}{\mathtt{s} \cdot \sum_{i=1}^{i} f_E(E_i)}$
- $\$_\alpha((E_1, \ldots, E_j; x_{j+1}, \ldots, x_k)) = \frac{1}{\mathtt{r} \cdot k + \mathtt{m} \cdot j}$
- $\$_\alpha(\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j; x_{j+1}, \cdots, x_k\}_{E_0}) = \frac{1}{\mathtt{d} \cdot k + \mathtt{m} \cdot j}$
- $\$_\alpha(f(E_1, \cdots, E_k)) = \frac{1}{\mathtt{f} \cdot k}$

Intuitively, these parameters represent the time spent performing the corresponding action on a single term. They occur in the denominator, therefore keeping the rule that the faster the time, the slower the rate. Since

transmission is usually more time-consuming than the corresponding reception, the rate of a communication, will always be that of output.
The rates of the transitions of N and $\hat{N}$ are $c_j = \$(\theta_j)$ and $\hat{c}_j = \$(\hat{\theta}_j)$, with $j \in [0,5]$ and $i \in [0,1]$.

$$
\begin{array}{ll}
c_0 = c_2 = \frac{1}{2\mathtt{s}}, & \hat{c}_0 = \hat{c}_2 = \hat{c}_3 = \frac{1}{2\mathtt{s}}, \\
c_1 = c_3 = \frac{1}{2\mathtt{e}+\mathtt{s}}, & \hat{c}_1 = \frac{1}{2\mathtt{e}+\mathtt{s}} \\
c_{4i} = \frac{1}{8\mathtt{f}+2\mathtt{s}} & \hat{c}_{4i} = \frac{1}{6\mathtt{f}+2\mathtt{s}} \\
c_{5i} = \frac{1}{\mathtt{s}} & \hat{c}_{5i} = \frac{1}{\mathtt{s}}
\end{array}
$$

For instance, the rate c_1 of the second transition is: $c_1 = \$(\theta_1) = \frac{1}{2\mathtt{e}+\mathtt{s}}$, where $\frac{1}{2\mathtt{e}+\mathtt{s}} = min\{\frac{1}{2\mathtt{e}+\mathtt{s}}, \frac{1}{2\mathtt{d}+\mathtt{r}+\mathtt{m}}\}$. Note that our costs can be further refined; we could e.g. use a different rate for transmission when internal to a node (costs c_j and $\hat{c}_j$ with $j \in [0,3]$) and when external (costs c_{ji} and $\hat{c}_{ji}$ with $j \in [3,4]$).

3.2 Stochastic Analysis

Now, we derive a Continuous Time Markov Chain (CTMC) from a transition system. Afterwards, we can calculate the actual performance measures, e.g. the throughput or utilisation of a certain resource (see [2,26] for more details on the theory of stochastic processes).

Markov Chains We use the rates of transitions computed in Subsect. 3.1, to transform a transition system N into its corresponding $CTMC(N)$.

Actually, the *transition rate* $q(N_i, N_j)$ at which a system changes from behaving like process N_i to behaving like N_j is the sum of the single rates ϑ_k of all the possible transitions from N_i to 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. Recall that a CTMC can be seen as a directed graph and that its matrix $\mathbf{Q}$, the *generator matrix*, (apart from its diagonal) represents its adjacency matrix. 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 indistinguishably CTMC and its corresponding $\mathbf{Q}$ to denote a Markov chain. More formally, the entries of $\mathbf{Q}$ are defined as follows. 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. Recall that a CTMC can be seen as a directed graph and that its matrix $\mathbf{Q}$, called *generator matrix*, (apart from its diagonal) represents its adjacency matrix. 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 indistinguishably CTMC and its corresponding $\mathbf{Q}$ to denote a Markov chain. More formally, the entries of $\mathbf{Q}$ are defined as follows.

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

Evaluating the Performance Performance measures should be taken over long periods of time to be significant. These measures are usually obtained by resorting to stationary probability distributions of $CTMCs$. The *stationary probability distribution* of a CTMC is $\mathit{\Pi} = (X_0, \ldots, X_{n-1})$ such that $\mathit{\Pi}$ solves the matrix equation $\mathit{\Pi}^T\mathbf{Q} = \mathbf{0}$ and $\sum_{i=0}^{n} X_i = 1$. If the transition system is finite and has a cyclic initial state, then there exists a unique stationary probability distribution.

Example (cont'd) Consider the transition system corresponding to the system of nodes N that is, as required above, finite and with a cyclic initial state. We derive the following generator matrix $\mathbf{Q_1}$ of $CTMC(N)$ and the corresponding stationary distribution is $\mathit{\Pi}_1$, where $C = 4\mathtt{s} + 2\mathtt{e} + 2\mathtt{f}$, by solving the system of linear equations $\mathit{\Pi}_1^T\mathbf{Q}_1 = \mathbf{0}$ and $\sum_{i=0}^{n} X_i = 1$, where $\mathit{\Pi}_1 = (X_0, \cdots, X_6)$. Similarly, we can derive the generator matrix $\hat{Q}_1'$ and the corresponding stationary distribution $\hat{\mathit{\Pi}}_1$ for the transition system corresponding to $\hat{N}$, where $\hat{C} = 9\mathtt{s} + 2\mathtt{e} + 3\mathtt{f}$.

$$\mathbf{Q_1} = \begin{bmatrix} -c_0 & c_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -c_1 & c_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -c_2 & c_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & -c_3 & c_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & -(c_{40} + c_{41}) & c_{40} & c_{41} \\ c_{50} & 0 & 0 & 0 & 0 & -c_{50} & 0 \\ c_{51} & 0 & 0 & 0 & 0 & 0 & -c_{51} \end{bmatrix}$$

$$\mathit{\Pi}_1 = \left[\frac{\mathtt{s}}{\mathtt{C}}, \frac{(2\mathtt{e} + \mathtt{s})}{2\mathtt{C}}, \frac{\mathtt{s}}{2\mathtt{C}}, \frac{(2\mathtt{e} + \mathtt{s})}{\mathtt{C}}, \frac{(4\mathtt{f} + \mathtt{s})}{2\mathtt{C}}, \frac{\mathtt{s}}{4\mathtt{C}}, \frac{\mathtt{s}}{4\mathtt{C}}\right]$$

$$\hat{\mathit{\Pi}}_1 = \left[\frac{2\mathtt{s}}{\hat{\mathtt{C}}}, \frac{(2\mathtt{e} + \mathtt{s})}{\hat{\mathtt{C}}}, \frac{2\mathtt{s}}{\hat{\mathtt{C}}}, \frac{2\mathtt{s}}{\hat{\mathtt{C}}}, \frac{(3\mathtt{f} + \mathtt{s})}{\hat{\mathtt{C}}}, \frac{\mathtt{s}}{2\hat{\mathtt{C}}}, \frac{\mathtt{s}}{2\hat{\mathtt{C}}}\right]$$

To define performance measures for a system N, we define a *reward structure* associated with N, following [19,18,12]. Usually, a reward structure is simply a function that associates a reward with any state passed

through in a computation of N. For instance, when calculating the utilisation of a resource, we assign value 1 to any state in which the use of the resource is enabled (typically the source of a transition that uses the resource). All the other states earn the value 0. We use a slightly different notion, where rewards are computed from rates of transitions [27]. To measure instead the throughput of a system, i.e. the amount of useful work accomplished per unit time, a reasonable choice is to use as nonzero reward a value equal to the rate of the corresponding transition. The reward structure of a system N is a vector of rewards with as many elements as the number of states of N. By looking at the probability stationary distribution of and varying the reward structure, we can compute different performance measures. The *total reward* is obtained by summing the values of the stationary distribution Π multiplied by the corresponding reward structure ρ.

Definition 2. *Given a system N, let $\Pi = (X_0, \ldots, X_{n-1})$ be its stationary distribution and $\rho = \rho(0), ..., \rho(n-1)$ be its reward structure. The* total reward *of N is computed as $R(N) = \sum_i \rho(i) \cdot X_i$.*

Example (cont'd) To evaluate the relative efficiency of the two systems of nodes, we compare the throughput of both, i.e. the number of instructions executed per time unit. The throughput for a given activity is found by first associating a transition reward equal to the activity rate with each transition. In our systems each transition is fired only once. Also, the graph of the corresponding CTMC is cyclic and all the labels represent different activities. This amounts to saying that the throughput of all the activities is the same, and we can freely choose one of them to compute the throughput of N. Thus we associate a transition reward equal to its rate with the last communication and a null transition reward with all the others communications. The total reward $R(N)$ of the system amounts then to $\frac{1}{2(8\mathtt{s}+4\mathtt{e}+4\mathtt{f})}$, while $R(\hat{N})$ amounts to $\frac{1}{2(9\mathtt{s}+2\mathtt{e}+3\mathtt{f})}$. By comparing the two throughputs, it is straightforward to obtain that $R(N) < R(\hat{N})$, i.e. that, as expected, $\hat{N}$ perform better. 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 energy consumption and on the guaranteed security level (see e.g. [24]).

4 Conclusions

In the IoT setting the risk for devices of being attacked is higher and higher, and still security is not taken sufficiently into account, since supporting security in an affordable way is quite challenging. We have presented the first steps towards a framework and formal design methodology that support designers in specifying an IoT system and in estimating the cost of security mechanisms starting from its specification. In this way, it suffices to have information about the activities performed by the components of a system in isolation, and about some features of the network architecture. 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 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 among different alternatives the one that ensures the best trade-off between security guarantees and their price.

In practice, we proposed the process algebra IoT-LySA, an extension of LySA with suitable primitives for describing the activity of sensors and of sensor nodes, and for describing the possible patterns of communication among the IoT entities. We have equipped the calculus with an enhanced semantics, following [15], where each system transition is associated to a rate in the style of [27]. Starting from the information about the rates of system activities, it is possible to mechanically derive Markov chains through which we can perform cost evaluation by using standard techniques and tools [33,30,32].

Our approach follows the well-established line of research about performance evaluation through process calculi and probabilistic model checking (see [13,20,21] for a survey). To the best of our knowledge, the application of formal methods to IoT systems or to wireless or sensor networks have not been largely studied and only a limited number of papers in the literature addressed the problem from a process algebras perspective, e.g. [22,23,10,31] to cite only a few. In [11] the problem of modelling and estimating the communication cost in an IoT scenario is tackled through Stochastic Petri Net. 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 [16] a performance compar-

ison between the security protocols IPSec and DTLS is presented, in particular by considering 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.

References

1. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols - The Spi calculus. *Information and Computation*, 148(1):1–70, 1999.
2. A. A. Allen. *Probability, Statistics and Queueing Theory with Computer Science Applications.* Academic Press, 1978.
3. M. Andreessen. *Why Software Is Eating The World.* The Wall Street Journal, August 20, 2011.
4. C. Bodei, L. Brodo, R. Focardi. Static Evidences for Attack Reconstruction. *Programming Languages with Applications to Biology and Security 2015*. LNCS 9465, pp.162-182, Springer, 2015.
5. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Automatic validation of protocol narration. *Proc. of CSFW'03*, pages 126–140. IEEE, 2003.
6. C. Bodei, M. Buchholtz, M. Curti, P. Degano, F. Nielson, and H. Riis Nielson, C. Priami. On Evaluating the Performance of Security Protocols *Proc. of PaCT'05*, LNCS 3606, pp. 115, 2005
7. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Static validation of security protocos. *Journal of Computer Security* 13(3): 347-390 (2005)
8. M. Bravetti, M. Bernardo and R. Gorrieri. Towards Performance Evaluation with General Distributions in Process Algebras. *Proc. of CONCUR'98*, LNCS 1466, 1998.
9. M. Buchholtz, F. Nielson, and H. Riis Nielson. A calculus for control flow analysis of security protocols. *International Journal of Information Security*, 2 (3-4), 2004.
10. V. Castiglioni, R. Lanotte and M. Merro. *A Semantic Theory for the Internet of Things.* arXiv:1510.04854v1.
11. 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): 3381-3385, 2012.
12. G. Clark. Formalising the specifications of rewards with PEPA. *Proc. of PAPM'96*, pp. 136-160. CLUT, Torino, 1996.
13. A. Clark, S. Gilmore, J. Hillston and M. Tribastone. *Stochastic Process Algebras.* Formal Methods for the Design of Computer, Communication, and Software Systems (SFM), 2007.
14. P. Degano and C. Priami. Non Interleaving Semantics for Mobile Processes. *Theoretical Computer Science*, 216:237–270, 1999.
15. P. Degano and C. Priami. Enhanced Operational Semantics. *ACM Computing Surveys*, 33, 2 (June 2001), 135-176.
16. A. De Rubertis, L. Mainetti,V. Mighali, L. Patrono,I. Sergi, M.L. Stefanizzi, S. Pascali, *Performance evaluation of end-to-end security protocols in an Internet of Things* in 21st International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2013.

17. H. Hermanns and U. Herzog and V. Mertsiotakis. Stochastic process algebras – between LOTOS and Markov Chains. *Computer Networks and ISDN systems* 30(9-10):901-924, 1998.
18. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
19. R. Howard. *Dynamic Probabilistic Systems: Semi-Markov and Decision Systems.* Volume II, Wiley, 1971.
20. M. Kwiatkowska, G. Norman and D. Parker. *Stochastic Model Checking.* Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07), LNCS 4486, 2007
21. M. Kwiatkowska and D. Parker. Advances in Probabilistic Model Checking. *Software Safety and Security - Tools for Analysis and Verification*: 33:126-151, IOS Press, 2012.
22. I. Lanese and D. Sangiorgi. An operational semantics for a calculus for wireless systems. *Theoretical Computer Science* 411(19): 1028-1948 (2010)
23. I. Lanese, L. Bedogni and M.D. Felice *Internet of Things: A Process Calculus Approach*. Proc. of the 28th Annual ACM Symposium on Applied Computing (ACM SAC '13), 2013
24. J. Lee, K. Kapitanova, S.H. Son: The price of security in wireless sensor networks. *Computer Networks* 54(17): 2967-2978 (2010)
25. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (I and II). *Information & Computation*, 100(1):1–77, 1992.
26. R. Nelson. *Probability, Stochastic Processes and Queeing Theory.* Springer, 1995.
27. C. Nottegar, C. Priami and P. Degano. Performance Evaluation of Mobile Processes via Abstract Machines. *Transactions on Software Engineering*, 27(10), 2001.
28. G. Plotkin. A Structural Approach to Operational Semantics. *Tech. Rep. Aarhus University, Denmark*, 1981, DAIMI FN-19
29. C. Priami. Language-based Performance Prediction of Distributed and Mobile Systems *Information and Computation* 175: 119-145, 2002.
30. A. Reibnam and R. Smith and K. Trivedi. Markov and Markov reward model transient analysis: an overview of numerical approaches. *European Journal of Operations Research*: 40:257-267, 1989.
31. A. Singh, C.R. Ramakrishnan and S.A. Smolka. A process calculus for Mobile Ad Hoc Networks. *Science of Computer Programming* 75(6): 440-469 (2010)
32. W. J. Stewart. *Introduction to the numerical solutions of Markov chains.* Princeton University Press, 1994.
33. K. S. Trivedi. *Probability and Statistics with Reliability, Queeing and Computer Science Applications.* Edgewood Cliffs, NY, 1982.