Tracking Data Trajectories in IoT

Chiara Bodei[1] and Letterio Galletta[2]
[1]*Dipartimento di Informatica, Università di Pisa, Italy*
[2]*IMT Institute for Advanced Studies Lucca, Italy*

Keywords: IoT, Static Analysis.

Abstract: The Internet of Things (IoT) devices access and process large amounts of data. Some of them are sensitive and can become a target for security attacks. As a consequence, it is crucial being able to trace data and to identify their paths. We start from the specification language IoT-LySA, and propose a Control Flow Analysis for statically predicting possible trajectories of data communicated in an IoT system and, consequently, for checking whether sensitive data can pass through possibly dangerous nodes. Paths are also interesting from an architectural point of view for deciding which are the points where data are collected, processed, communicated and stored and which are the suitable security mechanisms for guaranteeing a reliable transport from the raw data collected by the sensors to the aggregation nodes and to servers that decide actuations.

1 INTRODUCTION

In the Internet of Things (IoT), *things* are smart and interconnected devices that generate and transmit huge amounts of data over the net. Managing data is more complex than in traditional systems (see e.g. (Abu-Elkheir et al., 2013)): it consists in a production chain that starts from raw data collected by sensors, continues with aggregation nodes and possibly ends with servers that process data and decide consequent actuations. Secure communication of data is even more crucial in IoT scenarios, especially in multi-hop communications, where single nodes can be physically attacked and data can be eavesdropped or altered in passing. Thus it is important that IoT systems (i.e. networks of nodes, where each node interacts with the environment through sensors and actuators) are aware of the provenience and the trajectories of its data, especially when they are sensitive or when they impact critical decisions, such as stopping an industrial plant or the irrigation of a crop that uses precision agriculture technologies.

Usually, formal methods offers designers tools to support the development of systems and to reason about their properties. We follow this line of research by presenting preliminary results about using static analysis to study data trajectories in IoT systems. Technically, we start from the formal specification language IoT-LySA, a process calculus recently proposed for IoT systems (Bodei et al., 2016b; Bodei et al., 2017). IoT-LySA may help designers to adopt a *Security by Design* development model. Indeed, designers can exploit the calculus to model the structure of the system and how its components (smart objects) interact with each other. Furthermore, they can reason about the system correctness and robustness by using the Control Flow Analysis (CFA) of IoT-LySA. This static analysis safely approximates the system behaviour, by predicting how data from sensors may spread across the system and how objects may interact. Technically, 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. Designers can detect possible security vulnerabilities, inspecting this "abstract simulation" and intervene as early as possible during the design phase.

Here, we propose a variant of this CFA for *data path analysis*. This analysis predicts how data flows from specific data sources and which are their possible trajectories across the network nodes. It is then possible to investigate whether the predicted trajectories include nodes considered potentially dangerous from a security point of view. Moreover, it is possible to observe the trajectories of data used to make decisions in critical points of the system specification. Consequently, our analysis results may help designers in making educated decisions, on the exposure of both raw and aggregated data.

Because of over-approximatation if the predicted trajectories do not include dangerous nodes, we can be sure that at run time they will never be crossed. If instead they do, there is only the possibility of passing

Bodei, C. and Galletta, L.
Tracking Data Trajectories in IoT.
DOI: 10.5220/0007578305720579
In *Proceedings of the 5th International Conference on Information Systems Security and Privacy (ICISSP 2019)*, pages 572-579
ISBN: 978-989-758-359-9

by risky nodes, but it can be worthwhile to further investigate.

The paper is organised as follows. In Section 2, we introduce our methodology with an illustrative example. In Section 3 we briefly recall the process calculus IoT-LySA. In Section 4 we define the CFA, and reason on data trajectories. We conclude in Section 5.

2 AN IRRIGATION SYSTEM

In this section we illustrate our methodology through a simple yet realistic scenario similar to the one introduced in (Bodei et al., 2018). We consider a smart agricultural irrigation system, based on a Wireless Sensor Network, for monitoring and irrigating grapes crops. The main task of the system is to regulate irrigation according to *evapo-transpiration* (ET), a variable parameter that measures the crop water demand, which depends on several factors (e.g. weather, soil moisture, kind of plant and stage of development). Irrigation is needed when ET exceeds the supply of water coming from the soil or from precipitations.

In our model, a combination of wired and wireless sensors collects soil data like pH, moisture, temperature and so on. The collected data are sent in a multi-hop manner to a base station node. The base station node performs a first elaboration of data and then transmits the results to the remote server. The aggregated data are further processed and stored. The server decides the suitable irrigating actions for each sub-area in the crop: e.g. if, in one of them, the level of soil moisture goes down a given threshold then sprinkler actuators are activated. Users can directly access the server data at any location.

The corresponding model in IoT-LySA, described in Table 1, is a pool of nodes running in parallel (this is the meaning of the parallel composition operator $|$). Some of the terms are enriched with labels and tags that support the CFA and whose meaning will be clarified in the next section. Each node, uniquely identified by a label ℓ, consists of control processes and, possibly of sensors and actuators. Communication is multi-party: each node can send information to a set of nodes, provided that they are in the same transmission range. The communication patterns are not too complicate, so the example can serve the aim of illustrating our framework. Outputs and inputs must match in order to communicate. In more detail, output is modelled as $\langle\langle E_1,\cdots,E_k\rangle\rangle \triangleright L.P$ meaning that the tuple $E_1,\cdots,E_k$ is sent to the nodes with labels in L. Input is instead modelled as $(E_1,\cdots,E_j;\ x_{j+1},\cdots,x_k).P$ and embeds pattern matching. In receiving an output tuple $E'_1,\cdots,E'_k$ of the same size (arity), the communication succeeds provided that the first j elements of the output match the corresponding first elements of the input (i.e. $E_1 = E'_1,\cdots,E_j = E'_j$), and then the variables occurring in the input are bound to the corresponding terms in the output. Each base station node $N_{bs_{ij}}$ is connected to a bunch of sensors $S^1_{bs_{ij}},\cdots,S^k_{bs_{ij}}$ that sense the environment in the crop sub-area controlled by the base station control process $P_{bs_{ij}}$ and write the sensed values on its store. The node also includes other components that we omit because irrelevant here. Similarly, the action τ denotes internal actions of the sensor we are not interested in. The node N_{bs_i} collects data z^l of its sensors, processes them with the help of filter and aggregation functions $f_1,\cdots,f_m$ and transmits their results to the Cluster Head node N_{ch_j}. The communication is performed as explained above: e.g. the output $\langle\langle 1, f_1(z^1\cdots z^k),\cdots,f_m(z^1\cdots z^k)\rangle\rangle \triangleright \{\ell_{ch_1}\}$ performed by $P_{bs_{11}}$ matches the input $(1;x^1_1\cdots x^m_1)$ performed by P_{ch_1}, and therefore the variable x^1_1 is bound to $f_1(z^1\cdots z^k)$, the variable x^2_1 is bound to $f_2(z^1\cdots z^k)$ and so on. The construct $*[...]*$ implements the iterative behaviour of processes and of sensors. Each node N_{ch_j} controls a subset of the base station nodes $N_{bs_{ij}},\cdots,N_{bs_{ij}}$ that send it their data ($\vec{x_i}$ stands for the array $(x^1_i\cdots x^m_i)$). The Cluster Head node receives these data, aggregates them with the functions $aggr_1(),\cdots,aggr_r()$, and then sends the results to the Server N_{as}. The node N_{as} processes the data sent by all the Cluster Head nodes $N_{ch_1},\cdots,N_{ch_n}$ and makes its decision on irrigation. If the server detects that some water is needed in the area controlled by the node N_{ch_j} (i.e. if the water demand w_{j1} *exceeds* a given threshold th_{j1}), it sends a "start irrigation" order, and conversely, when it detects that there is sufficient water (i.e. if the water demand w_{j1} is *does not exceed* a given threshold th'_{j1}), it transmits a "stop irrigation" order. The relevant Cluster Head node N_{ch_j} transmits the received orders to the corresponding actuators A_j triggering the irrigation sprinklers. Furthermore, users N_{us_t} can access the data processed and stored by the server N_{as}, e.g. for checking whether there are the conditions for particular manual processes, like pruning. For the sake of simplicity, we omit the specification of the components of the users' nodes N_{us_t}.

Wireless communication depends on the transmission range. Secure data transfer between the sensors and the server, through base station nodes, is crucial.

Some nodes can be insecure and therefore may alter or tamper data passing from there, thus potentially impacting on the whole irrigation system. As a conse-

Table 1: Irrigation Control System $N_{as} \mid N_{ch_1} \mid (N_{bs_{11}} \mid \cdots N_{bs_{s1}}) \mid \cdots N_{ch_n} \mid (N_{bs_{1n}} \mid \cdots N_{bs_{sn}}) \mid N_{us_1} \mid \cdots N_{us_h}$.

Base Station Node i of Cluster Head j with $i \in [1,s], l \in [1,k], j \in [1,n]$

$N_{bs_{ij}} = \ell_{bs_{ij}} : [P_{bs_{ij}} \parallel S^1_{bs_{ij}} \cdots \parallel S^k_{bs_{ij}}]$

$P_{bs_{ij}} = *[(z^1 := 1^{a_{1ij}}). \cdots .(z^k := k^{a_{kij}}).\langle\langle i^{b_{ij}}, f_1(z^1 \cdots z^k)^{f_{1ij}}, \cdots, f_m(z^1 \cdots z^k)^{f_{mij}}\rangle\rangle \triangleright \{\ell_{ch_j}\}]*$

$S^l_{bs_{ij}} = *[(\tau.l_j := v_{lj}).\tau]*$

Cluster Head j with $j \in [1,n]$

$N_{ch_j} = \ell_{ch_j} : [P_{ch_j} \parallel P'_{ch_j} \parallel A_j]$

$P_{ch_j} = *[(1; x^1_1 \cdots x^m_1)^{X_{j1}} \cdot \ldots .(s; x^1_s \cdots x^m_s)^{X_{js}} . \langle\langle j^{c_j}, aggr_1(\vec{x_1}, \cdots, \vec{x_s})^{g_{1j}}, \cdots, aggr_r(\vec{x_1}, \cdots, \vec{x_s})^{g_{rj}}\rangle\rangle \triangleright \{\ell_{as}\}]*$

$P'_{ch_j} = *[(j; x)^{X'_j} . \langle j, x\rangle]*$

$A_j = *[(\lfloor j, \{\mathsf{StartIrrigation}, \mathsf{StopIrrigation}\}\rfloor)]*$

Server

$N_{as} = \ell_{as} : [\Sigma_{as} \parallel P_{as,1} \parallel \cdots \parallel P_{as,n} \parallel P_{as,us_t}]$

$P_{as,1} = *[\quad (1; w_{11}, \cdots, w_{1r})^{Y_1} \parallel$
$(w_{11} \geq th_{11})\ ?$
$\langle 1, \mathsf{StartIrrigation}\rangle :$
$(w_{11} < th'_{11})\ ?$
$\langle 1, \mathsf{StopIrrigation}\rangle]*$

$\ldots$

$P_{as,n} = *[\quad (n; w_{n1}, \cdots, w_{nr})^{Y_n} \parallel$
$(w_{n1} \geq th_{n1})\ ?$
$\langle n, \mathsf{StartIrrigation}\rangle :$
$(w_{n1} < th'_{n1})\ ?$
$\langle 1, \mathsf{StopIrrigation}\rangle]*$

User t with $t \in [1,h]$ $N\_us_t = \ell_{us_t} : [P_{us_t}]$

quence the grape crop can be also heavily damaged. Since our analysis identifies the possible trajectories of data in the system, we can check whether these trajectories include dangerous nodes.

3 THE CALCULUS IoT-LySA

Here, we briefly review the process calculus IoT-LySA (Bodei et al., 2016b; Bodei et al., 2017). IoT-LySA is an adaption of LySA (Bodei et al., 2005), a process algebra introduced to specify and analyse cryptographic protocols and checking their security properties (Gao et al., 2007; Gao et al., 2008). Differently from other process calculus approaches to IoT, e.g. (Lanese et al., 2013; Lanotte and Merro, 2016; Lanotte and Merro, 2018), IoT-LySA aims at providing a design framework that includes a static semantics to support verification techniques and tools for certifying properties of IoT applications.

Systems in IoT-LySA consist of a finite number of nodes, each of which hosts a store for internal communication and a finite number of control processes (representing the software), sensors and actuators. We assume that each sensor (actuator) in a node with label ℓ is uniquely identified by an index $i \in I_\ell$ ($j \in \mathcal{J}_\ell$, resp). Data are represented by terms. Annotations $a, a', a_i, ...$, ranged over by $\mathcal{A}$, which identify the occurrences of terms, are used in the analysis and do not affect the semantics. The syntax in presented in Table 2.

We assume as given a finite set $\mathcal{K}$ of secret keys owned by nodes, exchanged at deployment time in a secure way, as it is often the case. Terms come with annotations $a \in \mathcal{A}$. The encryption function $\{E_1, \cdots, E_r\}_{k_0}$ returns the result of encrypting values E_i for $i \in [1,r]$ under the shared key k_0. We assume to have perfect cryptography. The term $f(E_1, \cdots, E_r)$ is the application of function f to r arguments; we assume given a set of primitive functions, typically for aggregating or comparing values. We assume the sets $\mathcal{V}$, I_ℓ, $\mathcal{J}_\ell$, $\mathcal{K}$ be pairwise disjoint.

A node $\ell : [B]$ is uniquely identified by a label $\ell \in \mathcal{L}$ that may represent further characterising information (e.g. node location). Sets of nodes are described through the (associative and commutative) operator $\mid$ for parallel composition. The system 0 has no nodes. Inside a node $\ell : [B]$ there is a finite set of components described by the parallel operator $\parallel$. We impose that there is a *single* store $\Sigma_\ell : \mathcal{X} \cup I_\ell \rightarrow \mathcal{V}$, where $\mathcal{X}, \mathcal{V}$ are the sets of variables and of values resp.

The store is essentially an array whose indexes are variables and sensors identifiers $i \in I_\ell$. We assume that store accesses are atomic, e.g. through CAS instructions (Herlihy, 1991). The other node components are control processes P, and sensors S (less than $\#(I_\ell)$), and actuators A (less than $\#(\mathcal{J}_\ell)$) the actions of which are in Act.

The prefix $\langle\langle E_1, \cdots, E_r\rangle\rangle \triangleright L$ implements a simple form of multi-party communication: the tuple obtained by evaluating $E_1, \ldots, E_r$ is asynchronously sent to the nodes with labels in L that are "compatible" (ac-

Table 2: Syntax.

$\mathcal{E} \ni E$	$::=$ *annotated terms*		$\mathcal{M} \ni M,N$	$::=$ *terms*	
	M^a	annotated term with $a \in \mathcal{A}$		v	value ($v \in \mathcal{V}$)
				i	sensor location ($i \in I_\ell$)
				x	
				$\{E_1,\cdots,E_r\}_{k_0}$	encryption with key $k_0 \in \mathcal{K}$
				$f(E_1,\cdots,E_r)$	function on data
$\mathcal{N} \ni N ::=$	*systems of nodes*		$\mathcal{B} \ni B ::=$	*node components*	
	0	empty system		Σ_ℓ	node store
	$\ell : [B]$	single node ($\ell \in \mathcal{L}$)		P	process
	$N_1 \mid N_2$	par. composition		S	sensor (label $i \in I_\ell$)
				A	actuator (label $j \in \mathcal{J}_\ell$)
				$B \parallel B$	par. composition

$P ::=$	*control processes*	
	0	inactive process
	$\langle\langle E_1,\cdots,E_r\rangle\rangle \triangleright L.P$	asynchronous multi-output $L \subseteq \mathcal{L}$
	$(E_1,\cdots,E_j;x_{j+1},\cdots,x_r)^X.P$	input (with matching and tag)
	$\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1,\cdots,E_j;x_{j+1},\cdots,x_r\}_{k_0}\ \mathsf{in}\ P$	decryption with key k_0 (with match.)
	$E?P:Q$	conditional statement
	h	iteration variable
	$\mu h.\ P$	tail iteration
	$x^a := E.P$	assignment to $x \in \mathcal{X}$
	$\langle j,\gamma\rangle.P$	output of action γ to actuator j

cording, among other attributes, to a proximity-based notion). The input prefix $(E_1,\cdots,E_j;x_{j+1},\cdots,x_r)^X$ receives a r-tuple, provided that its first j elements match the corresponding input ones, and then assigns the variables (after ";") to the received values. Otherwise, the r-tuple is not accepted. As in (Bodei et al., 2015), each input in the syntax of processes P has a tag $X \in \mathbf{X}$, which is exploited to support the analysis and does not affect the dynamic semantics. A process repeats its behaviour, when defined through the tail iteration construct $\mu h.P$ (h is the iteration variable), intuitively rendered with $*[...]*$ in the motivating example. The process $\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1,\cdots,E_j;x_{j+1},\cdots,x_r\}_{k_0}\ \mathsf{in}\ P$ tries to decrypt the result of the expression E with the shared key $k_0 \in \mathcal{K}$. If the pattern matching succeeds, the process continues as P and the variables $x_{j+1},\ldots,x_r$ are suitably assigned.

A sensor can perform an internal action τ or put the value v, gathered from the environment, into its store location i. An actuator can perform an internal action τ or execute one of its actions γ, received from its controlling process. Sensors and actuators can iterate.

The semantics is based on a standard structural congruence and a two-level *reduction relation* $\rightarrow$ defined as the least relation on nodes and its components, where we assume the standard denotational interpretation $[\![E]\!]_\Sigma$ for evaluating terms. As examples of semantic rules, we show the rules (Ev-out) and (Multi-com) in Table 3, that drive asynchronous IoT-LySA multi-communications. In the first rule, to send a message $\langle\langle v_1,...,v_r\rangle\rangle$ obtained by the evaluation of $\langle\langle E_1,...,E_r\rangle\rangle$, a node with label ℓ spawns a new process, running in parallel with the continuation P; this new process offers the evaluated tuple to all the receivers with labels in L. In the second rule, the message coming from ℓ_1 is received by a node labelled ℓ_2, provided that: (i) ℓ_2 belongs to the set L of possible receivers, (ii) the two nodes satisfy a compatibility predicate *Comp* (e.g. when they are in the same transmission range), and (iii) that the first j values match with the evaluations of the first j terms in the input. Moreover, the label ℓ_2 is removed by the set of receivers L of the tuple. The spawned process terminates when all the receivers have received the message ($L = \emptyset$).

4 CONTROL FLOW ANALYSIS

Here we present a CFA for approximating the abstract behaviour of a system of nodes and for tracking the trajectories of data. This CFA follows the same schema of the one in (Bodei et al., 2016b; Bodei et al., 2016a) and in particular of the one in (Bodei and Galletta, 2017) for IoT-LySA. However, here we use different abstract values. Intuitively, abstract values "symbolically" represent runtime data so as to encode where these data have been introduced. Finally, we show how to use the CFA results to check which are the possible trajectories of these data.

Abstract values correspond to concrete values for sensors, data, functions, and encryptions, and also record the annotations. Since the dynamic seman-

Table 3: Communication semantic rules.

(Ev-out)

$$\frac{\bigwedge_{i=1}^{r} v_i = [\![E_i]\!]_\Sigma}{\Sigma \parallel \langle\langle E_1,\cdots,E_r\rangle\rangle \triangleright L.P \parallel B \;\rightarrow\; \Sigma \parallel \langle\langle v_1,\cdots,v_r\rangle\rangle \triangleright L.0 \parallel P \parallel B}$$

(Multi-com)

$$\frac{\ell_2 \in L \wedge\ Comp(\ell_1,\ell_2) \wedge \bigwedge_{i=1}^{j} v_i = [\![E_i]\!]_{\Sigma_2}}{\begin{array}{c}\ell_1 : [\langle\langle v_1,\cdots,v_r\rangle\rangle \triangleright L.0 \parallel B_1] \;\mid\; \ell_2 : [\Sigma_2 \parallel (E_1,\cdots,E_j;x_{j+1}^{a_{j+1}},\cdots,x_r^{a_r})^X.Q \parallel B_2] \;\rightarrow \\ \ell_1 : [\langle\langle v_1,\cdots,v_r\rangle\rangle \triangleright L \setminus \{\ell_2\}.0 \parallel B_1] \;\mid\; \ell_2 : [\Sigma_2\{v_{j+1}/x_{j+1},\cdots,v_r/x_r\} \parallel Q \parallel B_2]\end{array}}$$

tics may introduce encrypted terms with an arbitrarily nesting level, we have the special abstract values $\top^a$ that denote all the terms with a depth greater than a given threshold d. During the analysis, to cut these values, we will use the function $\lfloor - \rfloor_d$, defined as expected. Formally, abstract values are defined as follows, where $a \in \mathcal{A}$.

$\hat{\mathcal{V}} \ni \hat{v} ::=$ *abstract terms*

$(\top, a)$	value denoting cut
(v, a)	value for clear data
$(f(\hat{v}_1,\cdots,\hat{v}_n), a)$	value for aggregated data
$(\{\hat{v}_1,\cdots,\hat{v}_n\}_{k_0}, a)$	value for encrypted data

For simplicity, hereafter we write them as $\top^a, \nu^a, \{\hat{v}_1,\cdots,\hat{v}_n\}^a_{k_0}$, and indicate with $\downarrow_i$ the projection function on the i^{th} component of the pair. We naturally extend the projection to sets, i.e. $\hat{V}_{\downarrow_i} = \{\hat{v}_{\downarrow_i} | \hat{v} \in \hat{V}\}$, where $\hat{V} \subseteq \hat{\mathcal{V}}$. In the abstract value v^a, v abstracts the concrete value from sensors or computed by a function in the concrete semantics, while the first value of the pair $\{\hat{v}_1,\cdots,\hat{v}_n\}^a_{k_0}$ abstracts encrypted data. The second component records the annotation associated to the corresponding term. Note that once given the set of encryption functions occurring in a node N, the abstract values are finitely many.

To extract all the annotations of an abstract value, included the ones possibly nested in it, we use the following auxiliary function.

Definition 4.1. *Give an abstract value $\hat{v} \in \hat{\mathcal{V}}$, we define the set of labels $\mathbf{A}(\hat{v})$ as follows.*

- $\mathbf{A}(\top, a) = \mathbf{A}(v, a) = \{a\}$
- $\mathbf{A}(f(\hat{v}_1,\cdots,\hat{v}_n), a) = \{a\} \cup \bigcup_{i=1}^{n} \mathbf{A}(\hat{v}_i)$
- $\mathbf{A}(\{\hat{v}_1,\cdots,\hat{v}_n\}_{k_0}, a) = \{a\} \cup \bigcup_{i=1}^{n} \mathbf{A}(\hat{v}_i)$

Trajectories. We now introduce the notion of data trajectories, composed by micro-trajectories representing single communication hops.

Definition 4.2. *Given a set of labels $\mathcal{L}$, a set of input tags $\mathbf{X}$, we define a* micro-trajectory *μ as a pair $((\ell,\ell'),X) \in (\mathcal{L} \times \mathcal{L}) \times \mathbf{X}$. A* trajectory *$\tau$ is a list of micro-trajectories $[\mu_1,...,\mu_n]$, such that $\forall \mu_i, \mu_{i+1}$ with $\mu_i = ((\ell_i,\ell'_i),X_i)$ and $\mu_{i+1} = ((\ell_{i+1},\ell'_{i+1}),X_{i+1})$, $\ell'_i = \ell_{i+1}$.*

In our analysis, trajectories can be obtained, starting from a set of micro-trajectories and by suitably composing them in order. Trajectories can be composed if the head of the second trajectory is equal to tail of the first. In this case the two trajectories can be merged. Technically, we use a closure of a set of micro-trajectories, the inductive definition of which follows.

Definition 4.3.

- $\forall((\ell,\ell'),X \in M.\ [((\ell,\ell'),X)] \in Clos_X(M)$*;*
- $\forall[L,((\ell,\ell'),X)],\ [((\ell',\ell''),X'),L''] \in M.\ [L,((\ell,\ell'),X),(\ell',\ell''),X'),L''] \in Clos_X(M)$.

We assume that designers provide the analysis with a classification of the "dangerous" nodes and links or of bad flows.

CFA Validation and Correctness. We now have all the ingredients to define our CFA to approximate communications and data stored and exchanged and, in particular, the micro-trajectories. We specify our analysis in a logical form through a set of inference rules expressing the validity of the analysis results. The analysis result or *estimate* is a tuple $(\widehat{\Sigma}, \kappa, \Theta, T, \rho)$ (a pair $(\widehat{\Sigma}, \Theta)$ when analysing a term), where $\widehat{\Sigma}$, κ, Θ, T, ρ are the following *abstract domains*:

- the union $\widehat{\Sigma} = \bigcup_{\ell \in \mathcal{L}} \hat{\Sigma}_\ell$ of the sets $\hat{\Sigma}_\ell : \mathcal{X} \cup I_\ell \rightarrow 2^{\widehat{\mathcal{V}}}$ of abstract values that may possibly be associated to a given location in I_ℓ or a given variable in $\mathcal{X}$,
- a set $\kappa : \mathcal{L} \rightarrow \mathcal{L} \times \bigcup_{i=1}^{k} \widehat{\mathcal{V}}^i$ of the messages that may be received by the node ℓ, and
- a set $\Theta : \mathcal{L} \rightarrow \mathcal{A} \rightarrow 2^{\widehat{\mathcal{V}}}$ of the information of the actual values computed by each labelled term M^a in a given node ℓ, at run time.
- a set $\rho : \mathbf{X} \rightarrow \mathcal{L} \times \bigcup_{i=1}^{k} \widehat{\mathcal{V}}^i$ is the sets of output tuples that may be accepted by the input variables X.
- a set $T = \mathcal{A} \rightarrow (\mathcal{L} \times \mathcal{L}) \times \mathbf{T}$ of possible micro-trajectories related to the abstract values.

The component T is new, and also the combined use of these five components is new and allows us to potentially integrate the present CFA with the previous analyses of IoT-LySA.

An available estimate has to be validated correct. This requires that it satisfies the judgements defined according to the syntax of nodes, node components and terms. They are defined by a set of clauses. Here, we just show some examples. The judgements for labelled terms have the form $(\widehat{\Sigma},\Theta) \models_\ell M^a$. For each term M^a occurring in the node ℓ, the corresponding judgement requires that $\Theta(\ell)(a)$ includes all the abstract values $\hat{v}$ associated to M^a, e.g. if the term is x^a, $\Theta(\ell)(a)$ includes the abstract values bound to x collected in $\widehat{\Sigma}_\ell$. The judgements for nodes have the form $(\widehat{\Sigma},\kappa,\Theta,T,\rho) \models N$. The rule for a single node $\ell : [B]$ requires that B is analysed with judgements $(\widehat{\Sigma},\kappa,\Theta,T,\rho) \models_\ell B$. As examples of clauses, we consider the clauses for communication in Table 4.

An estimate is valid for *multi-output*, if it is valid for the continuation of P and the set of messages communicated by the node ℓ to each node ℓ' in L, includes all the messages obtained by the evaluation of the r-tuple $\langle\langle M_1^{a_1},\cdots,M_r^{a_r}\rangle\rangle$. More precisely, the rule (i) finds the sets $\Theta(\ell)(a_i)$ for each term $M_i^{a_i}$, and (ii) for all tuples of values $(\hat{v}_1,\cdots,\hat{v}_r)$ in $\Theta(\ell)(a_1)\times\cdots\times\Theta(\ell)(a_r)$ it checks whether they belong to $\kappa(\ell')$ for each $\ell' \in L$. Symmetrically, the rule for *input* requires that the values inside messages that can be sent to the node ℓ, passing the pattern matching, are included in the estimates of the variables $x_{j+1},\cdots,x_r$. More in detail, the rule analyses each term $M_i^{a_i}$, and requires that for any message that the node with label ℓ can receive, i.e. $(\ell',\langle\langle\hat{v}_1,\cdots,\hat{v}_j,\hat{v}_{j+1},\ldots,\hat{v}_r\rangle\rangle)$ in $\kappa(\ell)$, provided that the two nodes can communicate (i.e. $Comp(\ell',\ell)$), the abstract values $\hat{v}_{j+1},\ldots,\hat{v}_r$ are included in the estimates of $x_{j+1},\cdots,x_r$. Furthermore, the micro-trajectory $((\ell,\ell'),X)$ is recorded in the T component for each annotation related (via **A**) to the abstract value $\hat{v}_i$, to record that the abstract value $\hat{v}_i$ coming from the node ℓ can reach the node labelled ℓ', in the input with tag X, e.g. if $\hat{v}_i = (f((v_{i1},a_{i1}),(v_{i2},a_{i2})),a_i)$, then the micro-trajectory is recorded in $T(a_i)$, $T(a_{i1})$ and $T(a_{i2})$. Finally, the ρ component records the sets of output tuples that can be bound in the input with tag X.

Example 4.4. *In our running example, every valid estimate* $(\widehat{\Sigma},\kappa,\Theta,\rho,T)$ *must include at least the following entries, assuming* $d = 4$.

$$\Theta(\ell_{bs_{ij}})(a_{1ij}) \supseteq \{i^{a_{1ij}}\}$$
$$\hat{\Sigma}_{\ell_{bs_{ij}}}(z^i) \supseteq \{i^{a_{ij}}\}$$
$$\rho(X_{ji}) \supseteq \{(\ell_{ch_j},\langle\langle i^{b_{ij}},f_1(1^{a_{1ij}},...,k^{a_{kij}})^{f_{1ij}},...\rangle\rangle\}$$
$$\kappa(\ell_{ch_j}) \supseteq \{(\ell_{ch_j},\langle\langle i^{b_{ij}},f_1(1^{a_{1ij}},...,k^{a_{kij}})^{f_{1ij}},...\rangle\rangle\}$$
$$\hat{\Sigma}_{\ell_{ch_j}}(x_i^w) \supseteq \{f_i(i^{a_{1ij}})\}$$
$$\kappa(\ell_{as}) \supseteq \{(\ell_{ch_j},\langle\langle f^{c_j},aggr_1(\vec{f_{1j}})^{g_{1j}},...,aggr_r(\vec{f_{sj}})^{g_{rj}}\rangle\rangle\}$$
$$\hat{\Sigma}_{\ell_{ch_j}}(w^{jt}) \supseteq \{aggr_1(\vec{f_{ij}})^{g_{1j}}\}$$
$$T(a_{1ij}) \ni ((\ell_{bs_{ij}},\ell_{ch_j}),X_{ji}),((\ell_{ch_j},\ell_{as}),Y_j)$$

Indeed, an estimate must satisfy the checks of the CFA rules. The validation of the system requires, in particular, that $i^{a_{1ij}}$ *is in* $\hat{\Sigma}_{\ell_{bs_{ij}}}(z)$ *for the rule for variables, while for the rule for output, the inclusion in* $\kappa(\ell_{ch_j})$*, and so on.*

Our analysis respects the operational semantics of IoT-LySA, as witnessed by the following subject reduction result. It is also possible to prove the existence of a (minimal) estimate, as in (Bodei et al., 2016b). The proofs follow the usual schema and benefit from an instrumented denotational semantics for expressions, the values of which are pairs $\langle v,\hat{v}\rangle$, where v is a concrete value and $\hat{v}$ is the corresponding abstract value. The store (Σ_ℓ^i with an undefined $\perp$ value) is accordingly extended. The semantics uses the projection on the first component.

The following subject reduction theorem establishes the correctness of our CFA, by relying on the agreement relation $\bowtie$ between the concrete and the abstract stores. Its definition is immediate, since the analysis only considers the second component of the extended store, i.e. the abstract one: $\Sigma_\ell^i \bowtie \widehat{\Sigma}_\ell$ iff $w \in \mathcal{X}\cup I_\ell$ such that $\Sigma_\ell^i(w) \neq \perp$ implies $(\Sigma_\ell^i(w))_{\downarrow_2} \in \widehat{\Sigma}_\ell(w)$.

Theorem 4.5 (Subject Reduction)**.** *If* $(\widehat{\Sigma},\kappa,\Theta,T,\rho) \models N$ *and* $N \to N'$ *and* $\forall\Sigma_\ell^i$ *in* N *it is* $\Sigma_\ell^i \bowtie \widehat{\Sigma}_\ell$*, then* $(\widehat{\Sigma},\kappa,\Theta,T,\rho) \models N'$ *and* $\forall\Sigma_\ell^{i\prime}$ *in* N' *it is* $\Sigma_\ell^{i\prime} \bowtie \widehat{\Sigma}_\ell$.

Checking Trajectories. We now show that by inspecting the results of our CFA, we detect all the possible micro-trajectories of the data produced in the system of nodes that, put together, provide the overall trajectories.

The following corollary shows that we do track the trajectories of IoT data. The first item guarantees that κ and ρ predict all the possible inter-node communications, while the second item shows that our analysis records the micro-trajectory in the T component of each abstract value possibly involved in the communication.

Corollary 4.6. *Let* $N \xrightarrow{\langle\langle v_1,\ldots,v_r\rangle\rangle}_{\ell_1,\ell_2,X} N'$ *denote a reduction in which the message sent by node* ℓ_1 *is received by node* ℓ_2 *with an input tagged* X*. If*

Table 4: Communication CFA rules.

$$\frac{\bigwedge_{i=1}^{k} (\widehat{\Sigma},\Theta) \models_{\ell} M_i^{a_i} \ \wedge \ (\widehat{\Sigma},\kappa,\Theta,T,\rho) \models_{\ell} P \ \wedge \\ \forall \hat{v}_1,\cdots,\hat{v}_r : \bigwedge_{i=1}^{r} \hat{v}_i \in \Theta(\ell)(a_i) \ \Rightarrow \forall \ell' \in L : (\ell,\langle\langle \hat{v}_1,\cdots,\hat{v}_r\rangle\rangle) \in \kappa(\ell')}{(\widehat{\Sigma},\kappa,\Theta,T,\rho) \models_{\ell} \langle\langle M_1^{a1},\cdots,M_r^{ar}\rangle\rangle \triangleright L.P}$$

$$\frac{\bigwedge_{i=1}^{j} (\widehat{\Sigma},\Theta) \models_{\ell} M_i^{a_i} \ \wedge \\ \forall (\ell',\langle\langle \hat{v}_1,\cdots,\hat{v}_r\rangle\rangle) \in \kappa(\ell) : Comp(\ell',\ell) \Rightarrow \\ (\bigwedge_{i=j+1}^{r} \hat{v}_i \in \hat{\Sigma}_{\ell}(x_i) \ \wedge \\ (\ell',\langle\langle \hat{v}_1,\cdots,\hat{v}_r\rangle\rangle) \in \rho(X) \ \wedge \ \forall a \in \mathbf{A}(\hat{v}_i).((\ell,\ell'),X) \in T(a) \\ \wedge (\widehat{\Sigma},\kappa,\Theta,T,\rho) \models_{\ell} P)}{(\widehat{\Sigma},\kappa,\Theta,T,\rho) \models_{\ell} (M_1^{a1},\cdots,M_j^{aj}; x_{j+1}^{a_{j+1}},\cdots,x_r^{ar})^X.P}$$

$(\widehat{\Sigma},\kappa,\Theta,T,\rho) \models N$ *and* $N \xrightarrow{\langle\langle v_1,\ldots,v_r\rangle\rangle}_{\ell_1,\ell_2} N'$ *then it holds:*

- $(\ell_1,\langle\langle \hat{v}_1,\ldots,\hat{v}_r\rangle\rangle) \in \kappa(\ell_2) \wedge (\ell_1,\langle\langle \hat{v}_1,\cdots,\hat{v}_r\rangle\rangle) \in \rho(X)$, *where* $\hat{v}_i = v_{i\downarrow_2}$.
- $((\ell_1,\ell_2),X) \in T(a)$, *for all* $a \in \mathbf{A}(\hat{v}_i)$, *for all* $i \in [j+1,r]$.

Given a term E annotated by a, the over-approximation of its possible trajectories is obtained by computing the trajectory closure of the set composed by all the pairs $((\ell,\ell'),X)$ in $T(a)$.

$$Trajectories(E^a) = Clos_X(T(a))$$

Example 4.7. *Back to our example, we can now determine the possible trajectories of data, e.g. the ones of the term annotated with* a_{1ij}. *By applying the definition of closure* $Clos_X(i^{a_{1ij}})$ *to the entries in* $T(a_{1ij})$, *we can easily obtain that* $Trajectories(i^{a_{ij}})$ *includes* $[((\ell_{bs_{ij}},\ell_{ch_j}),X_{ji}),((\ell_{ch_j},\ell_{as}),Y_j)]$). *This allows us to check which are the nodes the data may pass from, in this case* $N_{\ell_{bs_{ij}}}$ *and* N_{ch_j}, *and which are the corresponding inputs, here* X_{ji} *and* Y_j. *This communication pattern is admittedly simple to illustrate our approach. It is easy to verify that the above CFA results reflect the dynamic behaviour.*

Now, given a classification of the "dangerous" nodes and links, we can analyse the trajectories of each piece of data of the analysed system. We can therefore inspect the paths possibly followed by sensible data and also be suspicious about data produced or passed by unreliable nodes. We can also detect possible illegal or bad flows from one point to another based on security levels. This is particularly crucial in a setting where encryption and other security mechanisms can be costly and power consuming. More in general, our analysis enables traceability of data. For every exchanged message $\langle\langle v_1,\ldots,v_r\rangle\rangle$, the CFA keeps track of the path of each of its components v_i and, in turn, for each v_i it keeps recursively track of the path of the data used to compose it.

5 CONCLUSIONS

We proposed a CFA, based on IoT-LySA, for tracking the propagation of data and for identifying their possible trajectories, as illustrated by a motivating example that offers a simple but non-trivial application of our methodology.

The analysis lends itself for many investigations. On the one hand, it can be used to evaluate the quality of the data managed by the considered system, both in the small and in the large. We can answer questions such as how secure are certain data crucial for critical decisions, or if the provenience of the data processed in a particular node offers sufficient security guarantees. We can also check whether a system respects policies on information flows among nodes.

On the other hand, the collection of possible trajectories of data allows us to discover patterns in general movements of data. We could in fact determine which data move together or in a similar way, thus observing possible emerging patterns. Furthermore, we can find which are the paths or segments of paths that are more used, and therefore may need special attention and suitable security mechanisms.

CFA results on the possible paths followed by data can also be exploited in an early phase of system design, as a supporting technique. Designers can be helped in understanding the potential vulnerabilities related to the presence of dangerous nodes and in determining in time possible modifications and validity checks.

In future, we would like to understand if it is possible to ensure that the nodes continue to behave in a reasonable way even in the presence of not completely reliable data, by linking our approach to that used in (Nielson et al., 2013; Nielson et al., 2015). There, the authors use the Quality Calculus, a process calculus for programming software components with a sort of backup plan in case the ideal behaviour fails due to unreliable communication or data.

Our present analysis would also be integrated with the taint CFA in (Bodei and Galletta, 2017), where

data are marked as tainted when sensitive, and as tamperable when coming from places where they can be tampered.

REFERENCES

Abu-Elkheir, M., Hayajneh, M., and Ali, N. A. (2013). Data management for the Internet of Things: Design primitives and solution. *Sensors*, 13(11):15582–15612.

Bodei, C., Brodo, L., and Focardi, R. (2015). Static evidences for attack reconstruction. In *Programming Languages with Applications to Biology and Security*, LNCS 9465, pages 162–182. Springer.

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

Bodei, C., Degano, P., Ferrari, G.-L., and Galletta, L. (2016a). A step towards checking security in IoT. In *Procs. of ICE 2016*, EPTCS 223, pages 128–142.

Bodei, C., Degano, P., Ferrari, G.-L., and Galletta, L. (2016b). Where do your IoT ingredients come from? In *Procs. of Coordination 2016*, LNCS 9686, pages 35–50. Springer.

Bodei, C., Degano, P., Ferrari, G. L., and Galletta, L. (2017). Tracing where IoT data are collected and aggregated. *Logical Methods in Computer Science*, 13(3).

Bodei, C., Degano, P., Ferrari, G.-L., and Galletta, L. (2018). Sustainable precision agriculture from a process algebraic perspective: A smart vineyard. *Atti Soc. Toscana di Sci. Nat., Memorie Serie B*, 125:39–43.

Bodei, C. and Galletta, L. (2017). Tracking sensitive and untrustworthy data in IoT. In *Procs. of ITASEC 2017*, CEUR 1816, pages 38–52.

Gao, H., Bodei, C., and Degano, P. (2008). A formal analysis of complex type flaw attacks on security protocols. In *Proc. of AMAST'08*, LNCS 5140, pages 167–183. Springer.

Gao, H., Bodei, C., Degano, P., and Nielson, H. (2007). A formal analysis for capturing replay attacks in cryptographic protocols. In *Proc. of ASIAN'07*, LNCS 4846, pages 150–165. Springer.

Herlihy, M. (1991). Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1).

Lanese, I., Bedogni, L., and Felice, M. D. (2013). Internet of Things: a process calculus approach. In *Procs of SAC '13*, pages 1339–1346. ACM.

Lanotte, R. and Merro, M. (2016). A semantic theory of the Internet of Things. In *Procs. of Coordination 2016*, LNCS 9886, pages 157–174. Springer.

Lanotte, R. and Merro, M. (2018). A semantic theory of the Internet of Things. *Inf. Comput.*, 259(1):72–101.

Nielson, H. R., Nielson, F., and Vigo, R. (2013). A calculus for quality. In *Proc. of FACS 2012*, LNCS 7684, pages 188–204. Springer.

Nielson, H. R., Nielson, F., and Vigo, R. (2015). A calculus of quality for robustness against unreliable communication. *J. Log. Algebr. Meth. Program.*, 84(5):611–639.