UNICA

UNIVERSITÀ
DEGLI STUDI
DI CAGLIARI

Ph.D. DEGREE IN

Electronic and Computer Engineering

Cycle XXXV

TITLE OF THE Ph.D. THESIS

Trustworthiness Management in the Internet of Things

Scientific Disciplinary Sector(s)

ING-INF/03

Ph.D. Student:	Claudio Marche
Supervisor:	Prof. Luigi Atzori
Co-tutor:	Ph.D. Michele Nitti

Final exam. Academic Year 2021/2022
Thesis defence: February 2023 Session

"Oh, I just forgot the semicolon..."

Claudio Marche gratefully acknowledges the Sardinian Regional Government for the financial support of her/his PhD scholarship (P.O.R. Sardegna F.S.E. - Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2014-2020 - Axis III Education and training, Thematic goal 10, Investment Priority 10ii), Specific goal 10.5.
Thanks to Michele Nitti, who has always supported the realization of this work.

Contents

List of Acronyms

AUC	Area Under the ROC
BMA	Bad Mouthing Attack
BSA	Ballot Stuffing Attack
C-LOR	Co-Location Object Relationship
C-WOR	Co-Work Object Relationship
DA	Discrimination Attack
IoT	Internet of Things
iSVM	incremental Support Vector Machine
MEE	Micro Engine Entity
ME	Malicious with Everyone
OOA	On-Off Attack
OOR	Ownership Object Relationship
OSA	Opportunistic Service Attack
POR	Parental Object Relationship
QGM	Query Generation Model
QoS	Quality of Service
ROC	Receiver Operating Characteristic
SA	Sybil Attack
SIoT	Social Internet of Things
SN	Social Network

SOR	Social Object Relationship
SPA	Self-Promoting Attack
SVM	Support Vector Machine
SVO	Social Virtual Object
SWIM	Small World In Motion
TMS	Trust Management System
TRM	Trust and Reputation Model
WA	Whitewashing Attack

List of Figures

List of Tables

Abstract

The future Internet of Things (IoT) will be characterized by an increasing number of object-to-object interactions for the implementation of distributed applications running in smart environments. Object cooperation allows us to develop complex applications in which each node contributes one or more services. The Social IoT (SIoT) is one of the possible paradigms that is proposed to make the objects' interactions easier by facilitating the search for services and the management of objects' trustworthiness. In that scenario, in which the information moves from a provider to a requester node in a peer-to-peer network, Trust Management Systems (TMSs) have been developed to prevent the manipulation of data by unauthorized entities and guarantee the detection of malicious behaviour. The cornerstone of any TMS is the ability to generate a coherent evaluation of the information received. The community concentrates effort on designing complex trust techniques to increase their effectiveness; however, strong assumptions still need to be considered. First, nodes could provide the wrong services due to malicious behaviours or malfunctions and insufficient accuracy. Second, the requester nodes usually cannot evaluate the received service perfectly. In this regard, this thesis proposes an exhaustive analysis of the trustworthiness management in the IoT and SIoT. To this, in the beginning, we generate a dataset and propose a query generation model, essential to develop a first trust management model to overcome all the attacks in the literature. So, we implement several trust mechanisms and identify the importance of the overlooked assumptions in different scenarios. Then, to solve the issues, we concentrate on the generation of feedback and propose different metrics to evaluate it based on the presence or not of errors. Finally, we focus on modelling the interaction between the two figures involved in interactions, i.e. the trustor and the trustee, and on proposing guidelines to efficiently design trust management models.

Chapter 1

Introduction

The Internet of Things (IoT) has become a reality with billions of devices able to send key information about the physical world and implementing simple actions, which leads to the paradigm of the anytime and anyplace connectivity for anything [1]. The massive amount of data flowing through the IoT has pushed forward the development of new applications in several domains, such as the management of industrial production plants, the logistics and transport supply chain, the e-health, the smart building, just to cite a few.

However, IoT solutions have posed new challenges in the management of the amount of information produced. Indeed, searching for (reliable) time- and location-relevant information, services and resources for the deployment of running applications exploiting the IoT infrastructure is a crucial challenge: in addition to the size of the searching space, most of the data produced by the sensors produce rapid changes, making the system highly dynamic, as it happens for instance when tracking the position of moving objects. A further complication derives from the shift we are witnessing in the interaction model. From a paradigm where humans look for information provided by objects (human-object interaction), the IoT will surely move towards a model where things look for other things to provide composite services for the benefit of human beings (object-object interaction). With such an interaction model, it will be essential to understand how the information provided by each object can be processed automatically by any other peer in the system. This cannot clearly disregard the level of trustworthiness of the object providing information and services, which should take into account the profile and history of it. If not, attacks and malfunctions would outweigh any of the benefits of these technologies [2].

An approach with the potential to properly address the mentioned issues, which is recently gaining increasing popularity, is based on the exploitation of social networking notions into the IoT, as formalized by the Social Internet of Things (SIoT) concept [3]. It introduces the vision of social relationships among different devices, independently from the fact that they belong to the same or different platforms owned and managed by different individuals or organizations. According to this vision, all the IoT objects are willing to collaborate with others and create relationships among them as humans do. This is expected to make the exchange of information and services among different devices easier and to perform the identification of malicious nodes by creating a society-based view about the trust level of each member of the community. In the resulting social network, each application running in the devices (or in the cloud) will be looking for information and services by crawling the social network starting from a requesting node towards the potential service provider(s). The performance of such a process of service/information retrieval is clearly dependent on several aspects: i) the structure of the social network; ii) the types of service/information requests that will mostly characterize the interaction in the IoT/SIoT; iii) the rules that are used to navigate the network.

However, with such an interaction model, it is essential to understand how the information provided by each object can be processed automatically by any other peer in the system. This cannot clearly disregard the level of trustworthiness of the object providing information and services, which should take into account the profile and history of it. Although we experience and rely on trust during our interactions in everyday life, trust can have many definitions so that it is challenging to define it accurately. The literature on trust is also quite confusing, since it manifests itself in fairly different forms. In this thesis, we adopt the following definition for trust:

Trust is the subjective probability by which an individual, the trustor, expects that another individual, the trustee, performs a given action on which its welfare depends [4].

In the SIoT scenario, the requester has the role of the trustor and has to trust that the provider, which is then the trustee, will provide the required service. However, misbehaving devices may perform several types of attacks for their own gain towards other IoT nodes: they can provide false services or false recommendations, they can act alone or create a group of colluding devices to monopoly a class of services. If not handled adequately, attacks and malfunctions would outweigh any of the benefits of the IoT [2] [5]. For example, in February 2020, Simon Weckert transported 99 smartphones in a handcart and was able to generate virtual traffic jam in Google Maps[1]. In this scenario, trustworthiness management models have to solve the important issue to identify and understand which, among the nodes in the network, are trustworthy and can then lead to successful collaborations.

Literature on Trust Management System (TMS) concentrates on analyzing the different phases involved in the process of managing the trust, its properties, the available techniques used to compose trust, the existing computation models and their effectiveness as defensive mechanisms against malicious attacks. Regardless of the proposed model, all the analyzed papers are tested under two strong assumptions:

- a node provides the wrong service intentionally; however well-behaving devices can show poor performance, due to errors, scarce accuracy or technical problems in general. This problem is usually overlooked by trust algorithm models

[1]http://www.simonweckert.com/googlemapshacks.html

while it is indeed fundamental for them to be able to discern a malicious node from a poor behaving one;

- the requesting node is able to accurately estimate the service received; notwithstanding, requesters do not usually have ground truth information regarding the service so that its evaluation is hardly accurate and even good services, and thus benevolent providers, can be poorly evaluated.

This means that even during a benevolent transaction, i.e. in a transaction involving only benevolent nodes, there could be two possible sources of errors, namely the entity providing the service and the requester evaluating it, without necessarily any malicious node involved. The presence of these errors can confuse the TMS and thus making it difficult to isolate only malicious nodes.

In this regard, this thesis proposes an exhaustive analysis of the trustworthiness management in the IoT and SIoT. To this, in the beginning, we generate a dataset and propose a query generation model, essential to develop a first trust management model to overcome all the attacks in the literature. So, we implement several trust mechanisms and identify the importance of the overlooked assumptions in different scenarios. Then, to solve the issues, we concentrate on the generation of feedback and propose different metrics to evaluate it based on the presence or not of errors. Finally, we focus on modelling the interaction between the two figures involved in interactions, i.e. the trustor and the trustee, and on proposing guidelines to efficiently design trust management models. This thesis then works in this direction and thus provides the following contributions:

- Creation of a synthetic dataset, which includes objects' information and positions from a real scenario, and also the services and applications they offer and use. The collected data derives from the devices installed in the city of Santander in Spain and on the data about people's mobility. This is made available to the research community to test IoT/SIoT management algorithms (e.g., relationship management, service search, trustworthiness management), with particular attention to network navigability.

- Definition of a query generation model, which is able to simulate the correlation between objects and applications and represents a fundamental tool to test the interaction among peers in the network. The proposed model is then used to evaluate the benefits of the social approach in terms of global navigability.

- Design of a decentralized trust management model based on a Machine Learning algorithm, which makes use of novel parameters, namely the goodness, the usefulness and the perseverance score. Thanks to these scores, the model trains and adapts itself, and it is able to identify and react to all possible malicious attacks in the literature.

- Discussion of the deficiencies of the scenarios used to test TMSs and demonstration of how, in such scenarios, advanced techniques are not necessary to deal with the trust management problem.
- Definition of the problem of feedback evaluation in the IoT. Moreover, the proposition of different metrics to evaluate a reference value and the demonstration of how this value should be used to rate services.
- Definition of guidelines to design a suitable trust management model in two different scenarios: an errorless scenario, where cooperative nodes are always able to deliver the requested service, and a realistic environment, where cooperative devices can show poor performance because of errors, poor accuracy, or technical problems in general.

The rest of this thesis is organized as follows: Chapter 2 presents the scenario of Social IoT, a brief survey on trust management models and the possible types of attacks. In Chapter 3, we introduce the scenario, present a dataset based on real objects and design a query generation model. Chapter 4 illustrates the proposed trust management model, while Chapter 5 discusses the deficiencies of the scenarios used to test TMSs and illustrates how a basic model can outperform complex ones with an overly simplified scenario. In Chapter 6, we propose a feedback evaluation model, and in Chapter 7 we propose the guidelines to design a suitable trust management model. Finally, Chapter 8 draws final remarks.

Chapter 2

State of Art

This section provides a brief overview regarding the paradigm of SIoT, then discusses the trust management models and, finally, the trustworthiness attacks.

2.1 The Social Internet of Things

The SIoT represents the convergence of the technologies belonging to two domains: IoT and Social Network (SN). The result is the creation of SNs in which things are nodes that establish social links as humans do [3]. This concept is fast gaining ground thanks to the key benefits deriving from the potentials of the SNs within the IoT domain, such as: simplification in the navigability of a dynamic network of billions of objects [3]; robustness in the management of the trustworthiness of objects when providing information and services [6]; efficiency in the dynamic discovery, selection and composition of services (and of information segments) provided by distributed objects and networks [7]. According to the SIoT model, every node is an object that is capable of establishing social relationships with other things autonomously, according to rules set by the owner.

To this aim, as underlined in [8], there is a strong need to improve the degree of connectivity between users and things, where things should be socialized to allow humans to easily establish relationships with them. The resulting paradigm of SIoT [3] includes these notions, so that people, through their IoT devices, can transparently (although according to clear policies they have set for inter-device interactions) improve the experience in the fruition of smart services and applications.

When it comes to the IoT paradigm, the idea is to exploit social awareness as a means to turn communicating objects into autonomous decision-making entities. The new social dimension shall, somehow, be able to mimic interactions among users and to motivate a drift from an egoistic behavior to altruism or reciprocity. The main principle is to enable objects to autonomously establish social links with each other (by adhering to rules set by their owners) so that "friend" objects exchange data in a distributed manner. Every network object will be capable of: (a) establishing social relationships with other objects autonomously with respect to the owner, but according to the preset rules for the owner; (b) interact with its friends when in need for some assistance, such as the provisioning of a piece of important information or a key service.

According to this model, a set of forms of socialization among objects is foreseen [9]. The Parental Object Relationship (POR) is defined among similar objects, built in the same period by the same manufacturer (the role of the family is played by the production batch). Moreover, objects can establish Co-Location Object Relationship (C-LOR) and Co-Work Object Relationship (C-WOR), like humans do when they share personal (e.g., cohabitation) or public (e.g., work) experiences. A further type of relationship is defined for objects owned by the same user (mobile phones, game consoles, etc.) that is named Ownership Object Relationship (OOR). The last relationship is established when objects come into contact, sporadically or continuously, for reasons purely related to relations among their owners (e.g., devices/sensors belonging to friends); it is named Social Object Relationship (SOR). These relationships are created and updated on the basis of the objects' features (such as type, computational power, mobility capabilities, brand, etc.) and activities (frequency in meeting the other objects, mainly).

However, to fully exploit the benefits of a SIoT network, a trustworthiness management model, able to defend against malicious attacks, is needed, which we investigate in this thesis.

2.2 Trustworthiness Management Models

This section provides a brief overview regarding the background of trustworthiness management in the IoT and SIoT. In the last years, many researchers have tackled this problem, so that the literature is now quite rich. In this Section, we want to show the most appreciated models in the literature and do not intend to cover all the published papers. We classified them into three categories based on the metric used to compute the trust value: metrics obtained from social aspects, metrics based on the Quality of Service (QoS) and mixed approaches, i.e. papers considering both social and QoS aspects.

Among the works considering social aspects, in [10] the authors propose an adaptive decentralized trust mechanism based on social trust. Through a weighted sum, the authors combine factors that concern the cooperativeness and the social communities and demonstrate the effectiveness of the model making use of two real-world social IoT scenarios. Another trust model concerning social trust is presented in [11]. Authors propose a machine learning-based approach to formalize the trust evaluation as a classification problem. The feature vector in a SN is constructed according to social factors like the reputation and the centrality. Another social approach is used in [12]. Throughout a few SIoT trust metrics as centrality, community interest and cooperativeness, the authors illustrate a trust management scheme to facilitate an automatic trustworthy decision making based on the behavior of smart objects. Two social scenarios are described in [13] and [14]. In the first work, the authors take into account metrics such as social similarity and the importance of the service. The resulting trust management algorithm is developed using social relationships to compute the trust level of the nodes in a SIoT network. In the second one, the authors propose a centralized trust-based protocol for mobile objects. To guarantee the trust accuracy between the devices the system makes use of friendships and social contacts.

Concepts of QoS are used for example in [15]: authors present a remote attestation mechanism for the sensing layer node in the IoT. A real-time trust measurement is realized through a combination of QoS factors, such as transmission delay, historical data and feedback originated from other objects. Firstly, a node verifies the identities of the other nodes and only then measures whether the computing environment is trustworthy. In [16], the authors compute the trust scores based on the exchange of feedback, which are provided taking into account QoS factors such as the monetary cost of the resources, the computation capabilities and the communication failures. Two other QoS approaches based on centralized architectures are described in [17] and [18]. In the first work, the authors propose a policy-based secure scheme for IoT, in which the trustworthiness of data and devices are evaluated according to the reporting history and the context in which the data are collected. In the second one, the centralized architecture is used for information sharing among health IoT devices. The proposed trust protocol considers the loss of probability of health data and the reliability of the IoT devices. Another approach concerning QoS factors is presented in [19]. Authors introduce an approach to evaluate the trust of services combining several QoS attributes (such as availability and response time) and user's ratings. The model focuses on satisfying the users' choices on web services and it is evaluated considering the influence of malicious rating.

The last group of papers makes use of both QoS and social trust metrics to compose the trust value. Among them, in [6], the authors propose a decentralized trust mechanism in a social scenario. In that model, each node computes the trustworthiness of the service providers on the basis of its own experience and on the opinion of its friends. The authors analyze the QoS factors, as computation capabilities, and social factors, such as centrality and credibility. QoS and social metrics are both considered also by Chen et al. in [20]. They adopt a distributed scheme where each node maintains its own trust assessments. The QoS factors (i.e., quality reputation and energy status) are related to the social relationships and recommendations from the other nodes. Two other mixed approaches are described in [21] and [22]. In [21], authors propose a trust evaluation model incorporating heterogeneous information from direct observation, personal experiences and global reputation. The subjective algorithm makes use of social factors, e.g. cooperativeness and community-interest, and of QoS factors, aggregated with a weighted sum mechanism and a machine learning to change the weights according to the particular context. In [22], the authors illustrate an IoT protocol that uses trust for the evaluation of nodes to make optimal routing decisions. It computes the trust of nodes by examining QoS factors, such as the number of exchanged packets, and the recommendations from the neighbors. A recent model is described in [23]. Authors propose guidelines for the design of a decentralized trust management model, which can be used for assisting humans and devices in the decision making process.

All the analyzed models are designed and tested to isolate nodes that implement a subset of the possible types of attacks. However, the heterogeneity of IoT scenarios call for models with no weak points, while existing works show a common limitation: they are not able to properly identify all the type of malicious attacks. The next subsection shows all the possible malicious behaviours that can be implemented in a network.

2.2.1 Trustworthiness Attacks

Two different behaviors can be considered in a network [24]: one is always benevolent and cooperative, while the other one is a strategic behavior corresponding to an opportunistic participant who cheats whenever it is advantageous for it to do so. The goal of a node performing maliciously is usually to provide low quality or false services in order to save its own resources; at the same time, it aims to maintain a high value of trust toward the rest of the network so that other nodes will be agreeable to provide their services when requested. This strategy, even if successful for a single node at first sight, involves a huge risk for the network because trusting the information from malicious devices could lead to serious compromises within the network and this has a direct impact on the applications that can be delivered to users [25]. A trust model has to identify this behaviour to discourage nodes from implementing it; however, such malicious nodes can perform several types of trust-related attacks, which represent the different solutions they adopt to avoid being detected. We classify trustworthiness attacks based on two dimensions: the first dimension is related to the **target** of the attack, i.e. if the malicious node aims to confuse the network by providing false services, false recommendations or both. The second dimension is connected to the **size** of the attack, i.e. if the trustworthiness attack is carried on by a single node or by a group. In the following, we briefly describe the different types of attacks known in the literature.

The largest group of attacks is composed of single nodes that indiscriminately provide both bad services and recommendations. In this group, trustworthiness attacks differ based on the mechanism they adopt in order not to be recognized:

Malicious with Everyone (ME): a malicious node acts maliciously with everyone. This is the most basic attack: a node always provides bad services and recommendations, regardless of the requester [6].

Discrimination Attack (DA): a malicious node modifies its behavior based on the service requester. This means that a node can discriminate non-friends nodes or nodes with weak social ties. As a result, some devices can consider the node as benevolent while others can label it as malevolent [26].

		Size	
		Single	*Group*
Target	*Service*	[OSA]	
	Service	[ME] [DA] [OOA]	
	Recommendation	[WA] [SPA]	[SA] [BSA]
	Recommendation	[BMA]	

Table 2.1: Classification of different types of trustworthiness attacks.

On-Off Attack (OOA): a node periodically changes its behavior, by alternatively being benevolent (ON) and malevolent (OFF). During the ON state, the node builds up its trust, which is then used to attack the network [27].

Whitewashing Attack (WA): a node with a bad reputation leaves the network and then registers again with a different identity. When the node re-join the network its reputation is reset to a default value [28].

Self-Promoting Attack (SPA): a malicious node provides good recommendations for itself in order to be selected as a service provider. After it is selected as a provider, it provides only bad services [29].

The other types of attacks concentrate on a single target, i.e. malicious nodes only provide bad services or bad recommendations.

Bad Mouthing Attack (BMA): this attack is addressed to ruin the reputation of other nodes; a malicious node only provides false recommendations to decrease the chance of benevolent nodes being selected as providers. Usually, this attack is part of a collusive behavior where a group of nodes works together to ruin the reputation of a good node but it can also be carried on by a single node [30].

Ballot Stuffing Attack (BSA): this is a type of collusive attack, where a malicious node provides good recommendations toward another malicious node to boost its reputation and increase its chances to be selected as the provider [31].

Sybil Attack (SA): a malicious node uses multiple identities to provide different types of recommendations on the same service. These multiple identities are usually fake and they are all responsible for the attack process [32].

Opportunistic Service Attack (OSA): a malicious node provides good services only when it senses that its trust reputation is dropping. In this way, the node tries to maintain an acceptable level of trust in order to still be selected as a service provider [33].

To sum it up, Table 2.1 shows a classification of trust-related attacks based on the two dimensions identified, while Table 2.2 compares the analyzed models with the attacks they are able to identify. To the best of our knowledge, all available trustworthiness models are able to isolate only a subset of the presented attacks, i.e. they are designed to recognize and isolate some specific attacks, but none of them is able to defend efficiently against all the attacks. Table 2.2 does not show the ME attack, which is used as a reference attack by all the models, and the OSA attack since a node performing it can not be completely isolated but it is only possible to

Reference	**DA**	**OOA**	**BSA**	**WA**	**BMA**	**SA**	**SPA**
[10]	✓	-	✓	✓	✓	-	✓
[11]	-	-	-	-	-	-	-
[12]	-	✓	-	-	-	-	-
[13]	-	-	✓	-	✓	-	✓
[14]	✓	-	✓	-	✓	-	✓
[15]	-	-	-	✓	-	✓	✓
[16]	-	-	-	-	-	-	
[17]	-	✓	✓	-	✓	-	-
[18]	-	-	✓	-	✓	-	-
[19]	-	-	-	-	✓	-	-
[6]	✓	✓	-	-	-	-	✓
[20]	✓	✓	-	-	✓	-	✓
[21]	-	-	✓	-	✓	-	✓
[22]	-	-	-	-	-	✓	-
[23]	-	-	✓	-	✓	-	-

Table 2.2: Resiliency of existing models against identified trust related attacks.

reduce the number of times a node acts maliciously due to the reliability needed to build up the trust.

These attacks span from simple ones, which have a constant behaviour over time, such as ME, to more complex ones which are able to change their behaviour over time: among them, for example, there is the On-Off Attack, the Discriminatory Attack or the Opportunistic Service Attack, which have all been tested in this thesis. In particular, the OSA is considered the most complex attack in the literature since it knows exactly how the trust model implemented in a system works, so it is able to accurately predict how its trust value will change based on its behaviour and then behaves accordingly.

Chapter 3

SIoT Dataset of and Query Generation Model

This chapter aims to provide the two essential elements needed to develop and test management algorithms in an IoT or SIoT ecosystem, namely a query generation model and a dataset of objects with realistic behavior. As it is depicted in Figure 3.1, even though these two elements can cooperate, i.e. the query generation model can be tested by using the dataset, they exist and can be used independently. The details about their functionalities will be better explained in Section 3.1 (for the query generation model) and Section 3.2 (for the dataset).

The SIoT provides the objects with some capabilities typical of humans' behavior when looking for and providing information in their social communities. Accordingly, social relationships are created among objects, which are used when the peers are looking for help [3]. As in most of the IoT architectures, the owner has the control on which social interactions the objects are allowed to perform and which information and services can be shared with other peers.

The applications installed by owners in their cloud space and that rely on their objects' capabilities often need to look for services provided by other objects. This results in queries that are managed by the SIoT by making use of objects' social connections through word of mouth.

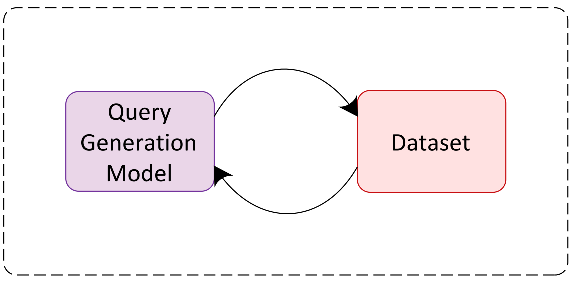

Figure 3.1: Proposed structure to develop and test IoT management algorithms

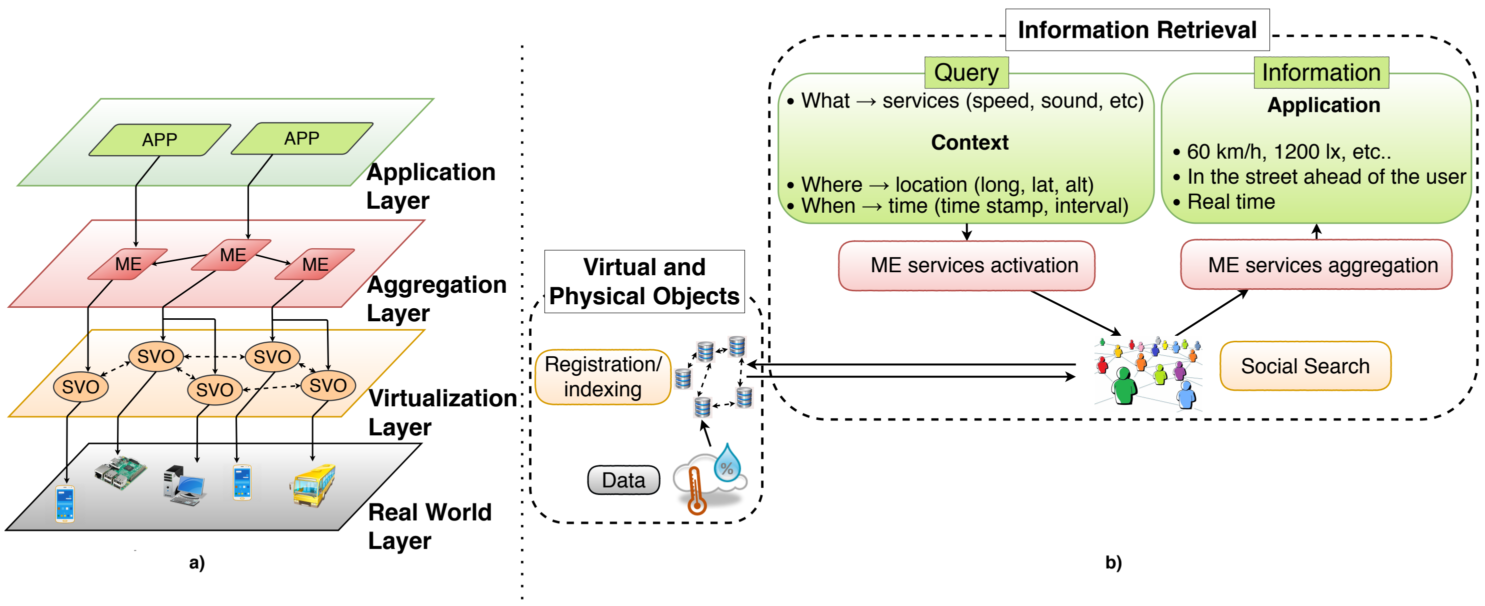

Figure 3.2: Reference SIoT architecture and query generation model.

The focus of this chapter is the analysis and modeling of this query generation process. To this, we consider the reference SIoT architecture shown in Figure 3.2a), which is based on four levels [34]: Application, Aggregation, Virtualization and Real World. The lower layer is made up of the "things" of the real world, which have the role to sense the physical environment and provide data to the higher layers. The Virtualization layer is made of Social Virtual Object (SVO)s, which represent the digital counterparts of any entity of the real world enhanced with social capabilities, fully describing their characteristics and the services they are able to provide [35]. The Micro Engine Entity (MEE), which is the main entity of the Aggregation layer (represented as ME in the Figure), is a mash-up of one or more SVOs and other MEEs, and it is responsible for getting and processing information from SVOs into high-level services requested by applications at the higher level. Finally, the Application layer is installed in the Cloud and partially in the devices, so that applications can be deployed and executed exploiting one or more MEEs.

Figure 3.2b) illustrates a generic service query in the SIoT, which highlights all the components involved in the process. The whole process starts when the application layer triggers some processing that requires to look for other services and then generates a relevant query. The query specifies *what* services are required and it is enhanced with context parameters, which represent the application requirements, such as a specific time (*when*) or a specific place (*where*).

The generated query is then handled at the Aggregation Layer, where the needed MEEs for data elaboration are activated. After this, the query is taken over by the

SVO of the device that triggered the process, which navigates its social network in order to search for other SVOs that can offer the data related to the desired services by the application. Indeed, in the SIoT, each SVO maintains information related to its friends and to the services that the corresponding physical object can provide. In this sense, SVOs can be seen as atomic registration/indexing servers. However, it is not the focus of the thesis to design an indexing mechanism of data.

In order to better explain the query process, an explanatory example is presented here. Suppose that an object installed the application *RealTimeTrafficEvaluation-App*, which evaluates the traffic of a specific street in real-time, i.e. within a limited time interval with respect to the current time. Accordingly, the object creates a query with the list of services needed to execute the application and the requirements related, in this case, with the reference location and time. The aggregation layer than activates the MEEs associated with the services and passes the query to the SVO, which looks for the objects, among its social network, that can execute the services fulfilling the desired requirements. Once they are found, the aggregation layer processes the result and provides the requested information, i.e. the real-time traffic condition in the specified street, to the user.

When all the services are retrieved, they are forwarded again to the Aggregation Layer which composes them through the activated MEEs and finally provides the result of the application back to the device that triggered the request.

The depicted scenario where objects collaborate by mashing their services has great potentials as this allows for the deployment of powerful applications. This is the case of objects (e.g. cars) that share information to decide on the best route to get to a destination, objects that perform collaborative spectrum sensing and objects that need to send alarms to all the people in a given area to reach humans nearby, just to cite few examples. Reaching the right device(s) with whom interact is a key task in this context, and the SIoT provides a potentially effective approach to this by relying on the created social network. However, to evaluate the relevant performance, there is the need to model the generation of the query characterizing these scenarios, which should help in conducting a proper system performance evaluation. Such a model should describe which object (with relevant characteristics) would typically need to retrieve information from any other objects with other relevant features. Whereas the query model that is proposed in the following is adopted to evaluate the performance of the SIoT paradigm, it can be adopted for other IoT architectures as well. Finally, since our goal is to model the objects' behavior when requesting services at the application layer, in this chapter we do not consider how the query is handled.

In our modeling, the set of nodes in the SIoT, i.e., the set of SVOs, is represented by $\mathcal{N} = \{n_1, ..., n_i, ...n_I\}$ with cardinality I, where n_i represents a generic SVO. Its physical counterpart can be static or mobile with position $\boldsymbol{L}_i = \left[l_i^a, l_i^b\right]$, which can then be fixed or varying over time. In our problem setting, SVOs create social relations so let the resulting SN be described by an undirected graph $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$, where $\mathcal{E} \subseteq \{\mathcal{N} \times \mathcal{N}\}$ is the set of edges, each representing a social relation between a couple of nodes.

The modeling of our problem can not overlook the different typologies of objects in a network, since objects with different profiles can provide different services and are interested in different applications [36]. We then define the following sets: $\mathcal{T} = \{t_1, ..., t_x, ...t_X\}$ as the set of possible typologies of objects, such as smartphones, cars, traffic lights, and others. For every typology t_x, we define a set $\mathcal{B}_x = \mathcal{B}(t_x) = \{b_{x1}, ..., b_{xy}, ...b_{xY}\}$ as the set of possible brands inside the typology t_x, while the set $\mathcal{M}_{xy} = \mathcal{M}(b_{xy}) = \{m_{xy1}, ..., m_{xyz}, ...m_{xyZ}\}$ represents the set of possible models for typology t_x and brand b_{xy}. All the possible models available in the network can then be described by the set $\bar{\mathcal{M}} = \{\cup_{\forall xy}\mathcal{M}_{xy}\}$, which allows us to define the following 2-tuple $\Gamma = \langle \mathcal{N}, \bar{\mathcal{M}} \rangle$, which associates to every node n_i the corresponding model of the device and thus enables also to infer the typology and the brand. This tuple will be useful to enable the creation of the parental relation (i.e. the POR defined in Section 2.1), which is based on these characteristics.

Then, we need to define the applications in the network, which are those that are requested during the querying process and the possible services provided by the nodes and that can satisfy the queries. Let $\mathcal{A} = \{a_1, ..., a_w, ...a_W\}$ be the set of possible applications that can be installed by the devices in our scenario. However, applications do not run on all the devices but only on those they are meant to, so a single device will only have a subset of applications installed on it; we can then define the matrix $\boldsymbol{O} = [o_{iw}]$ where the generic element o_{iw} is equal to 1 if node n_i can potentially install application a_w and 0 otherwise.

Then, we define $\mathcal{S} = \{s_1, ..., s_j, ...s_J\}$ as the set of services that can be performed by any node in the network and that can be used to compose the applications in $\mathcal{A}$. Thus, we can define the matrix $\boldsymbol{D} = [d_{ij}]$, where the generic element d_{ij} is equal to 1 if node n_i can provide service s_j and 0 otherwise.

It is true that both the matrices of installed applications and available services, namely $\boldsymbol{O}$ and $\boldsymbol{D}$, should be related to the typology of the node, since it is the typology that determines the possible uses for an object. However, this approach is too simplistic since different nodes can offer different services and run different applications based on external characteristics related to their owner, such as privacy settings. Let us consider two users which own a smartphone each: one of them is willing to share all the smartphone's services while the other one only one or two of them; similarly, even if the set of applications they can install on their smartphone is the same, they have decided to install different applications based on their interests.

To model how an application generates a query, let's recall that a query only specifies the needed services and their requirements, and it is the aggregation layer that combines them to fulfill the request of the application. To this, we can model the query as the tuple $\mathcal{Q} = \langle \mathcal{Q}^{serv}, \mathcal{Q}^{req} \rangle$, where $\mathcal{Q}^{serv} = \{q_1^{serv}, ..., q_h^{serv}, ...q_H^{serv}\}$ is the set of atomic queries representing the individual services needed to fulfill the application requests using a node's social network, while $\mathcal{Q}^{req} = \{q_1^{req}, ..., q_k^{req}, ...q_K^{req}\}$ is the set of requirements. The goal of a query generation model is then to calculate the probability to generate a specific query $\mathcal{Q}$. In our modeling, we make the assumption

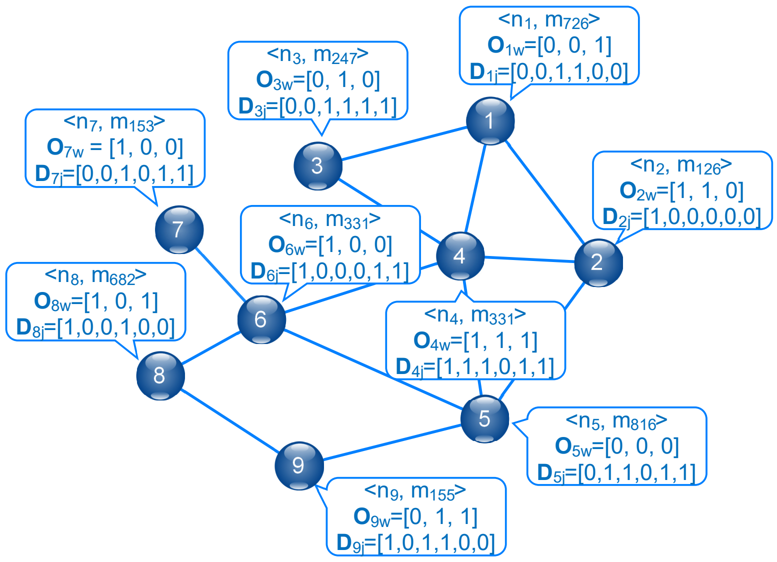
$\langle n_1, m_{726} \rangle$ $\mathbf{O}_{1w}$=[0, 0, 1] $\mathbf{D}_{1j}$=[0,0,1,1,0,0]
$\langle n_3, m_{247} \rangle$ $\mathbf{O}_{3w}$=[0, 1, 0] $\mathbf{D}_{3j}$=[0,0,1,1,1,1]
$\langle n_7, m_{153} \rangle$ $\mathbf{O}_{7w}$ = [1, 0, 0] $\mathbf{D}_{7j}$=[0,0,1,0,1,1]
$\langle n_2, m_{126} \rangle$ $\mathbf{O}_{2w}$=[1, 1, 0] $\mathbf{D}_{2j}$=[1,0,0,0,0,0]
$\langle n_6, m_{331} \rangle$ $\mathbf{O}_{6w}$=[1, 0, 0] $\mathbf{D}_{6j}$=[1,0,0,0,1,1]
$\langle n_8, m_{682} \rangle$ $\mathbf{O}_{8w}$=[1, 0, 1] $\mathbf{D}_{8j}$=[1,0,0,1,0,0]
$\langle n_4, m_{331} \rangle$ $\mathbf{O}_{4w}$=[1, 1, 1] $\mathbf{D}_{4j}$=[1,1,1,0,1,1]
$\langle n_5, m_{816} \rangle$ $\mathbf{O}_{5w}$=[0, 0, 0] $\mathbf{D}_{5j}$=[0,1,1,0,1,1]
$\langle n_9, m_{155} \rangle$ $\mathbf{O}_{9w}$=[0, 1, 1] $\mathbf{D}_{9j}$=[1,0,1,1,0,0]
1
2
3
4
5
6
7
8
9

Figure 3.3: Representation of the network nodes.

that the number of atomic queries matches the number of services to be found; nonetheless, based on the particular search mechanism implemented in the SIoT, the number of queries can be lower w.r.t. the number of services, since a query can be used to find two or more services at the same time. However, the modeling of the search engine is not considered in this thesis.

Figure 3.3 provides a simple example of a generic SIoT graph $\mathcal{G}$, where $I = 9$ and each node is characterized by a tuple $\Gamma_i = \langle n_i, m_{xyz} \rangle$, which defines for node n_i its model m_{xyz}, from which we can infer the typology t_x and the brand b_{xy}. In our example, we can notice how nodes can share the same typology, as it is the case of nodes n_2, n_7, n_9, since they have the same first digit of the $\bar{\mathcal{M}}$ set, and even the same brand, as n_7 and n_9 are described by m_{153} and m_{155} respectively. In particular, if nodes belong to the same typology, brand and model, such as the case of nodes n_6 and n_4, they are then able to create a POR.

In this example, each SVO can have up to 3 applications installed, as indicated by the number of columns of matrix $\boldsymbol{O}$, and it is capable of providing up to six services, as shown by the column dimension of $\boldsymbol{D}$. Suppose that a user, which owns node n_1, is interested in the *DriveMonitoringApp* application that monitors and evaluates his/her driving behavior and the related risks and then installs it in n_1. To provide the requested application, which is indicated in our example as o_{13}, to the user, the related SVO will have to search for the needed services, which are shown by the orange balloons in Figure 3.4 and that are indicated as the services s_1, s_2, s_3, s_5 and s_6 in our example scenario. Node n_1 will then generate a query $\mathfrak{q}$ with $\mathcal{Q}^{serv} = \{q_1^{serv}, q_2^{serv}, q_3^{serv}, q_4^{serv}, q_5^{serv}\}$ and $\mathcal{Q}^{req} = \emptyset$ to look for the five services among its friends. When all the services are retrieved, they are sent

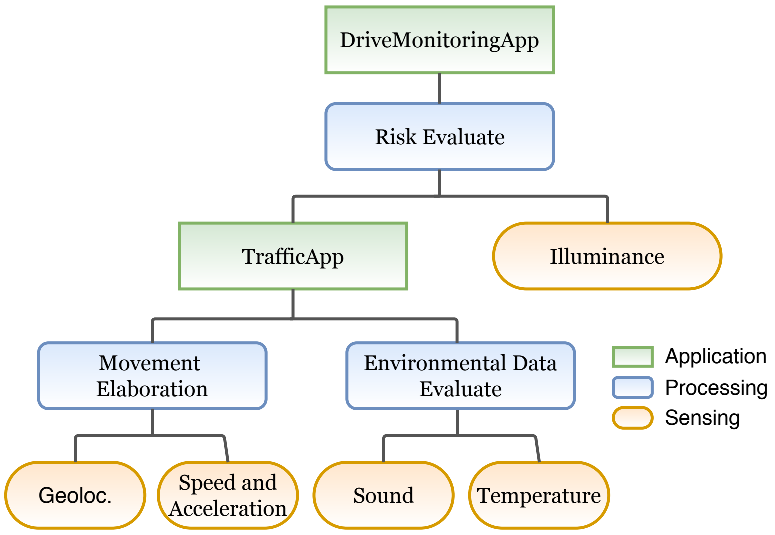

Figure 3.4: Decomposition of the *DriveMonitoringApp* application into services.

to the aggregation layer, which provides the necessary processing capabilities (blue balloons in Figure 3.4). Please note that in some cases the node could be able to provide some of the services by itself as the case of node n_1, which can provide service s_3 ($d_{13} = 1$).

As we will see in the next section, the query generation model is more complicated than this example, since it has to take into account space and time requirements.

3.1 Query Generation Model

In the IoT, the number of possible applications is huge, but not all the types of things can install the same set of applications and even the same application installed in the same object can generate queries with different requirements.

When studying the IoT, and in particular the Social IoT, it is difficult to evaluate the performance of service search mechanisms, i.e. how IoT/SIoT systems can fulfill application requests. This is due to the lack of query generation models, that are needed to understand which application can generate a query and with which requirements. As described previously, the goal of a query generation model is to compute the probability that a query $\mathcal{Q}$ will be generated; the composition of the atomic queries in $\mathcal{Q}^{serv}$ represents the set of services needed by the application. The choice of the application that will generate a query, and that will then determine the services to search, depends on the particular object in which the application is installed. Figure 3.5 shows the main functionalities that characterize the query generation model. According to this picture, based on the chosen application and on which node it is installed, the model has to generate the set of query requirements

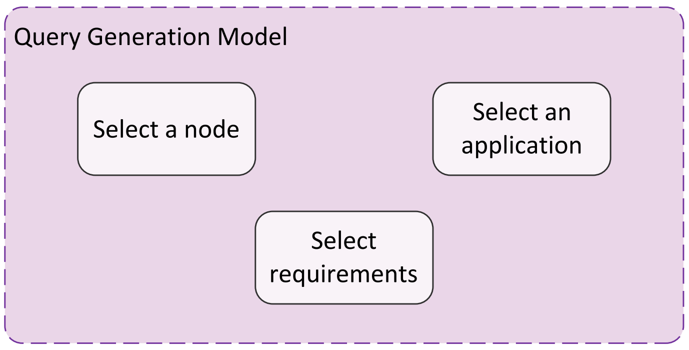

Figure 3.5: Query generation model functionalities

$\mathcal{Q}^{req}$, which are applied to the set of atomic queries.

Applications and nodes are highly intertwined: choosing a node determines which applications can be installed on that node, and selecting an application fixes the possible nodes in which the application can be installed. In order to obtain the probability to generate a specific set of atomic queries $\mathcal{Q}^{serv}$, that corresponds to application $\mathcal{A} = a_w$, we have to compute the joint density function of nodes $\mathcal{N}$ and application $\mathcal{A}$ as follows:

$$p_{\mathcal{A},\mathcal{N}}(a_w, n_i) = p(\mathcal{A} = a_w \cap \mathcal{N} = n_i) = \begin{cases} 0 & \text{if } o_{iw} = 0 \\ p_i(\mathcal{Q}^{serv}) & \text{if } o_{iw} = 1 \end{cases} \tag{3.1}$$

where $p_i(\mathcal{Q}^{serv})$ is the probability that node n_i, which potentially installed application a_w, generates the set of atomic queries $\mathcal{Q}^{serv}$. For $o_{iw} = 1$, it can also be written in terms of conditional distributions:

$$\begin{aligned} p_i(\mathcal{Q}^{serv}) &= p(\mathcal{N} = n_i | \mathcal{A} = a_w) * p(\mathcal{A} = a_w) = \\ &= p(\mathcal{A} = a_w | \mathcal{N} = n_i) * p(\mathcal{N} = n_i) \end{aligned} \tag{3.2}$$

Eq. 3.2 shows the double nature of the query generation process, which can begin both by selecting an application or a node.

The probability that the set of atomic queries $\mathcal{Q}^{serv}$ is generated by any node in $\mathcal{N}$ is then defined as

$$P(\mathcal{Q}^{serv}) = \sum_i p_i(\mathcal{Q}^{serv}) \tag{3.3}$$

The application' selection greatly influences the difficulty of the search operations, since applications can have different levels of intricacy, ranging from simple ones, which only need one or two services, to complex ones, with nested applications and multiple services. Moreover, not all applications require information with the same frequency. To this, in Section 3.2.3, we will test several different distributions for the applications' frequency, namely $p(\mathcal{A} = a_w)$, to evaluate how the SIoT network reacts in terms of navigability.

The choice of the node affects both its geographical and social position. The first one is important since it influences the requirements of the query, while the position of the node in the social network impacts on the number of friends selectable and thus on the number of friends a node can rely upon when looking for services. Since there is no particular constraint in the choice of a node, i.e., every node has the same probability to trigger an application request, $p(\mathcal{N} = n_i)$ follows a uniform distribution.

Once the query for services has been generated, it is important to know which requirements are needed for the specific application, namely to generate the set of requirements for the query. Indeed, different nodes requesting the same application can also specify different attributes or characteristics for it. The set of possible requirements can be quite large, ranging from the accuracy of the sensed data to their precision; however, not all the requirements are always needed: the *only* ones that need to be declared, either explicitly or implicitly, are space and time. For example, an application that needs temperature measurements as inputs could be requested in different areas, such as in a room or a park (space requirement) and for different time intervals, as it is the case for historical or real-time data (time requirement). The minimum set of requirements can then be expressed as $\mathcal{Q}^{req} = \{q_{s1}^{req}, q_{s2}^{req}, q_t^{req}\}$, where q_{s1}^{req} and q_{s2}^{req} indicate the space requirements, namely for the x and y-coordinates, while q_t^{req} expresses the time requirement.

As suggested in [37], to describe the concept of interest in a specific point in space, the best distribution should be normal: to this, we describe the space requirements as a 2-dimensional normal distribution, where the probability density function can be expressed as follows:

$$f_{wi}(q_{s1}^{req}, q_{s2}^{req}) = \frac{1}{2\pi\sigma_{q_{s1}^{req}}\sigma_{q_{s2}^{req}}} * exp\left(-\frac{1}{2}\left[\frac{\left(q_{s1}^{req} - l_i^a - \mu_{q_{s1}^{req}}\right)^2}{\sigma^2_{q_{s1}^{req}}} + \right.\right.$$
$$\left.\left. + \frac{\left(q_{s2}^{req} - l_i^b - \mu_{q_{s2}^{req}}\right)^2}{\sigma^2_{q_{s2}^{req}}}\right]\right) \tag{3.4}$$

where $\mu_{q_{s1}^{req}}$, $\sigma^2_{q_{s1}^{req}}$ and $\mu_{q_{s2}^{req}}$, $\sigma^2_{q_{s2}^{req}}$ are the mean and variance values for the x and y-coordinates respectively.

All these values are application dependent, i.e. they depend on the particular application a_w at hand, but we have decided not to show such dependence in the above formula to keep it clean. In particular, when the mean values, $\mu_{q_{s1}^{req}}$ and $\mu_{q_{s2}^{req}}$, are both equal to 0, then the distribution is centered on the current position of the node n_i, namely l_i^a and l_i^b, i.e a node is looking for information around itself.

Also, the time requirement can be modeled using the time interest of applications, as suggested in [38], since objects require information mostly in real-time and less as we move farther in time, i.e. historical data. We modeled such behavior as an exponential distribution as follows:

$$f(q_t^{req}) = \begin{cases} 0 & \text{if } q_t^{req} > 0 \\ \lambda_a * exp\left(\lambda_a q_t^{req}\right) & \text{if } q_t^{req} \leq 0 \end{cases} \tag{3.5}$$

where λ_a is a constant, depending on the particular application at hand. The requirement for $q_t^{req} = 0$ means that the application is needed in real-time, while the values of $q_t^{req} < 0$ indicate that historical data are requested. Whenever a SVO receives a request with a temporal requirement, it will check if its stored data can satisfy the requirements, otherwise, it has to contact the physical objects to retrieve the data; however, in some cases, the SVO would not contact its physical counterpart, in order to avoid consuming resources.

Once the query has been generated, the goal of the SIoT system will be to find all the services in $\mathcal{Q}$ starting from the SVO of the node with the selected application, making use of its social relations to crawl the network.

As an example of query generation, let us consider the following flow: the system chooses randomly an object among the available ones, e.g. a car. This car can be interested in several applications, so the model has to pick one of them, based on how frequently they require information, e.g. the *DriveMonitoringApp* showed in Figure 3.4: the resulting set of atomic queries is then $\mathcal{Q}^{serv} = \{$ Geoloc., Speed and Acceleration, Sound, Temperature, Street Lights $\}$. The final step is to set the requirements for the application, that will be inherited by every service in $\mathcal{Q}^{serv}$: as spatial requirement, the car chooses an area of [100 m x2 km] around itself (i.e. in the road ahead), while as time requirement, the car selects $q_t^{req} = 0$ thus asking for the information to be obtained continuously in real-time.

The goal of the SIoT system will then be to find all the services in $\mathcal{Q}^{serv}$ starting from the SVO of the selected car, making use of its social relations to crawl the network.

3.1.1 Query Model Validation

In order to simulate and validate the query generation model proposed, a set of real IoT queries is required. These data are obtained by the Lysis platform [39]: a collection of more than 11000 queries from 154 devices over a period of 7 months, from April 2017 to October 2017 (a complete description of the data is available here[1]). The network is composed of two types of nodes: smartphones and Raspberry boards; based on the typology, the devices can require up to five different applications.

[1]http://www.social-iot.org/index.php?p=downloads

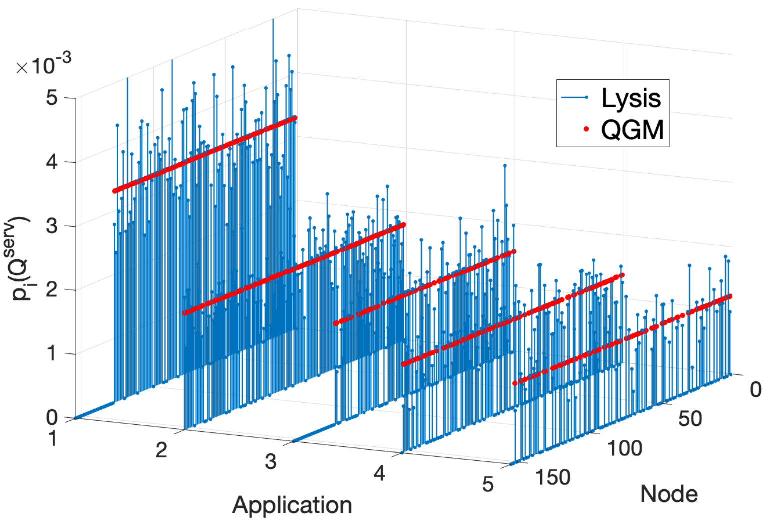

Figure 3.6: Query probability distribution for the Lysis data and for the Query Generation Model (QGM).

Figure 3.6 illustrates how the proposed query generation model, displayed by red dots and labeled Query Generation Model (QGM), matches the probability for each node to generate a specific set of atomic queries $\mathcal{Q}^{serv}$ obtained from the Lysis dataset, represented with blue lines. Moreover, since not all the devices can install the same applications, then it will happen that some nodes will *never* require a given service and then *never* generate the corresponding query and thus $p_i(\mathcal{Q}^{serv}) = 0$. In our model, the nodes follow a uniform distribution, while the applications' frequency is proportional to the number of services needed by each application, i.e. that the first app requires more services than the last app. To evaluate the performance of our model, we made use of an f-divergence measure, namely the Hellinger distance [40], to quantify the similarity of the two probability distributions. Unlike other f-divergence measures, the Hellinger distance is a bounded metric: given two probability distribution P and Q, the maximum distance 1 is achieved if P and Q are completely divergent, while a distance $H(P, Q) = 0$ means that the two probability functions are completely overlapping and hence identical. In our case, the value of the Hellinger distance is equal to 0.0047, so we can conclude that our model is able to generate an almost identical distribution w.r.t. the real data.

Table 3.1 shows the parameters used to describe the space and time requirements for each of the five applications. The two columns labeled as H_1 and H_2 indicates the Hellinger distance between the real data from the Lysis dataset and our requirement distributions for space and time, respectively. The maximum value of the Hellinger distance is under 0.16 thus indicating a very good approximation of our model.

The values of the model's parameters, namely $\mu_{q_{s1}^{req}}$, $\mu_{q_{s2}^{req}}$, $\sigma_{q_{s1}^{req}}$, $\sigma_{q_{s2}^{req}}$ and λ_a, are computed by applying linear regression to a small set of interactions for each

App	$\mu_{q_{s1}^{req}}$	$\mu_{q_{s2}^{req}}$	$\sigma_{q_{s1}^{req}}$	$\sigma_{q_{s2}^{req}}$	H$_1$	λ_a	H$_2$
1	0	0	0	0	0	0.50	0.099
2	0	0	0.1	0.1	0.019	2.00	0.116
3	0	0	0.3	0.3	0.013	0.50	0.108
4	0.45	-0.45	0.1	0.1	0.012	10.0	0.118
5	0.70	-0.70	0.1	0.1	0.018	20.0	0.159

Table 3.1: Requirements for the Lysis Applications.

application (around 1% of the total number of requests).

3.2 Dataset and Network Analysis

This Section presents a dataset of profiles for the objects in a Smart City environment, based on the FIWARE Data Models [41]. The dataset is then used to construct a social network of objects, which is analyzed in Subsection 3.2.2. Finally, the last subsection assesses the performance of the network when tested with the query generation model in terms of navigability.

3.2.1 Dataset

The main functionalities required to create a dataset are illustrated in Figure 3.7. As it will be better explained in the rest of the subsection, these functionalities are in charge of creating: objects' information (e.g. owner, typology, brand, model), traces of the positions and timestamps of the devices, the list of all the applications that can be installed by the objects, objects' profiles (expressed as the set of the available services) and an adjacency matrix with the social relationships for each object.

The first step to construct a dataset is to obtain the profile of the objects. To this, we extracted objects' information and positions from the SmartSantander project [42], which is experimental research in support of typical applications and services for a smart city. We have classified all the available devices according to the data models proposed in the FIWARE Data Models. This enables the portability of the dataset among different platforms. These models consider both static and mobile objects and are mostly located in the city center of the city.

Each of the three public mobile categories of objects, namely buses, taxis and garbage trucks, moves in an independent way: buses' movement is created according to the list of bus stops, which are available from the Servicio Municipal de Tranportes Urbanos de Santander (TUS) [43]; taxis can start from 1 out of 3 taxi stations around the city; garbage trucks start from the landfill and cover all the city.

However, a complete Smart City scenario must also consider devices from private users. To this, we introduce 4000 users in the city, so that each user owns a certain

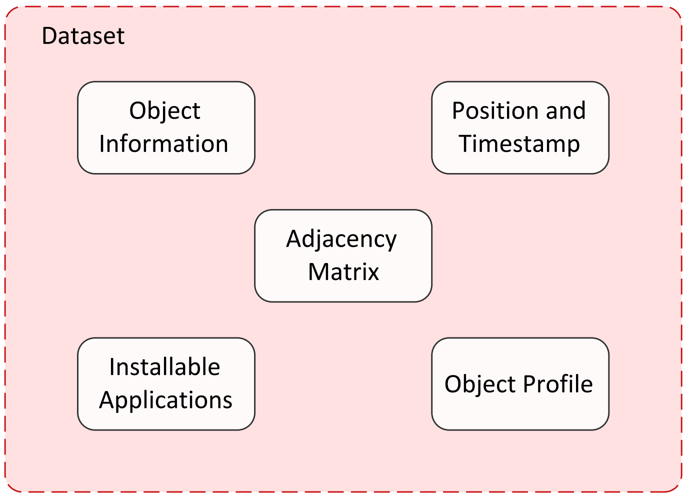

Figure 3.7: Dataset functionalities

Mobile Devices	**Ownership (%)**
Smartphone	91
Car	55
Tablet	40
Smart Fitness	22
Smartwatch	5
Static Devices	**Ownership (%)**
Pc	84
Printer	53
Home sensors	15

Table 3.2: Distribution Ownership Devices over 50.000 Users Aged 16-64.

number of devices. The devices' distribution is based on the ownership report of the Global Web Index in 2017 [44] calculated over 50000 users aged among 16 and 64 years old and it is shown in Table 3.2; some of these devices are considered mobile, i.e. they are carried by the users during their movements, while others are static and are then left at the user's home.

To simulate the mobility of these 4000 users, we rely on the well-known mobility model Small World In Motion (SWIM) [45]. SWIM can generate synthetic data, which can create mobility traces able to mimic human social behaviors. In fact, it has been proven that the SWIM mobility model allows obtaining accurate matching between the output of the model and the most popular mobility traces available in CRAWDAD [46], generating data with the same statistical properties, such as in terms of inter-contact time between people. The simulation area needs to match the city center of Santander, so since SWIM only considers areas of interest of unitary square, we had to scale down the city center (which roughly has an area of 4 km x 4 km) and then modify the model to avoid users to move towards uninhabited areas, such as the sea.

The simulator requires some additional parameters. The user perception radius, set to 0.015, indicates the distance within which a user, or in our case a device, can *see* all other users/devices; this parameter is set according to the communication range of a Wi-Fi connection [47] specifically scaled considering that the simulation area of SWIM is a unitary square. The parameter α, which can have values in the range $[0;1]$, is used to determine whether the users prefer to visit popular sites (smaller values) rather than nearby ones (bigger values). Accordingly to the city scenario, it has been set to 0.9. The entire simulation covers a time-lapse of ten days.

Following the proposed network modelling, the dataset as a total number of devices equals $I = 16216$, 14600 of which are private and 1616 are public. The resulting network comprehends a total of $X = 16$ typologies of objects and to each of the typologies owned by private users, a brand and a model selected randomly among $Y = 12$ brands and $Z = 24$ models have been assigned. We suppose that the municipality bought all the objects inside an object's typology with the same brand and model, so only the category is needed to classify public objects.

The devices of the smart cities, compared to the dataset in [48], are able to provide $J = 18$ services, which can be arranged to provide $W = 28$ different applications for the users.

A complete description of the data obtained in this thesis is available for tests here[2] and includes objects' information (such as owner or typology), traces on the positions and timestamps of the devices, the list of all the applications we envision in a Smart City scenario, objects' profiles (expressed as a set of available services and possible applications requests) and an adjacency matrix with the social relationships for each object.

3.2.2 Network Analysis

Based on object movements and profiles, each device can create its own set of relations with other devices. All the relations depend on the rules set in the system: as explained in [49], these rules have a direct impact on the overall navigability of the network: for the overall network to be navigable, i.e. to enable a node to easily reach any other node in the network, all, or the most of, the nodes must be connected, i.e., a giant component must exist in the network, and the effective diameter must be low. Moreover, the distribution of the number of connections each node has with its peers, namely the degree distribution, should be close to a power-law distribution. This results in a scale-free network and indicates the presence of hubs, i.e. nodes with a large number of connections w.r.t. the average, in the network. With this goal in mind, in the following, we discuss the characteristics of the obtained resulting

[2]http://www.social-iot.org/index.php?p=downloads

network. The only relation we did not consider in these experiments is the C-WOR since it has been demonstrated from [49] that its contribution to the navigability of the network can be negligible.

All relationships, except for the OOR, are created using as a starting point [48]. An overview of the relations and their differences is illustrated below:

- The *Ownership Object Relationship* (OOR) is created between devices that belong to the same owner. To avoid too many relations, objects will create a relation only if they are in the communication range of each other. We assume that private devices use one out of three possible technologies: LoRa, Wi-Fi and Bluetooth.

- The *Parental Object Relationship* (POR) is created among two objects in the same category, brand and model. Since the reasoning behind the POR is to create long-distance links, two devices owned by private users, with the same typology, brand and model, will establish a relationship only if their distance is greater than a threshold, which is set to 3.8 km in order to reduce the number of relationships. For the public devices, a node is elected as a hub and all the other nodes with the same model will create a POR with the hub.

- Devices located in the same place can create a *Co-Location Object Relationship* C-LOR. These relationships are created between a static device and a mobile one and do not take into account the contact duration but only the number of meetings between the two objects. A number of meetings equal to 10 has given an appropriate number of relations.

- The *Social Object Relationship* SOR is a relation type that can be created among mobile devices and it is based on three parameters, namely the number of meetings (N), the meeting duration (T_M) and the interval between two consecutive meetings (T_I). These parameters are set to $N = 3$, $T_M = 15$ minutes and $T_I = 3$ hours, respectively.

- Mobile public object have hardly any chance to create SORs, so in order to include them in the SIoT network, we introduce another specific type of SOR. This SOR, which we called SOR$_2$, uses the same parameter of the SOR but with less stringent constraints; in particular, we set them to $N = 2$, $T_M = 2$ minutes and $T_I = 1$ hour.

The resulting distribution for the network is shown in Figure 3.8, which considers the matchup of the versions for the different relations. The graph illustrates how the network is able to create a giant component with all the nodes since its degree distribution is close to a power law distribution. While Table 3.3 shows the main network parameters for each relation and the whole network. We can notice that the SIoT degree distribution can be approximated to a power-law distribution, thus

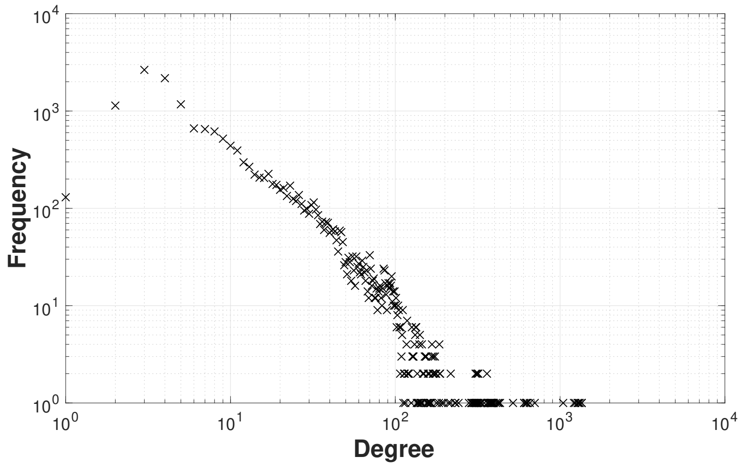

Figure 3.8: SIoT Degree Distribution.

Parameters	OOR	POR	C-LOR	SOR	SOR_2	SIoT
Number of relationships	58173	21245	27440	21245	20910	146117
Giant component (%)	8.99	4.17	51.28	24.06	15.23	100
Average degree	50.01	2.00	6.59	10.89	16.93	18.02
Average path length	2.15	1.99	27.31	4.34	3.01	4.22
Diameter	5	2	69	8	7	8

Table 3.3: Relationships' parameters.

indicating its navigability. This is due to the presence of C-LOR and SOR, while the OOR and POR, which originate from other parameters, i.e., nodes characteristics and number of devices owned by a user, deviate from such a distribution: however, these relationships are still important since they connect groups of nodes so that the majority of nodes have more than one connection.

The average degree indicates the average number of edges connected to each node: OOR is the relation that creates the greatest number of friendships, however, it only creates small clusters of highly interconnected objects and thus the dimension of the giant component, the highest percentage of nodes belonging to the largest finite fraction of the entire graph's nodes, is low. Similar reasoning also applies to the POR: since the goal of the POR is to create long-distance links, the relation is created only if the distance between two devices is greater than a threshold so that the resulting number of relations is lower w.r.t. OOR. Finally, OOR and POR are able to create a highly connected cluster has can be inferred by the low values of the average path length, which is the average number of steps along the shortest paths for all possible pairs of network nodes. In order to connect the public mobile devices (buses, taxies and garbage trucks) we had to add another type of relation, that we called SOR_2: this relation makes use of the same parameter of the SOR but considering less stringent requirements. The contribution of the SOR_2 to the navigability of the network is the same as C-LOR and SOR: since all these relations create short distance links among devices regardless of their characteristics, they are able to connect the cluster of objects created by OORs and PORs. The resulting SIoT network then comprehends a giant component with all the devices where the longest shortest path between any two nodes, i.e. the diameter of the network, is still low.

3.2.3 SIoT network navigability

The navigability in a network indicates how a node can reach any other peer and thus represents a fundamental parameter both for the generation of a network and to understand the average distance between cooperating nodes. To test the navigability of our dataset and query generation model, we have chosen the object typology with the highest number of requested applications, the smartphones, and analyze 1000 processes of the query generation model. All the results are shown with a 95% confidence interval around the mean value, i.e. that 95% of the values from the distribution lie within ± 1.96 standard deviations. The network's response is calculated in terms of average distance, i.e. the average number of hops needed to find all the required services, computed on the number of services of the application. This is done to avoid disparity among applications that require a different number of services: if, for example, the application depicted in Figure 3.4 is satisfied in three hops, that means that the five services composing it are found by the search engine in 15 hops, and then with an average of three hops each. This is also justified by the fact that the services can be found in parallel; however, in this thesis we do not implement any specific searching mechanism, i.e. we are not using any mechanism for a node to navigate the network on its own with local information. On the other hand, we compute the distance among two nodes in terms of global network navigability, i.e., routing is performed by assuming that each object has a view about the global social network topology.

In order to compare the performance of the query generation model for the SIoT, we also created two other networks with similar characteristics: a Random network and a Barabási-Albert network, which is able to generate scale-free networks based on preferential attachments [50]. The characteristics of the three networks are shown in Table 3.4: we can see how, at a global level, the SIoT has a higher average path length and diameter w.r.t the other two networks.

The first set of simulations focuses on the impact of the applications' frequency. Queries are then generated with a frequency related to the number of services needed by the application and without any kind of requirements, i.e. the network has to

Parameters	**Random**	**BA**	**SIoT**
Number of relationships	145852	146449	146117
Average degree	17.99	18.06	18.02
Average path length	3.68	3.17	4.22
Diameter	5	5	8

Table 3.4: Characteristics of the Random, Barabási-Albert and SIoT networks.

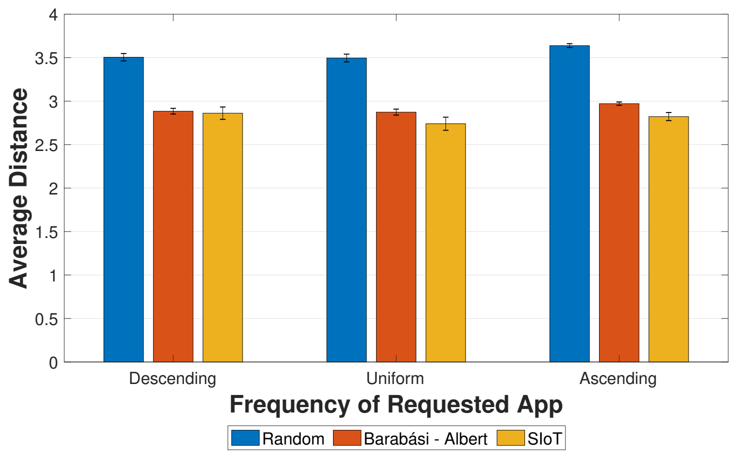

Figure 3.9: Average number of hops needed to solve a query. The applications' frequency changes based on the number of services requested by the application itself.

find all the nodes that can provide the required service. The results are shown in Figure 3.9.

We can notice that the SIoT network is able to outperform the other two networks, independently of the frequency. This is due to the fact that the SIoT relations are created to connect nodes with similar interests, so as to facilitate the discovery of information. Moreover, the impact of the applications' frequency is negligible. This result can be explained considering that the final goal of a search engine is to find the services needed by an application and that the same services can be arranged in several ways thus providing different applications: this means that even by changing the frequency, the services that need to be found are mostly the same. In the following, we will consider that all applications generate queries with the same frequency.

The second set of experiments consists of the analysis related to the space requirement: we first evaluate the impact of the mean values for the x and y-coordinates and then we investigate the effects of the variance.

Figure 3.10 shows the hop distance when nodes request applications that are located, on average, 0.5, 1.5 and 2.5 km away from them. This value is calculated

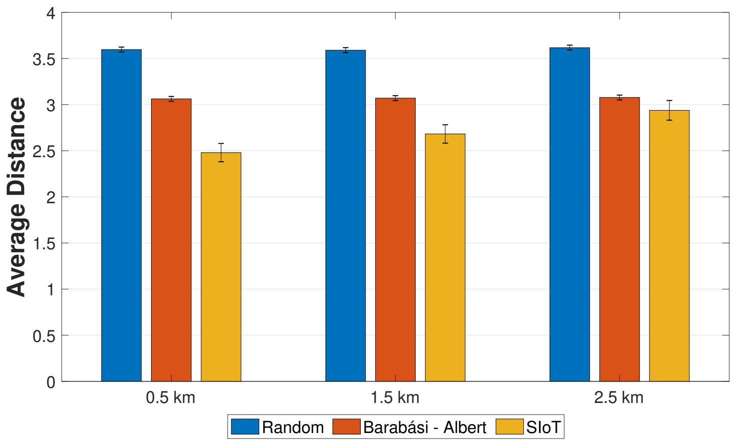

Figure 3.10: Average distance for different mean values of the space requirement.

as the Euclidean distance between the requester and the possible providers. We can note that there is a difference of almost half a hop between the two extreme cases, namely 0.5 and 2.5 km; even if the SIoT envisages the creation of long-distance links, such as the PORs, the greatest number of relations are created with nearby devices, so the best results can be obtained when a node looks for services in its vicinity. This is justified by Figure 3.11, which shows the average number of friends created within 1, 2 and 3 km from a node. We can note that, w.r.t. the other two networks, in the SIoT the greatest number of relations are created with nearby devices, so the best results can be obtained when a node looks for services in its vicinity.

To test the impact of the variance, shown in Figure 3.12, we consider different values that can cover respectively 50, 100 and 500 meters. As expected, the bigger the variance, the bigger the number of nodes that can provide the requested services and thus is simpler for the search engine to quickly find them.

The third set of simulations focuses on the time requirements, i.e. how fresh the information a node is requesting must be. As explained before, the search mechanism is performed at the virtual level, where the virtual counterparts store the information provided by the physical objects: however, this information can not be always synchronized with the ones sensed by the objects due to energy and bandwidth constraints. Based on the characteristics of the objects, every typology has a different synchronization time (see Table 3.5), so it may happen that the information found by the search engine is not fresh enough. In this case, the SVO interacts with the physical object, and then consumes its resources, to ask for additional reading in order to satisfy the query. Synchronization times are chosen as prime numbers to avoid that a large number of objects upload information to their corresponding SVOs at the same time. At the start of our simulation (time 0), all the devices synchronize their data with the corresponding SVO, and then they follow the syn-

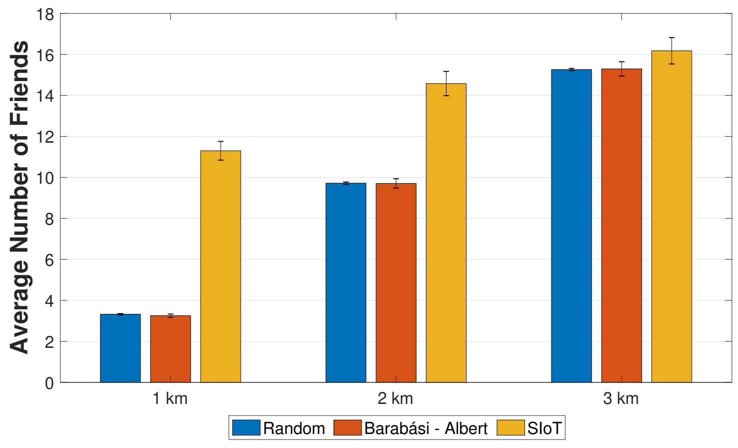

Figure 3.11: Average number of friends for objects within an area of 1, 2 and 3 km radius.

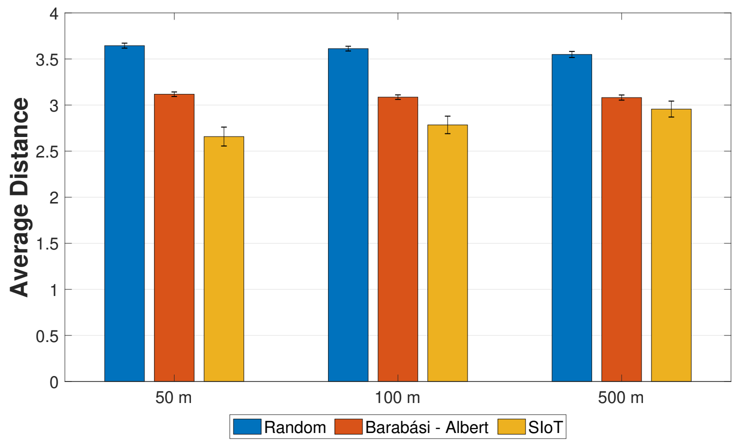

Figure 3.12: Average distance for different variance values of the space requirement.

Typologies	**Sync. Time**
Car, Indicator, Smart Fitness, Street lighting and Waste Management	7 minutes
Alarms, Home sensors, Parking, Smartphone, Smartwatch, Tablet and Transportation	17 minutes
Environment, Pc and Weather	23 minutes
Point of Interest and Printer	29 minutes

Table 3.5: Synchronization time for all types of objects

chronizations depicted in Table 3.5. So at any point in time, when a request with a temporal constraint arrives, we are able to compute if the SVO can satisfy it with its information or it has to contact the physical device.

The following results are shown only for the SIoT network; indeed, the network is created considering only space parameters, so there are no further differences among the networks.

Figure 3.13 shows the average distance to satisfy a query looking for information generated within 1, 5 or 10 minutes. We run 100 query processes and each process is repeated 10 times. Relaxing the time constraint, as it happens with a 10 minutes requirement, leads to results similar to those obtained without any requirement for the query; on the other hand, the number of hops increases when the application is requested within a short time. As we approach the real-time requirement, $q_t^{req} = 0$, we can note that some points start to be missing from the curves: in particular, when requesting applications with a 1-minute requirement, the corresponding curve has no data for processes 12, 74 and 77. This means that the search engine has not been able to find any SVO satisfying the query in any of the 10 runs.

We then decided to analyze the number of times an SVO has to contact its physical counterpart during the 10 runs to satisfy the query. Figure 3.14 shows the corresponding results: as expected, with 1-minute requirement and during processes 12, 74 and 77, the SVO had to contact the physical object for all 10 runs and 6.43 times on average over the 100 processes, while with the 10-minutes requirement it is always possible to find an SVO with the required service. Finally, with the 5-minutes requirement, the physical objects are contacted on average less than once for each process (0.77 times).

The last set of simulations concerns the performance of the network to satisfy a complete query with both space and time requirements. To this, we decide to create a generic query requesting an application 1.5 kilometer away from the requester (mean value) with a range of 200 meters (variance value) and with information related to the last 5 minutes (time value). Figure 3.15 on the left axis shows the distance to solve the query, which is 3.16 hops over the 100 processes.

Finally, on the right axis of Figure 3.15, we try to analyze if hubs are involved in

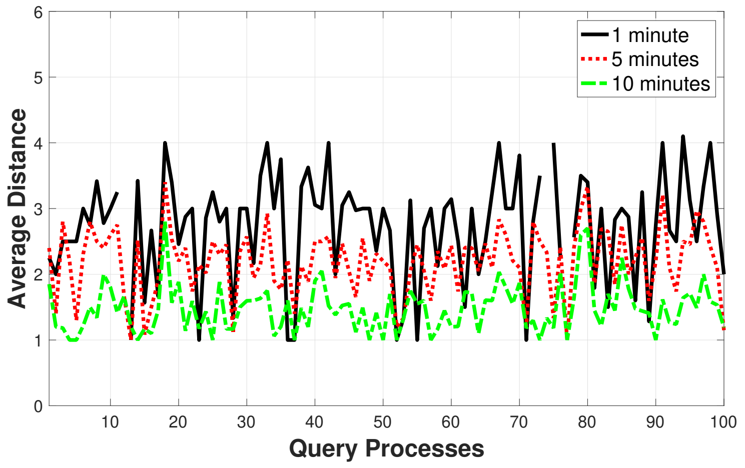

Figure 3.13: Average distance for different values of the time requirement.

the search process by plotting the average degree of the intermediate nodes between requester and provider, i.e. the degree of the nodes forwarding the query. Given that the global network navigability returns the best possible path, it is also able to find the best intermediate nodes, i.e. the hubs in the network. By analyzing the degree of such nodes, we provide hints to the development of local routing algorithms, that should make use of these nodes. The average degree for the requesters is 17.43 friends, which is in line with the average degree of the network, while the average degree for the providers is 50.25 friends. However, when studying the degree of the intermediate nodes we find that its value is 149.08 connections thus confirming that the hubs are a crucial part of the search mechanisms in the SIoT.

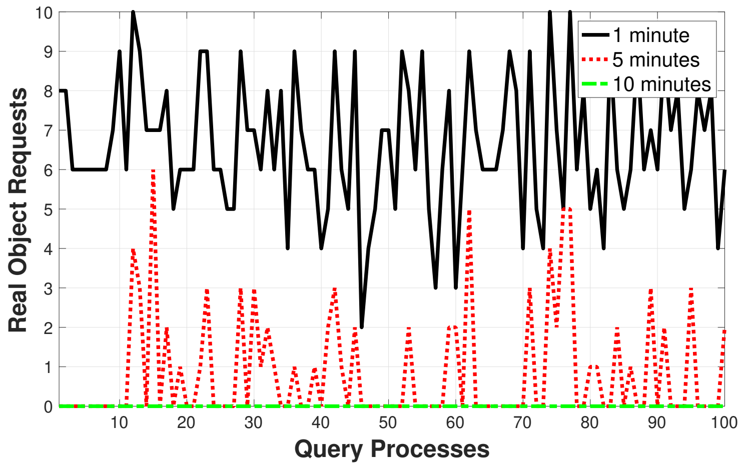

Figure 3.14: Number of request to real world objects due to the time requirement.

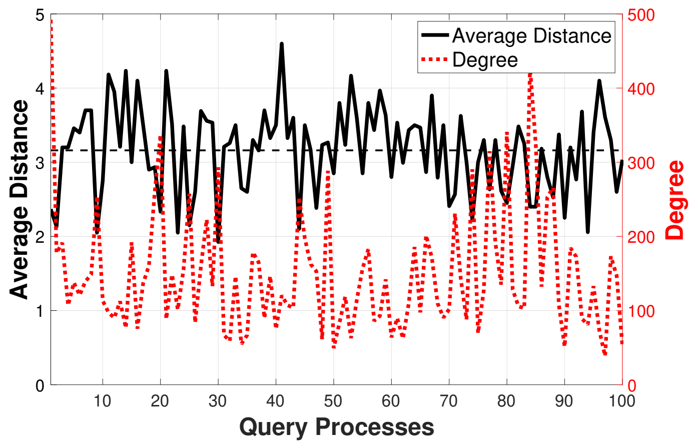

Figure 3.15: Average Distance and average of intra-nodes degree for queries with both requirements, *Space* and *Time*.

Chapter 4

Trust Management Model

The focus of this chapter is to propose a first trust management model able to identify all the trust attacks analyzed in Section 2.2.1 and isolate the nodes performing them. According to the scenario presented in Chapter 3, in our modeling, the set of nodes in the SIoT is represented by $\mathcal{N} = \{n_1, ..., n_i, ...n_I\}$ with cardinality I, where n_i is the generic node. The resulting SN, created by the devices' relationships, can be described by an undirected graph $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$, where $\mathcal{E} \subseteq \{\mathcal{N} \times \mathcal{N}\}$ is the set of edges, each representing a social relation between a couple of nodes. The friends of the generic node n_i are represented in our model by $\mathcal{N}_i = \{n_j \in \mathcal{N} : n_i, n_j \in \mathcal{E}\}$, that is the set of nodes that share a relation with it; moreover, we define $\mathcal{H}_{ij} = \{n_h \in \mathcal{N} : n_h \in \mathcal{N}_i \cap \mathcal{N}_j\}$ as the set of common friends between n_i and n_j.

Every node in our network can provide one or more services, so that $\mathcal{S}_j$ is the set of service that can be provided by n_j. The reference scenario is then represented by a node n_i requesting a particular service S_h: a Service Discovery component in the network is able to return to n_i a list of potential providers $\mathcal{P}_h = \{n_j \in \mathcal{N} : S_h \in \mathcal{S}_j\}$. At this point, the requester has to select one of the providers in $\mathcal{P}_h$ based on their level of trust. The trust level is usually computed based on the previous interactions among the nodes. Indeed, after every transaction l, the requester n_i assigns feedback to the selected provider n_j to evaluate the service: we can then define the set of feedback $\mathcal{F}_{ij} = \left\{f_{ij}^1, ..., f_{ij}^l, ...f_{ij}^{L_{ij}}\right\}$, where l indexes from the latest transactions ($l = 1$) to the oldest one ($l = L_{ij}$), so that L_{ij} represents the total number of transactions between the two nodes. Each feedback can be expressed using values in the continuous range $[0, 1]$, where 1 is used when the requester is fully satisfied by the service and 0 otherwise.

Figure 4.1 provides a simple example of a generic graph $\mathcal{N} = \{n_1, ..., n_9\}$, with each node capable of providing one or more services, as highlighted in the grey clouds; n_1 is the node that is requesting the service S_7, as highlighted in the white cloud; $\mathcal{P}_h = \{n_5, n_6\}$ is the set of nodes that can provide the requested service. In this figure, we also highlight the set $\mathcal{N}_1 = \{n_2, n_3, n_4\}$ of nodes that are friends

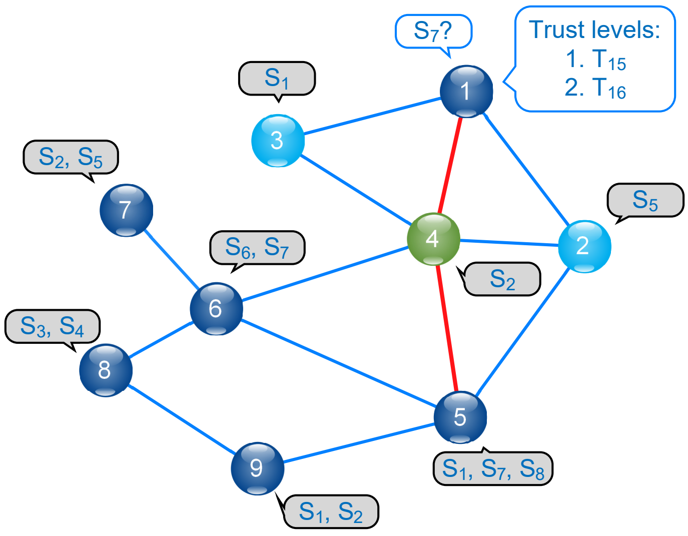

Figure 4.1: Trust Management Model.

of n_1 (in light blue color). Within note that the set $\mathcal{H}_{15} = \{n_2, n_4\}$ and the set $\mathcal{H}_{16} = \{n_4\}$ of nodes represent the common friends between n_1 and n_5 and between n_1 and n_6, respectively. For each of the provider in $\mathcal{P}_h$, the requester n_1 computes the trustworthiness levels, T_{15} and T_{16}, and then chooses the provider with the highest value, which is n_5 in our example.

The goal of any trustworthiness management model is to compute and list the trust level of all the providers. This step is fundamental to help the requester to identify the most reliable node to whom require the service and to avoid any malicious node. In our model, we envision that each node n_i computes the trustworthiness level T_{ij} of all the possible providers n_j on its own, so that different nodes can make different choices when selecting a provider based on their past experiences.

4.1 Trust Model Design

According to the presented scenario, we propose a decentralized scenario, where each node calculates and stores information about the other nodes, so to have its own opinion about the network: in this way, malicious attacks that change their behaviour based on the requester, such as DA, are easily identified. Whenever a node n_i has to evaluate the trustworthiness of another node n_j, it computes the trust value as follows:

$$T_{ij} = \alpha^{L_{ij}} C_j + \beta^{L_{ij}} R_{ij} + \gamma^{L_{ij}} O_{ij} + \delta^{L_{ij}} S_{ij} \tag{4.1}$$

All these addends are in the range $[0, 1]$ and the weights are selected based on the total number of transactions L_{ij} between node n_i and n_j. Moreover, the weights' sum, namely $\alpha^{L_{ij}} + \beta^{L_{ij}} + \gamma^{L_{ij}} + \delta^{L_{ij}}$, is always equal to 1, in order to normalize the trust value in the interval $[0, 1]$, and their relative value can be changed to give more impact to a particular parameter.

A generic node n_i evaluates the trustworthiness T_{ij} based on four parameters: the *Computation Capabilities* C_j of the service provider, the *Relationship Factor* R_{ij} between the two nodes, the *External Opinions* O_{ij} provided by n_i's friends and the *Dynamic Knowledge* S_{ij} acquired by the requester. The Dynamic Knowledge represents the core of our system, which has to learn how to identify malicious nodes. This ability is tied to its experience, i.e. to the past transactions of the node. Accordingly, the proposed trustworthiness model is divided into two phases: a training phase and a steady-state phase. In the training phase, the contribution of the Dynamic Knowledge is limited, because the requester is trying to learn the behavior of the provider: since the requester has to understand the behavior of each node it interacts with, the four weights are dependent by both the requester and the provider; we omit this dependency to avoid too much confusion in the presented equations. In particular, the value of $\delta^{L_{ij}}$ grows with the total number of transactions L_{ij} between the requester n_i and the provider n_j, as follows:

$$\delta^{L_{ij}} = \begin{cases} (L_{ij} - 1)/L^{tr} & \text{for } L_{ij} \leq L^{tr} \\ 1 & \text{for } L_{ij} > L^{tr} \end{cases} \tag{4.2}$$

where L^{tr} represents the number of transactions needed to train the Dynamic Knowledge. The residual weight, i.e. $1 - \delta^{L_{ij}}$, is then shared among the other weights.

4.1.1 Training Phase

The goal of this phase is to let the Dynamic Knowledge factor collects enough experience. Until this happens, the trust value of the potential providers is calculated based on the elements described below.

The **Computation Capabilities** C_j is a static characteristic of an object which does not vary over time. This factor accounts for the heterogeneity of the IoT where some devices are more powerful than others so their ability to act maliciously is higher and they can lead to more uncertain transactions. To take into account this possibility, the model assigns lower values to objects with great computational capabilities w.r.t. devices with only sensing and actuation capabilities.

The **Relationship Factor** R_{ij} is a unique characteristic of the SIoT and it is related to the relationships that ties node n_i and n_j. Using [6] as a starting point, we set the greatest value for the OOR relationship and decreasing values for the other relations. If two nodes are tied by two or more relationships, e.g. they have created both an OOR and a SOR, we consider the strongest relation which then they have with the highest value. If two nodes have no direct relation, the model computes the sequence of social links between them and consider the weakest link in the path, i.e. the minimum value of all the relationship factor. To account for the uncertainty of the intermediates nodes, this value is further divided for the number of hops that separate node n_i and node n_j.

The **External Opinion** O_{ij} evaluates the recommendations provided to n_i by the friends in common with n_j, namely the nodes in $\mathcal{H}_{ij}$ and is expressed as:

$$O_{ij} = \sum_{h=1}^{|\mathcal{H}_{ij}|} T_{ih} \cdot T_{hj} \Bigg/ \sum_{h=1}^{|\mathcal{H}_{ij}|} T_{ih} \tag{4.3}$$

where T_{hj} represents the opinion, i.e. the trust value, that each of the common friends n_h has for node n_j. These values are weighted with the trust values that node n_i has already computed towards its friends, so that the opinion of trustworthy nodes is considered more than the one from low trustworthy nodes. Indeed, recommendations represent an effective strategy, adopted by many trust algorithms, to easily obtain information regarding other nodes. This is especially true when a node's direct experience is still scarce. However, they are also exploited by many trustworthiness attacks, such as BMA and BSA, to confuse the network: using the external opinion only in the training phase, our model is resilient to all these types of attacks.

Moreover, at the end of each transaction, n_i assigns a feedback not only to the provider but also to the friends in $\mathcal{H}_{ij}$, which have contributed to the computation of the external opinion. According to Eq. 4.4, if a node provided a positive opinion, it receives the same feedback as the provider, i.e. a positive feedback if the transaction was satisfactory, $f_{ij}^l \geq 0.5$, and a negative one otherwise, $f_{ij}^l < 0.5$. Instead, if n_h gave a negative opinion, then it receives a negative feedback if the transaction was satisfactory and a positive one otherwise.

$$f_{ih}^l = \begin{cases} f_{ij}^l & \text{if } T_{hj} \geq 0.5 \\ 1 - f_{ij}^l & \text{if } T_{hj} < 0.5 \end{cases} \tag{4.4}$$

Moreover, to further reduce the possibility of attacks on the recommendations, in our algorithm, a node uses them only in the training phase to accumulate experience and then it only relies on its Dynamic Knowledge.

4.1.2 Steady-State phase

After the training phase, only the Dynamic Knowledge is used to evaluate the possible providers. According to the presented scenario, certain types of malicious nodes, e.g. OOA and OSA, continuously change their behaviour. In order to address this issue, the Dynamic Knowledge must be able to continuously learn and adapt to the myriad of possible malicious behaviours. To compute its value, we make use of an incremental Support Vector Machine (SVM), so that a node can constantly extends its knowledge after a new transaction: in particular, a SVM is a supervised learning model that analyzes a set of data, in our case the first L^{tr} transactions, to provide some sort of classification. SVM algorithms have been applied to solve a variety of applications [51]. With respect to other machine learning algorithms, the risk of over-fitting is less, it is relatively memory efficient and is effective when there is a margin of separation between classes. The accuracy of this classification is tied to the number of historical data obtained [52]: in our case, the output of the SVM represents the probability that a service provider is benevolent or not, i.e. its trust value. More details regarding the validation process of the iSVM and a comparison with other incremental machine learning algorithms will be presented in Section 4.2.2.

After every transaction, the Dynamic Knowledge is updated, so that it is able to learn from its past experience and can provide a more accurate evaluation. Since we make use of an incremental SVM, with each new transaction the model's knowledge is extended and updated, without the need to train the SVM from scratch. This way, each node can implement a dynamic Machine Learning algorithm even with limited resources and active learning is much faster w.r.t. a traditional approach. In order to train the SVM, past transactions are expressed in terms of scores, which have the goal to highlight different aspects of the interaction among nodes. Three scores are used as inputs for the Dynamic Knowledge, which are able to evaluate the entire history of the nodes as well as their recent behavior. In this way, the attacks with a dynamic behaviour over time, such as OOA and OSA, can be recognized. The first score is the **Goodness Score**: this score enables the SVM to evaluate nodes on a long-term period and measures how benevolent the node has been during all its transactions. The score is evaluated as the fraction of all the "good" transactions, i.e. all the transactions evaluated in a positive way by the requester:

$$G_{ij} = \frac{|\{f_{ij}^l \in \mathcal{F} : f_{ij}^l > TH\}|}{L_{ij}} \tag{4.5}$$

where TH is the threshold a requester set to consider services as "good". High values of this score mean that the service requester is overall satisfied by the services obtained from the provider. This factor is also useful to identify benevolent nodes which provide services with low accuracy that a requester would like to avoid and that are then labeled with a low value of the Goodness Score.

However, the Goodness Score is not able to react to sudden changes in the behavior of a node, as it happens for dynamic attacks such as OOA and OSA. To overcome these attacks, we make use of two other scores, which evaluate the behavior of the service provider considering a small temporal window, which makes use of the last L^s transactions.

The **Usefulness Score** is used to evaluate only the recent behavior of a node, as follows:

$$U_{ij} = \sum_{l=1}^{L^s} w^l \cdot f_{ij}^l \tag{4.6}$$

where, in order to give more relevance to the latest transaction w.r.t. the oldest one, the weights w^l of each feedback follows a geometric distribution with parameter ρ

$$w^l = \rho(1-\rho)^{l-1} + (\xi^{res}/L^s) \tag{4.7}$$

to maintain the score in the range $[0, 1]$, we introduce the term ξ^{res} which account for all the residual weight of the distribution due to the transactions older than L^s. ξ^{res} is then computed as:

$$\xi^{res} = \sum_{r=L^s+1}^{L_{ij}} \rho(1-\rho)^{r-1} \tag{4.8}$$

The **Perseverance Score** evaluates the constancy of a node in providing good services and it is computed as:

$$\begin{cases} P_{ij}^{L_{ij}} = 0.5 & \text{if } L_{ij} = 1 \\ P_{ij}^{L_{ij}} = P_{ij}^{L_{ij}-1} + V_{ij} & \text{if } L_{ij} > 1 \end{cases} \tag{4.9}$$

where V_{ij} is a parameter that reward/punish a node based on its constancy in providing good/bad services, as described by:

$$V_{ij} = \begin{cases} v_{ij}u & \text{for } f_{ij}^{L_{ij}} \geq TH \\ -v_{ij}d & \text{for } f_{ij}^{L_{ij}} < TH \end{cases} \tag{4.10}$$

u and d represent the basic increase/decrease of the score; however, consecutive good or bad transactions can further reward/penalize a node, which is then encouraged to stay benevolent, according to the value of v_{ij}: this value is calculated as the number of consecutive transactions evaluated positively/negatively by the requester. As the other scores, also the Perseverance Score is limited in the interval $[0; 1]$; in the event the score obtained from Eq. 4.9 is out of these bounds, its value is set to the nearest bound.

4.2 Model Evaluation

4.2.1 Simulation Setup

In order to test our trustworthiness model, we need a large dataset of a SIoT scenario. To this, we make use of the dataset and the QGM illustrated in Chapter 3. We decide to consider only a connected sub-network of around 800 nodes to increase the probability of two nodes interacting with each other.

Two main behaviors are implemented in the network: one is cooperative and benevolent, so that a node always provides good services and recommendations. The other one is a malevolent behavior, where a node tries to disrupt the network by implementing one of the trust attacks presented in Section 2.2.1. Table 4.1 shows the optimal configuration of the simulation parameters for the proposed system, and the different weights used for the model. For simplicity, we suppose that the service requester is able to perfectly rate the received service providing binary feedback: 1 for satisfactory services and 0 otherwise. Finally, Table 4.2 presents the values for the relations created by the objects and for their computation capabilities. Between two objects that belong to the same owner and then are linked by an OOR, the relationship factor has been assigned with the highest value. C-LORs have been set with only a slightly lower value since they are established between domestic objects and objects of the same workplace. SORs are relationships established between objects that are encountered occasionally (then owned by acquaintances) and for this reason a smaller value is given. Finally, the PORs are the riskiest, since they are created between objects of the same brand but that never met and depend only on the model object. If two nodes are tied by two or more relationships, the strongest relation with the highest factor is considered. Computation capabilities are divided into two classes: Class1 is assigned to objects with only sensing capabilities, that is, an object just capable of providing a measure of the environment status and to the RFID-tagged objects. Class2 is assigned to objects with great computational and communication capabilities; to this class belong objects such as smartphones, tablets, vehicle control units, displays, set top boxes, smart video cameras.

To find the optimal setting for the residual weight, i.e. $1 - \delta^{L_{ij}}$, we analyze the model's response at varying the other weights, namely α, β and γ. Table 4.3 displays the transaction success rate when the system has reached the steady-state phase. As expected, the external opinion has more impact than the static characteristics, since it can help to identify malicious behaviors, however, since we are considering the startup phase, they are still useful when there is no information available.

L^{tr} is selected based on the machine learning algorithm validation. As shown in the next section, the selected value ensures a sufficient initialisation for the iSVM algorithm and an efficient prediction in the classifications. The ρ parameters guarantees a compromise in the evaluation of the feedback: a value close to 1 only considers the newest feedback, while a value close to 0 considers all the feedback as equally important. Finally, u and d are picked asymmetric in order to encourage benevolent behaviours and punish malicious nodes.

4.2.2 Simulation Results for Machine Learning algorithms

This Section aims to validate the performance of the incremental SVM (iSVM) algorithm and to compare it with other incremental machine learning algorithms.

Parameter	**Description**	**Value**
α	Residual weight of the Computation Capabilities	0.3
β	Residual weight of the Relationship Factor	0.3
γ	Residual weight of the External Opinion	0.4
L^{tr}	Number of transactions to train the Dynamic Knowledge	5
TH	Threshold to consider a service as "good"	0.5
L^{s}	Temporal window to compute Usefulness and Perseverance Score	10
ρ	Parameter of the geometric distribution	0.4
u	Basic increase of the Perseverance Score	0.1
d	Basic decrease of the Perseverance Score	0.2
I	Number of nodes in the network	791
	Percentage of malicious nodes	25%

Table 4.1: Simulation Parameters

Relationship Factor

Relationship	OOR	C-LOR	SOR	POR
$\mathbf{R_{ij}}$	1	0.9	0.6	0.5

Computation Capabilities

Capabilities	Class 1	Class 2
$\mathbf{C_j}$	1	0.4

Table 4.2: Parameters for Relationship Factor and Computation Capabilities

$\alpha = 0.1$	$\beta = 0.1$	$\gamma = 0.8$	SR = 0.83
$\alpha = 0.1$	$\beta = 0.8$	$\gamma = 0.1$	SR = 0.82
$\alpha = 0.8$	$\beta = 0.1$	$\gamma = 0.1$	SR = 0.81
$\alpha = 0.3$	$\beta = 0.3$	$\gamma = 0.4$	SR = 0.85

Table 4.3: Parameters Settings

In order to validate the performance of the algorithms we have used the Receiver Operating Characteristic (ROC) curve and the Area Under the ROC (AUC) curve as performance metrics. The ROC represents the diagnostic ability of a binary classifier system, i.e. the true positive rate versus the false positive rate at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The measure of performance between the algorithms is provided by the AUC, which indicates how much a model is capable of distinguishing between classes: a model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0. We compare the performance of the iSVM with two well-known incremental algorithms, the incremental Logistic Regression (iLR) [53] and an incremental artificial neural network, the incremental Radial Basis Function network (iRBF) [54]. The testing network used for the validation is composed by a requester interacting with nodes, as providers, that implement each a different behaviour, from benevolent to all of the seven possible attacks. We vary the number of total transactions among the requester and all the providers to study the ability to learn of the algorithms; we consider that out of all the transactions, 70% of them are used to train the incremental models while the remaining 30% are used for the validation. Figure 4.2 shows the trend of the ROC curve for 4 experiments based on 160, 650, 1600 and 3200 transactions of the requester. Considering all the possible providers, this means that the number of transactions used for validation with each node is 6, 25, 60 and 120 transactions. The Figure shows how the incremental Support Vector Machine (iSVM) is able to outperform the other two algorithms: except for the first set of simulations, with only 6 transactions per node used for validation, the iSVM has the best values of AUC: the system continuously learns from the processed data so that the iSVM increases its percentage of correct predictions with the growth of the dataset of transactions. Moreover, even if the accuracy of the iSVM is low when considering few transactions per node, the proposed model is able to mediate it thanks to the training phase, which makes use of other parameters to obtain higher accuracy in selecting trustworthy nodes.

4.2.3 Simulation Results for Trust Management Model

We evaluate the performance of the proposed system by analyzing the success rate, i.e. the ratio between the number of successful transactions and the total number of transactions, or by directly calculating the level of trust computed by a node.

We compare the performance of the proposed model with two well known models by the research community that, similar to our model, are designed for the same scenario, i.e. Social IoT scenario, namely the model proposed by Nitti et al. [6] and the one presented in [20] by Chen et al. Both these models make use of a subjective approach where every node has its own vision of the network and relies on the recommendations from its friends to speed up the evaluation of trust. Differences in the performance of the models can depend on the structure of the SN considered

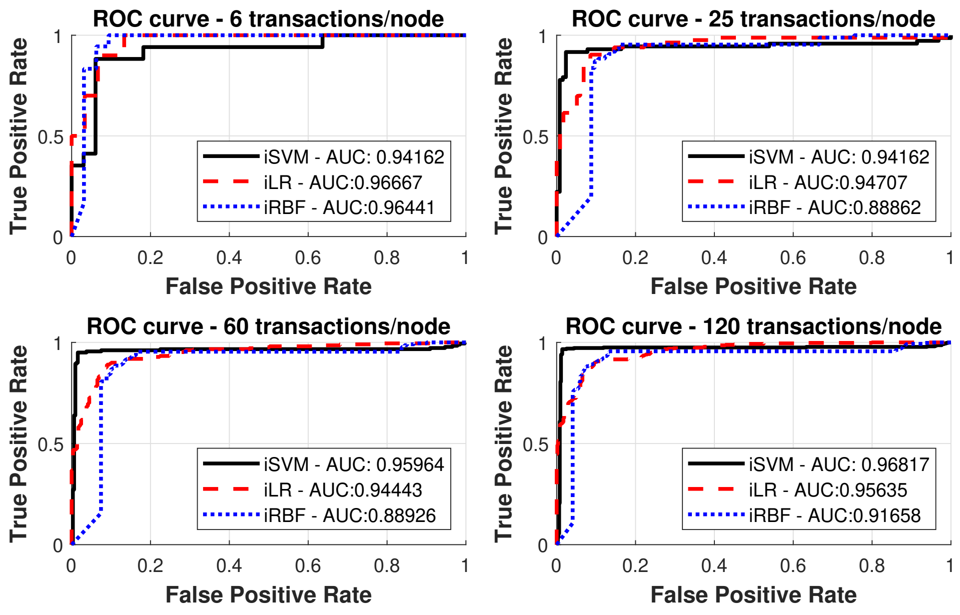

Figure 4.2: ROC curves for the machine learning algorithms for 4 experiments based on 6, 25, 60 and 120 transactions per node.

and on the types of service/information requested. To this, we did not consider our ad-hoc SN but we have adopted the Social IoT dataset described in the previous subsection, opportunistically re-scaled to a size comparable to their experiments. Moreover, we have considered the same requests for all the three models, so we are confident that the obtained results are consistent with those obtained by the authors.

These comparisons are aimed at analyzing the improvements we obtain with respect to the state of the art in the specific reference SIoT scenario. We tested all the different types of attacks, except for the SA and the SPA, which are avoided by default in our system: even if a node creates multiple identities or provides good recommendations for itself, the computed trust can not be influenced.

Figure 4.3 shows the transaction success rate when malicious nodes implement two trust-related attacks, ME and DA. We consider that 25% of the nodes are malicious and in the case of the DA, they only act maliciously with nodes that they meet occasionally or they have never met, i.e. with nodes they have a weak relation with, such as POR and SOR. All the models have a good reaction to these two attacks and are able to achieve a high success rate, ranging from 88% to 94%. This is not a surprising result, since both these attacks are usually the ones used to test trustworthiness models. All the implemented algorithms have a better performance to the Discriminatory Attack w.r.t. to the ME, even if devices implementing ME are

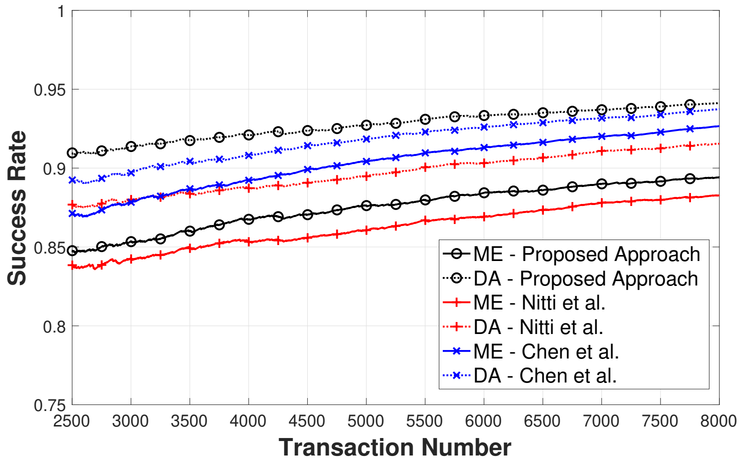

Figure 4.3: Transaction success rate for two classes of trust attacks.

		Proposed	Chen et al.	Nitti et al.
Requester	n_1	$T_{14} = 0.97$	$T_{14} = 0.81$	$T_{14} = 0.83$
	n_2	$T_{24} = 0.97$	$T_{24} = 0.8$	$T_{24} = 0.8$
	n_3	$T_{34} = 0.02$	$T_{34} = 0.31$	$T_{34} = 0.42$

Table 4.4: Performance Comparison of the Three Models against the Discriminatory Attack.

easier to be identified since they do not behave differently according to the requester: this can be explained considering that due to the changing behavior of the DA, the total number of transactions in which a node acts as malicious are only a subset of all its transaction.

To better understand how the three models react to the DA, we set up a small network of 4 nodes fully connected, where 3 benevolent nodes, n_1, n_2 and n_3, have 15 interactions each with one malevolent DA node n_4. Only the relation $\{n_3, n_4\}$ is weak, so n_4 only behaves maliciously with n_3 and benevolent with n_1 and n_2. The results are shown in Table 4.4: as expected, in all the models, both n_1 and n_2 have a high trust value for n_4 while, despite n_3 is able to identify n_4 as a malevolent node in all the models, the trust value obtained is highly variable. Only our proposed model assigns a really low trust value to n_4, while the other two models compute higher values due to the strong influence of the common friends within their algorithms.

We now want to analyze the results at varying percentage of the malicious nodes. Figure 4.4 refers to a scenario where all the malicious nodes implement ME: it shows that even with 70% of malicious nodes the success rate is over 50% and the algorithm is still able to converge. This happens since every node has its own vision of the network based on the acquired Dynamic Knowledge, however, the accuracy decreases, since it increases the possibility that all the available service providers

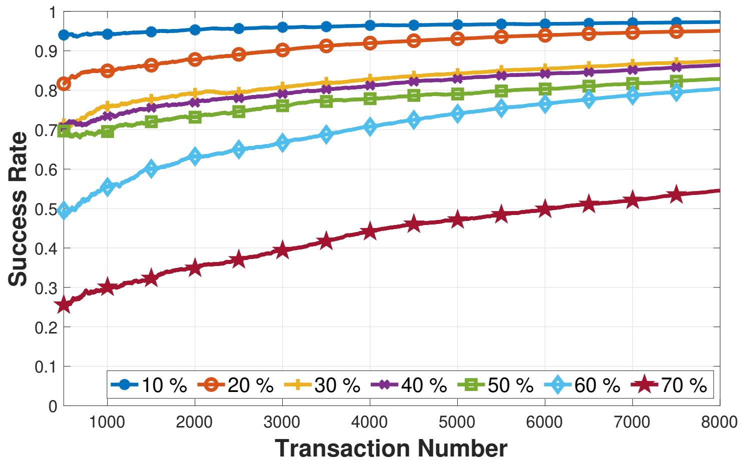

Figure 4.4: Transaction success rate at increasing values of % of malicious nodes.

are malicious. We need more than 75% of malicious nodes for the success rate to drop below 0.5: we have run a similar test also for the other two algorithms: Chen's algorithm is able to resist over 80% of malicious nodes while Nitti's performance is similar to our with 75% of malicious nodes. This result is related to the subjective approach of these two models, where each node takes its own decisions.

The focus of the next set of simulations is to test how the proposed model works with the dynamic behavior of the nodes, i.e. against the OOA. We suppose that after 40 transactions, a malicious node starts to change its behavior from benevolent to malicious and vice versa every 20 transactions. Figure 4.5 illustrates the trust value of a node performing such attack and shows how the algorithm is able to quickly adapt to the changes in the node behavior: only 3 transactions are needed to modify the trust value of the malicious node, both when the node is exploiting its good reputation and when it is trying to build up its trust. Table 4.5 presents a comparison with the other two models in terms of the number of transactions needed to change the trust value past 0.5 and highlighting the initial and final trust, T_i and T_f respectively, computed before and after the changing behavior. We note how our model is the fastest one to recognize the dynamic behavior so that only a node changing its behavior every 2 transactions is able to successfully being undetected. Moreover, we also observe that the final trust values T_f assigned by our model are rather confident, since they are closer to the trust limits, i.e. 0 for malicious nodes and 1 for benevolent nodes, while the other two models compute a trust value of around 0.5, thus indicating uncertainty in the evaluation of the node.

The next set of simulations focus on the reaction of the models against BSA (solid lines) and WA (dotted lines), as shown in Figure 4.6. In the BSA case, the requester node receives high recommendation values concerning a malicious provider from *two* common malicious friends. To tackle this attack is important to understand how

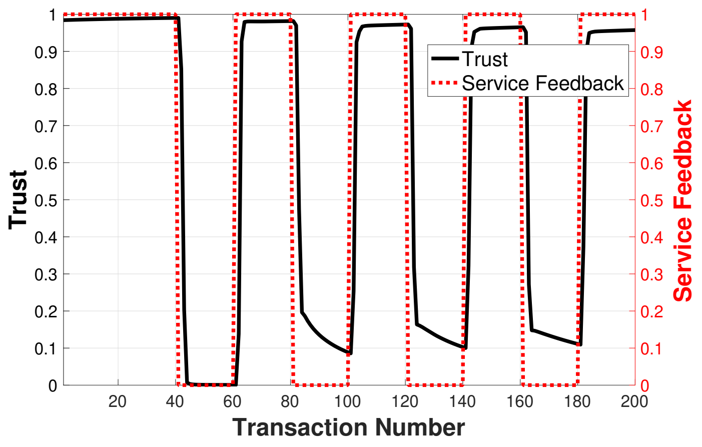

Figure 4.5: Trust value of a malicious node that performs an On-Off Attack.

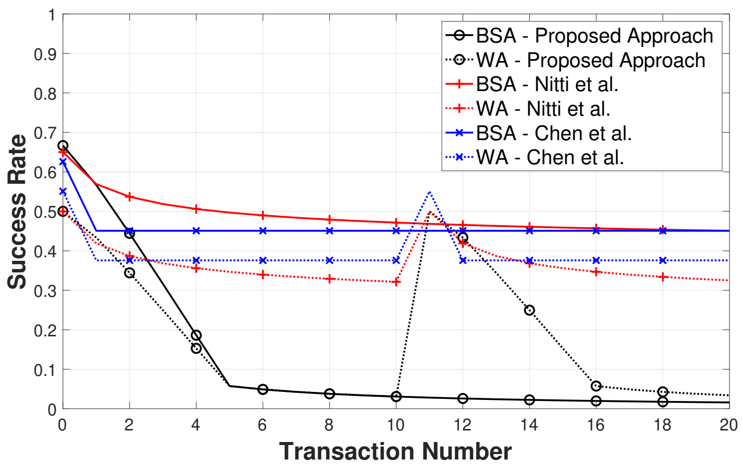

Figure 4.6: Trust value of a malicious node that performs a Ballot Stuffing Attack and a Whitewashing Attack.

	Proposed			**Chen et al.**			**Nitti et al.**		
	# tr.	T_i	T_f	# tr.	T_i	T_f	# tr.	T_i	T_f
ON → OFF	3	0.99	0.09	4	0.81	0.49	5	0.82	0.45
OFF → ON	3	0.05	0.77	5	0.41	0.5	5	0.17	0.5

Table 4.5: Performance Comparison of the Three Models against the On-Off Attack.

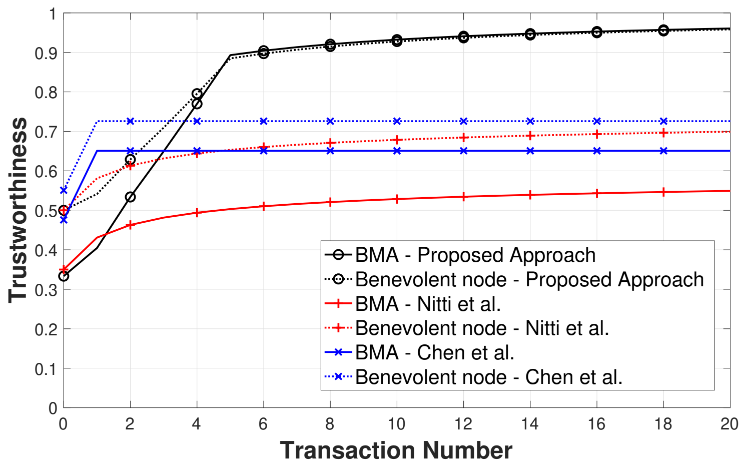

Figure 4.7: Trust value of benevolent node with or without BMA.

each model manages the recommendations received by the common friends: in our model, such recommendations are used only in the startup phase and their weight decreases with the number of transactions as the Dynamic Knowledge acquires more experience (see Eq. 4.2). Chen's and Nitti's algorithms share a similar approach: the indirect opinion has always a certain relevance in the trust score computation, however, its weight is different in the two models (0.15 in Chen's algorithm and 0.3 in Nitti's). From the Figure, we can see the trust value of the nodes implementing BSA and we can notice how our proposed model is almost non affected by the BSA (low trust values after only a few transactions), while the trust value computed by the other two models is definitely higher but still under the 0.5 threshold, thus marking the BSA nodes as malicious. In the case of WA, a malicious node with a bad reputation after 10 transaction leaves and re-joins the network to reset its trust to the default value. All the models are able to identify the node with few transactions and to label it again as malicious. However, Nitti's and Chen's algorithms are more robust to this attack, since the gain in the trust value of the WA node is lower w.r.t. our model.

The next set of experiments tests the BMA, where a malicious node provides false recommendations to decrease the trust of benevolent nodes. We first test if this attack could lead a requester to choose a malevolent node over a benevolent one: all the models select the malevolent node *only once* and are then able to select the

	% of Positive Transactions		
Trust Percentile	**Proposed**	**Nitti**	**Chen**
10%	100	86	82
20%	93	68	67
30%	86	61	57

Table 4.6: Percentage of positive transactions for an Opportunistic Service Attack over 100 transactions

benevolent node. This is due to the higher importance given by the models to the direct experiences w.r.t. indirect recommendations. Moreover, the number of nodes implementing BMA does not affect this result. Then we investigate how the trust value changes in a scenario where a benevolent node is attacked by bad-mouthing nodes w.r.t. a benevolent node with no attackers. Figure 4.7 shows how our model is only affected by the BMA in the startup phase and it is then able to achieve the same trust values for the two benevolent nodes; the other two models present a lower trust value, which does not increase with the number of transactions, due to fixed parameters external to the requester experience, such as the centrality or the computation capabilities. Moreover, it clearly appears how BMA nodes can confuse the network, especially in Nitti's algorithm which gives a higher weight to the indirect opinion than Chen's.

The next set of simulations examines the OSA, where a node changes its behavior so that its trust value computed by the requester maintains an acceptable level. However, a node's goal is not to have a high trust value but rather to have a value higher than the other providers in order to be chosen (and then have a chance to behave maliciously). To test this attack, we consider only a service requester and a malicious service provider performing the attack. We suppose that the provider is perfectly aware of its trust reputation and act maliciously only when its trust value is among the 10%, 20% and 30% percentile of the most trustworthy nodes. Considering 100 transactions between the two nodes, Table 4.6 shows the percentage of positive transactions for the three models. As expected, a larger percentile enables the malicious node to perform more attacks, however the node could not be selected as a provider if there are other possible providers for the same service. If the malicious node wants to be sure to be selected and set a stringent percentile, the number of opportunities to behave maliciously reduces. However, our approach is able to compel the malicious node to perform the highest number of positive transactions w.r.t. to the other two models, thus indicating the ability of the model to cope with this attack. In particular, if a malicious node wants to stay in the 10% percentile, it has to *always* perform benevolent.

We want now to show how the three models respond to a network with a mix of all the attacks analyzed. The result, in terms of transaction success rate, is shown in Figure 4.8, considering 5% of malicious nodes for each type of attack, for a total of 30% malicious nodes. Our model is able to converge faster and to outperform

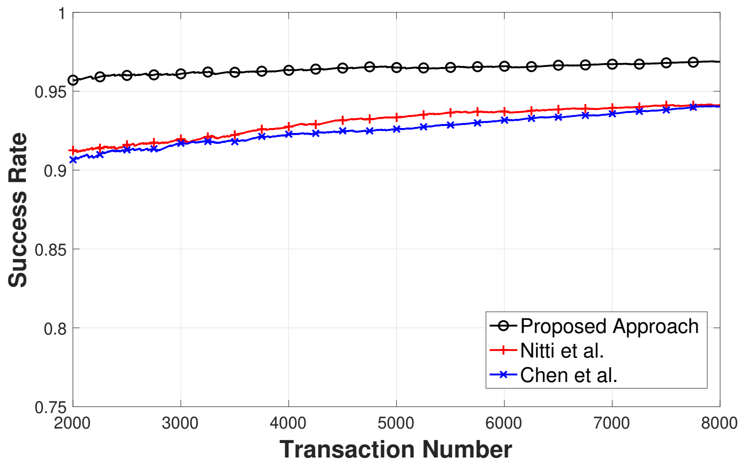

Figure 4.8: Transaction success rate with all types of malicious attacks.

	Benevolent Node - Error percentage						Malicious Node			
	0%	**10%**	**20%**	**30%**	**40%**	**50%**	**ME/ DA**	**WA**	**OOA**	**OSA**
Trust	0.99	0.86	0.79	0.72	0.57	0.28	0.02	0.05	0.12	0.88

Table 4.7: Average of Trust for the benevolent nodes with error percentage and the malicious nodes.

the other two with a success rate of over 95%. By analyzing which attacks had a higher impact, we see how simple attacks are better managed by Chen and Nitti's algorithms, however, as expected, they highly suffer smart attacks, such as OSA and OOA, which are not sufficiently tackled by them.

Finally, the last set of simulations is aimed to understand how our system reacts when benevolent nodes offer poor services due to errors related to several reasons. We then consider a requester which interacts with benevolent nodes which have a different probability to respond with an incorrect service due to some kind of error. For each value of the error percentage, we simulate 100 transactions between the nodes and mediate the results over 100 runs. Table 4.7 shows the resulting trust values for different error rates of the benevolent nodes and compare them with the trust values of malicious nodes, without considering the attacks on the recommendation, i.e. BSA and BMA. Due to the subjective approach of our model, DA performs similar to ME, since, if it is connected to the requester by a weak link, it will always provide false services; nodes implementing WA have a slightly higher trust value, since they can reset their trust to the default value. As expected, the results show how increasing the error rate, the average trust of benevolent nodes decreases. However, even for nodes with a 50% error rate, their trust is still higher than nodes implementing OOA, which has a similar behaviour, i.e. 50% benevolent transactions and 50% malicious transactions: this is due to the Perseverance Score, which evaluates negatively the consecutive bad services of the OOA. Only a node implementing OSA is able to maintain a high level of trust. In this set of simulations, we consider that an OSA node acts maliciously only when its trust value is among the 20% percentile of the most trustworthy nodes. As seen in Table 4.6, this means that the node will have more than 90% of trustworthy transactions, and thus can be considered as a node that offers bad services 10% of the time.

Chapter 5

Trust Management Models Deficiencies

According to previous chapters, the goal of any TMS is to identify malicious behaviours as soon as possible in order to isolate the nodes implementing such behaviours and discourage them from acting maliciously. However, the papers in the literature overlook essential deficiencies of the scenarios used to test TMSs. In the following chapter, we present a research study evaluation and an important valuation for classifying the most appreciated approaches in the state of the art. Figure 5.1 illustrates the distribution of the 43 analysed research papers over time according to their publishers (IEEE, Springer, Elsevier, MDPI, ACM, Hindawi, Intelligent Networks and Systems Society, The Institution of Engineering and Technology, and Bentham Science Publishers). We considered studies published online in recent years, from 2017 to 2021, which were published to provide a management method to guarantee trust in the IoT. The following string was defined to process the investigation:

- ("Trustworthiness" OR "Trust model" OR "Trust management" OR "Trust Technique") AND ("IoT" OR "Internet of Things").

The systematic review analyses all the resulting papers from our investigation in order to provide a response to the following analytical questions (AQ) in agreement with our thesis's goals:

– AQ1: Can providers make errors when providing a service?

  All the analysed trust management models rarely dealt with errors in service providing; among all the analysed papers, only [55–57] take into account this problem. Some errors involve benevolent nodes in a transaction: well-behaving devices could provide the wrong services due to errors, poor performance, poor accuracy or technical problems in general. The majority of the state-of-the-art models do not consider any type of error in the trust composition; even so,

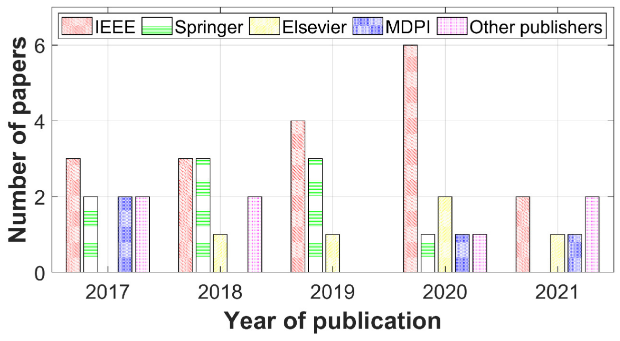

Figure 5.1: Distribution of research papers.

a few evaluate the simulations in a scenario with errors, not distinguishing provider errors from malicious behaviours. However, considering the probability of error in the trust composition could improve the performance of algorithms against errors.

– AQ2: How are the services evaluated? Is there any evaluation system?

  When a node receives the requested service, it needs to check if it is consistent and then rate it. Therefore, the feedback has the role of being the source of a trust model, which takes care of the services and their processes. Unfortunately, the community barely examines the mechanism of evaluation, and usually, in their algorithms, the nodes provide perfect feedback in a discrete dimension, where 0 represents a poor service and 1 a good one. Among the analysed works, [58–60] propose an evaluation system for receiving services, which is used to rate the interactions with the other nodes in the network. On the other side, most proposed models focus only on trust computation techniques and superficially treat feedback generation.

– AQ3: Do trust algorithms contemplate the possibility of the requester making errors in the evaluation?

  As we previously remarked, the state-of-the-art works rarely evaluate errors: the models do not tackle any discrimination between service evaluation errors (requester side) and service providing errors (provider side). However, few authors contemplate a feedback algorithm in their algorithms, even though they presuppose perfect evaluation competence in the requesters. Examining the probability of errors in the evaluation service phase could improve the scenario, which would be more comparable to reality. Therefore, evaluating the feedback could increase the number of applications making use of a trust algorithm. In all the analysed papers, only [61–63] consider errors in the feedback evaluation process.

The majority of the state-of-the-art models exhibit the same assumptions: each requester can evaluate perfectly the received service, and any probability of error is contemplated in the algorithm creation. However, a trust model should consider a probability of error, whether from the requester side (service evaluation) or the provider side (sending a service).

In the following, we show how these assumptions are essential for a trust model. In addition, we illustrate how a simple trust algorithm could replace a complex one if it does not consider errors, with the best results in terms of malicious behaviour detection.

5.1 Testing Different Trust Models

In this section, we test and compare different models in order to show the importance of accurately describing the scenario used to test the models. Our tests show that advanced techniques are not necessary in the scenarios commonly used in the literature. On the other hand, complex solutions become essential when models do not overlook error probability in service providing or evaluation.

We used as a baseline to compare other models the simplest approach possible (labelled as Basic Approach in the figures): this approach makes use only of the requester experience, i.e., the requester does not make use of recommendations from its neighbours, and only the last interaction with the provider is considered in order to evaluate its trust. This means that, since the general scenario used to test TMSs only considers the ability for the requester to perfectly evaluate the service received, the trust can only assume two values: 1 if the service was good and 0 otherwise. Moreover, if the trust of a provider reaches 0, i.e., the last service was not evaluated positively by the requester, the provider will no longer selected.

We first evaluated the performance of the basic approach by analysing the success rate, i.e., the ratio between the number of successful transactions and the total number of transactions, or by directly calculating the level of trust computed by a node. We compared its performance with well-known models in the research community. In the first work [6], Nitti et al. proposed a trust model designed for the Social IoT (SIoT) scenario. The authors propose a decentralized architecture in which each node computes the trust values of providers on the basis of its own experience and on the opinion of the neighbours. The trust is evaluated considering QoS parameters, such as transaction service quality and computation capability, and social metrics, such as centrality, relationship factors and credibility. Each node computes the trust value of providers applying a static weighted sum, considering all the mentioned parameters and feedback of past interactions. Another work designed for the social scenario is presented in [20]. Chen et al. illustrate a scheme for service access based on recommendations. The authors considered both QoS metrics, such as energy status, quality reputation and social relationships. In the scheme, each node has its own vision of the network and relies on the recommendations from its friends to speed up the evaluation of trust. The final trust values are computed based on the parameters and past performances toward a weighted sum. The other two models, i.e., those from Adewuyi et al. [64] and Mendoza et al. [65], are designed for a generic IoT scenario. In the first work, the authors propose a subjective approach to evaluate and manage trust between nodes in collaborative applications. A trust aggregation function based on a weighted sum is used to calculate trust values. A concept based on trust decay is introduced to address the issue of trust update, and many resources, such as recommendations and past experience, are used for the computation. In addition, in the second study, the authors present a distributed trust management based on only direct information acquired by communication between nodes. The model considers only service quality attributes; it assigns positive trust value to the node that collaborates in service providing and a negative trust value to the node that refuses to cooperate. No social attributes are considered, and the model proposes mitigating attacks towards a reward and punishment mechanism and analysing QoS attributes.

Figure 5.2 shows the transaction success rate when malicious nodes implement the ME attack, i.e., nodes that act maliciously with everyone providing inadequate services, under the scenario conditions at varying percentages of the malicious nodes. We considered that 10%, 20% and 30% of the nodes are malicious and that every time a requester is looking for a service there is an average of 60 possible providers. All the models had a good reaction to this attack and were able to achieve a high success rate, always higher than 95%, 93% and 80%. However, it is clear that the basic approach is able to outperform the other approaches: this happens since as soon as a requester detects a provider implementing the ME attack, that requester is labelled as malicious, its trust reaches 0 and it is never selected again. Moreover, we can see how models have different behaviours for various percentages of errors, and each algorithm performs differently with respect to others for several percentages of errors.

As the first result, the simplest algorithm would seem to overcome the complex ones; this is possible because most trust models overlook the important issue of errors in the scenario used to test them. Benevolent providers could supply inadequate services due to malfunctions or scarce accuracy; in the same way, requesters are not able to accurately evaluate received services, and they could consider a good service as a bad one. To test this critical condition, we inserted an error percentage in which a node could make an error in service providing or service evaluation. Moreover, an important parameter that has to be considered is represented by the providers' availability. Figure 5.3 shows the performance of the trust models based on the error percentage, with different averages of available providers. Each graph exhibits the success rate after 12,000 transactions for all the evaluated models at the variation of error percentage. The results illustrate how the basic approach performs well for a high number of providers, even with the increasing percentage of error. However, the lower the average of the providers, the higher the probability of the blocking of the basic approach. The blocking problem is represented by the number of malicious nodes discarded by the simplest algorithm, which does not allow for the selection of

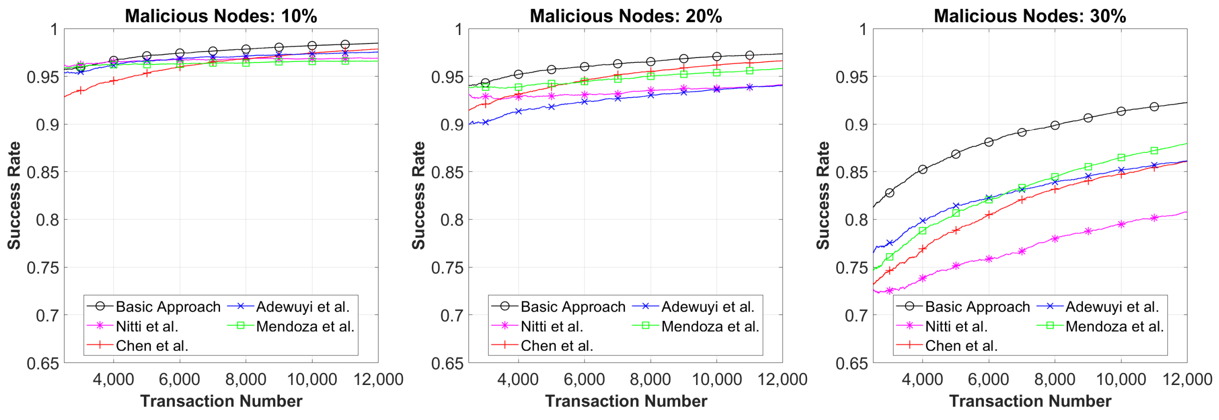

Figure 5.2: Transaction success rate for different trust management models for the ME attack at varying percentages of the malicious nodes.

malicious providers, i.e., the requesters do not select any provider. The probability of block increases with reductions in the number of providers, regardless of the error percentage. With the minimum number of available providers, that is, only one provider, the smallest error level provokes the interruption of the algorithm and its uselessness.

In order to overcome the blocking problem, we integrated the simplest algorithm with a tolerance mechanism: each requester considers a window of past interactions, which can be used to evaluate the trust of the providers. Then the trust is calculated as the average of the past feedback, i.e., evaluations of the historical interactions. Therefore, to improve the functioning of the basic approach, we needed to increase the simple algorithm complexity. The larger the window, the higher the probability of attack by malicious nodes; otherwise, the narrower the window, the higher the probability of errors. We next wanted to show the behaviour of the new basic approach with different dimensions of transaction windows considering an average of 20 providers. Figure 5.4 illustrates how the models respond to a network with a mix of all the attacks analysed. The results are shown in terms of transaction success rate considering 10% of malicious nodes for each type of attack, for a total of 40% malicious nodes. The graph depicts how the complex approaches are able to converge well and better than the basic approach with different dimensions of the window. Until 15% of error percentage, the narrowest window operates well in terms of success rate, whereas by increasing the error the best results were revealed with the larger window. In any case, the basic approach suffers from complex attacks which only the well-designed trust models can overcome. By analysing which attacks had a higher impact, we can see how the basic approach better manages simple attacks; however, it suffers from smart attacks, such as OSA and OOA, which are not sufficiently detected.

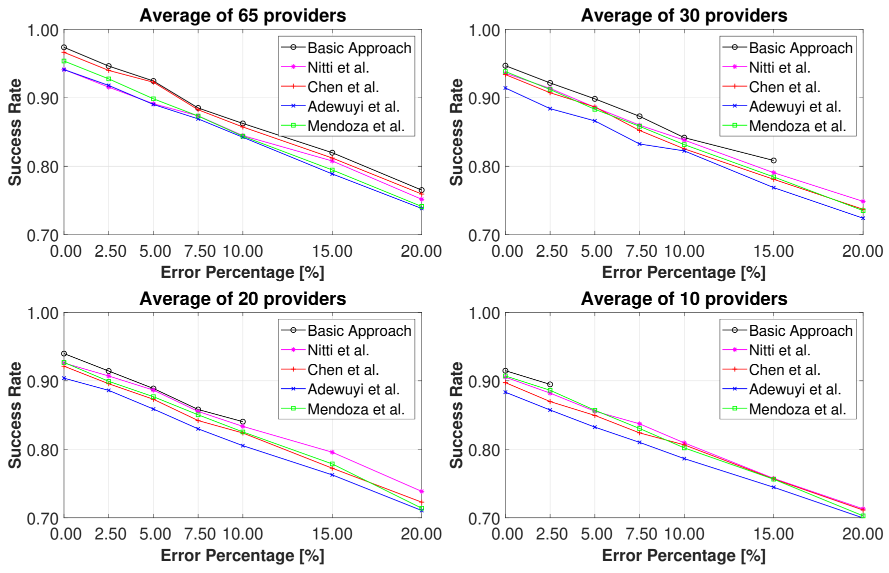

Figure 5.3: Transaction success rate for different averages of providers and error percentages.

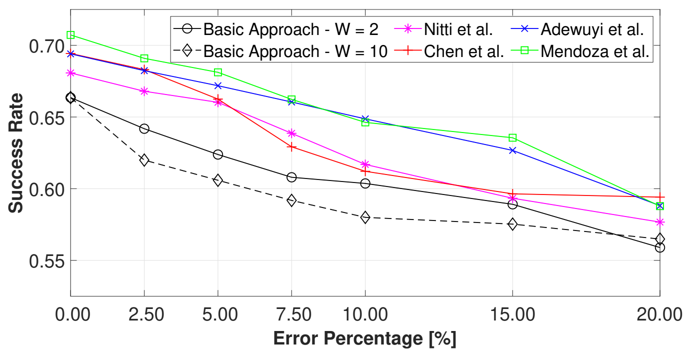

Figure 5.4: Transaction success rate with all types of malicious attacks.

Chapter 6

Feedback evaluation

In order to overcome the deficiencies of the trust models proposed in the literature, this chapter proposes a feedback evaluation mechanism able to rate the service received by the requester.

In literature, a generic Trust and Reputation Model (TRM) usually assume the requester is able to perfectly evaluate the service and then it has to select only one of the providers in $\mathcal{P}_h$ based on their level of trust. However, the requester does not know the ground-truth value v^h of the service and has then no means to evaluate if the received service is good or not and its level of accuracy. In order to assess a value for the service to be used by the application, the requester has to contact more than one provider in $\mathcal{P}_h$: from every provider n_j, the requester will receive a value v_j^h for the service S_h and then has to aggregate all the received values into a reference value v^{h*}. Moreover, to compute the trust level of every provider, a TRM has to rely on the previous interactions among the nodes in the form of feedback, which represents how a requester is satisfied with the received service. After every transaction, the requester n_i has to assign a feedback to all the providers to evaluate the service, based on the provided values v_j^h and on their "distance" from the computed reference value v^{h*}. Each feedback can be expressed using values in the continuous range $[0, 1]$, where 1 is used when the requester is fully satisfied by the service and 0 otherwise.

According to the scenario presented in Chapter 3, Figure 6.1 provides a simple example of a generic graph $\mathcal{N} = \{n_1, ..., n_1 1\}$, with each node capable of providing one or more services, as highlighted in the grey clouds; n_1 is the node that is requesting the service S_7, as highlighted in the white cloud; $\mathcal{P}_h = \{n_5, n_6, n_9\}$ is the set of nodes that can provide the requested service. For each of the provider in $\mathcal{P}_h$, the requester n_1 receives a value for the service S_7 (red lines in the Figure) then aggregate them to compute the reference value v^{7*}. Based on this value, n_1 can rate the providers individually and assign to each of them feedback (dotted green lines in the Figure) that can be used to update their trustworthiness levels.

The goal of the feedback evaluation algorithm is twofold: compute the reference value v^{h*} that will be used by the application and estimate the reliability of the

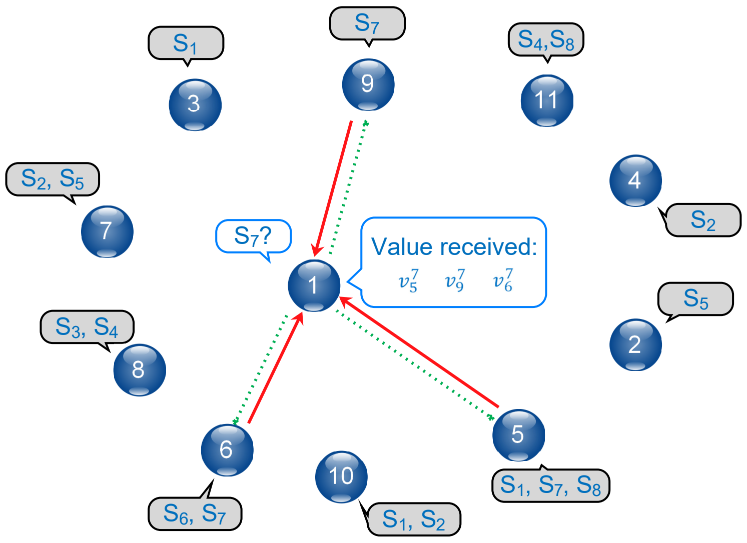

Figure 6.1: Reference Scenario.

providers. This step is fundamental to help the requester to assess a solid value for the requested service and to reward/penalize the providers so as to avoid any malicious node in future transactions.

6.1 Feedback Evaluation Model

We propose a service and feedback evaluation model, where each node, during a transaction, calculates the reference value v^{h*} to be passed to the application. This value is then used as a reference to compute the feedback that will be assigned to the different providers. In this way, every TRM can have information regarding the past interactions available to be used for the trust computation value.

Whenever a node n_i has the need to retrieve a service, it has to select a subset of providers from the list of potential providers $\mathcal{P}_h$. To this, the requester immediately discards any provider with a trust value lower than a given threshold TH, so that the reliable providers are included in the set $\mathcal{T}_h = \{n_j \in \mathcal{P}_h : T_{ij} \geq TH\}$, where T_{ij} is the trust of node n_i towards node n_j. The value of the threshold has to be decided based on the TRM implemented in the system since different models have different dynamics to label a node as malicious. From this set, the requester contacts the M most trustworthy providers in order to actually require a service's value

When the requester receives all the service's values from the providers, it has to implement some mechanism in order to infer the reference value v^{h*} to be passed to the application. Several strategies can be used:

- mean of all the values obtained by the M providers;

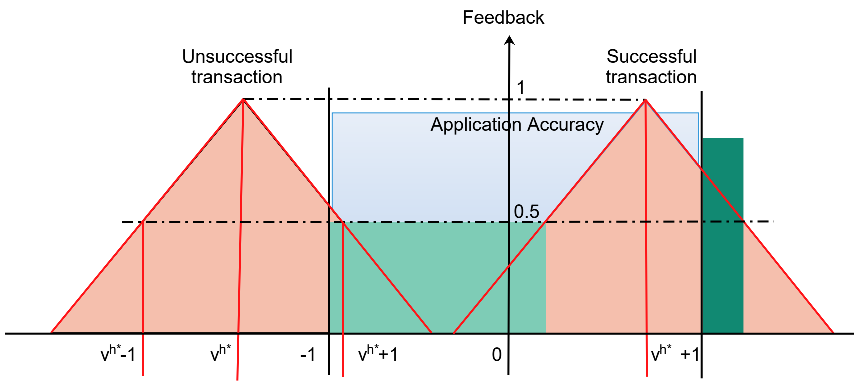

Figure 6.2: Feedback evaluation.

- sum of all the values obtained by the M providers weighted by their trustworthiness;
- median of all the values obtained by the M providers.

Since every application requires a certain accuracy, it is possible to assign feedback to the different providers. For simplicity, we consider the ground-truth value to be 0 and we always normalize the application accuracy so that values in the interval $[-1, 1]$ are acceptable by the application (blue area in Figure 6.2). The reference value v^{h*}, calculated with one of the metrics proposed above, can deviate from the ground-truth value; in this case, if v^{h*} is still within the application accuracy, i.e. $-1 \leq v^{h*} \leq 1$, the transaction is labelled as successful, otherwise it means the malicious attack was able to confuse the network and the transaction was unsuccessful.

The requester is unable to assess the outcome of the transaction, so despite its result, it has to assign feedback to each of the providers. The maximum feedback is assigned to those providers that sent exactly the reference value v^{h*}, while the other providers receive lower feedback based on how much the provided value is distant from v^{h*}, as shown by the orange areas in Figure 6.2. Nodes that have provided values with a distance equal to the accuracy of the application, i.e. $v^{h*} - 1$ and $v^{h*} + 1$, which represent the points of greatest uncertainty, are assigned feedback equal to 0.5. For intermediate values, feedback follows a linear behaviour, but other approaches are feasible and could also depend on the application at hand.

Due to the difference between the pseudo and the ground-truth value, feedback assignment leads to error in the evaluation of the providers: benevolent nodes that provided values in the light green area are given feedback lower than 0.5 and in some cases even 0, while malicious nodes, which provided values unacceptable by the application, i.e. outside of the blue area of application accuracy, are given positive feedback as it is the case of the dark green area in Figure 6.2.

The introduction of a feedback evaluation system leads to a new malicious behaviour that would not be possible if the requester is able to perfectly rate the received service. Generally, malicious behaviour is a strategic behaviour corresponding to an opportunistic participant who cheats whenever it is advantageous for it to do so. The goal of a node performing maliciously is usually to provide low quality or false services in order to save its own resources; at the same time, it aims to maintain a high value of trust toward the rest of the network so that other nodes will be agreeable to provide their services when requested. A group of nodes (collusive attack) can work together to provide the same malicious value, so to influence the reference value v^{h*} and let the requester believes that it is the correct value to be passed to the application and assign to them a positive feedback. To the best of our knowledge, this is the first time this attack is presented, so TRMs were never tested against it and we do not know their ability to detect and react to such an attack.

6.2 Feedback Model Testing

This Section aims to evaluate the performance of the service and feedback evaluation. The first set of simulations aims to analyze the transaction success rate, i.e. the ratio between the number of successful transactions and the total number of transactions: a transaction is considered successful if the reference value is within the application accuracy. Figure 6.3 shows the performance of the Marche et al. trust algorithm, proposed in this thesis, when the requester can perfectly evaluate the service received and when it implements one of the three possible strategies to infer the reference value and for different values of the numbers of providers M. The mentioned model is designed for a Social IoT scenario and makes use of a subjective approach where every node has its own vision of the network and relies on the recommendations from its friends to speed up the evaluation of trust. In these simulations, malicious nodes implement a strategic behaviour corresponding to an opportunistic participant who acts maliciously with everyone. This is the most basic attack: a node always provides bad services and recommendations, regardless of the requester.

Surprisingly, the trust model making use of a perfect feedback evaluation has the lowest success rate out of all the versions: this is related to the number of providers M contacted by the requester. Requester with perfect feedback evaluation only interact with one provider and thus they will need a lot of transactions to accumulate enough experience to accurately evaluate the providers. In the approach proposed in this thesis, multiple providers are required to infer the reference value and, even if each of them receives not perfect feedback, the overall learning process of the trust algorithm speeds up. Moreover, the presence of a higher number of providers enables to increase the success rate, since it is difficult for few malicious nodes to change the reference value enough for the transaction to be considered malicious. The approach based on the median, dark lines in Figure, is the most reliable since there must be $M/2 + 1$ malicious nodes in order for the transaction to be labelled as unsuccessful;

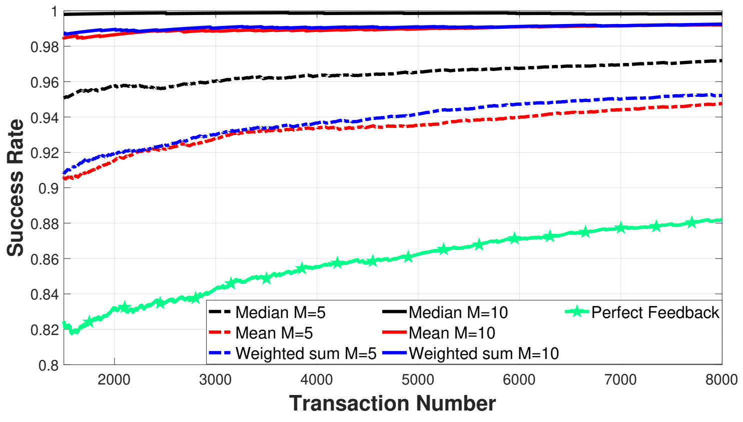

Figure 6.3: Transaction Success Rate for all strategies

using the mean is not ideal since a single malicious node providing a value very distant from the ground-truth value could significantly change the reference value. Finally, the weighted sum is able to obtain good results, since it smoothes out this effect by weighting every received value for the trustworthiness of the node that provided it.

We evaluated the system performance by analysing the mean of malicious nodes per transaction, the mean and standard deviation of the reference value, the reference value errors, and the number of errors in feedback assignment. This last parameter considers both when a malicious node gains a good score, i.e. feedback higher than 0.5, and when a benevolent node receives feedback lower than 0.5. Table 6.1 shows these results when considering 25% of malicious nodes but with two different types of attacks: the first six simulations implement the basic attack with malicious nodes always providing bad services, while the last six simulations show what happens with the collusive attack described in Section 6.1. Coherently with Figure 6.3, when implementing only the basic attack, the median is the approach with the lowest percentage of the malicious transactions, which means it is able to efficiently isolate malicious nodes. This can also be inferred by the accuracy of the reference value and by the number of times it is outside of the bounds accepted by the application. Moreover, it confirms how selecting a higher number of providers leads to better results. Differently, if malicious nodes implement the collusive behaviour, all the strategies show worse performance: if we consider the error percentage in calculating the reference value, i.e. the number of times the reference value is outside of the application accuracy, we can notice how this percentage is an order of magnitude higher w.r.t. the basic attack and in particular the median approach with $M = 5$ goes from 3.33% errors to 16.48%, thus showing the threat of the collusive behaviour. Overall, the median approach involves a higher number of malicious nodes per transaction and it has more difficulties in correctly evaluating malicious nodes, as can be inferred by the error percentage in providing feedback

	Malicious nodes per transaction [%]	**mean** v^{h*}	**SD**(v^{h*})	**Reference value errors** [%]	**Feedback Errors Malicious vs Benevolent** [%]
Median, M=10	9.9	0.0025	0.27	0.16	1.57 - 13.97
Mean, M=10	10.1	0.0041	0.44	0.81	2.11 - 15.05
Weighted sum, M=10	10.1	0.0028	0.43	0.76	2.02 - 14.85
Median, M=5	12.4	0.0346	0.79	3.33	3.44 - 18.81
Mean, M=5	12.2	0.0305	0.81	5.52	3.08 - 18.36
Weighted sum, M=5	12.4	0.0161	0.81	5.15	3.44 - 20.07
Median, M=10, Collusive	11.3	0.0050	0.98	5.15	4.60 - 16.60
Mean, M=10, Collusive	10.9	0.0035	0.83	7.73	3.31 - 19.74
Weighted sum, M=10, Collusive	11.0	0.0228	0.84	8.01	3.49 - 19.85
Median, M=5, Collusive	15.4	0.0060	1.41	14.68	9.26 - 26.66
Mean, M=5, Collusive	14.6	0.0284	1.36	15.87	7.21 - 26.86
Weighted sum, M=5, Collusive	14.6	0.0226	1.38	16.43	7.56 - 26.69

Table 6.1: Comparison between simple and collusive attacks.

which is higher w.r.t. to the other two approaches. However, the median approach still shows the lowest percentage of errors when computing the reference value.

In order to better understand this behaviour, we tested the performance of the trust algorithm at varying the percentage of the malicious nodes. Figure 6.4 refers to a scenario where all malicious nodes implement the collusive behaviour: on the left (Figure 6.4a), there is the case with $M = 5$ providers, while on the right (Figure 6.4b) we consider $M = 10$ providers. Two approaches are evaluated: the median (solid lines) and the weighted sum (dotted lines). The Figure confirms the Table above: with more providers, the trust algorithm performs slightly better but at the expense of more traffic exchanged among requester and providers. However, when comparing these results with the original trust algorithm, which implemented a perfect evaluation of the service received, we can notice how the success rate greatly decreases: for 50% of malicious nodes, the algorithm proposed by Marche et al. shows a success rate higher than 80% while, with the collusive attack, it is not able to even reach 60% of successful transactions with a decrease of over 20%.

Finally, the last set of simulations aims to understand how different trust algorithms react with 20% of nodes implementing the collusive attack and considering the median as the strategy to infer the reference value. Other than Marche et al.

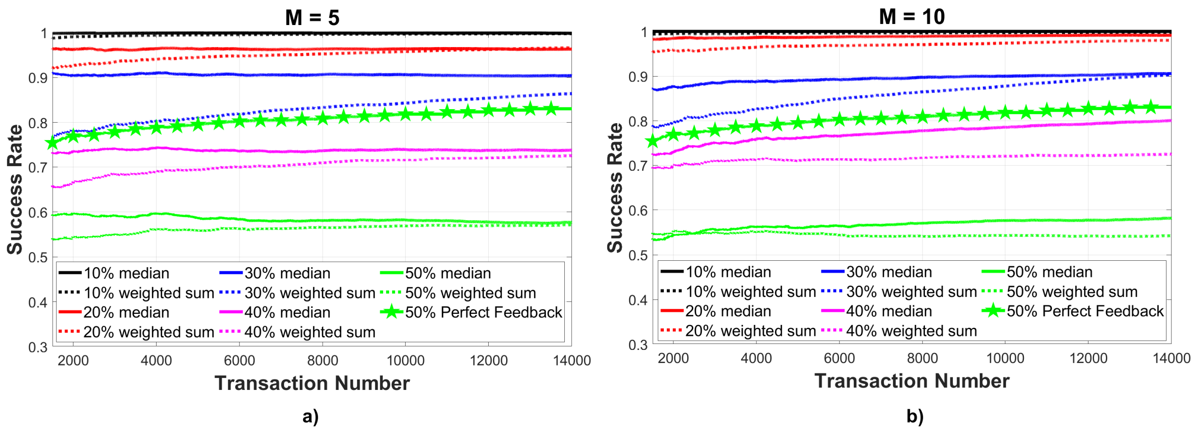

Figure 6.4: Transaction success rate at increasing values of % of malicious nodes, with $M = 5$ (a) and $M = 10$ (b) providers.

algorithm, we have also implemented three other algorithms, namely Chen et al. [20], Adewuyi et al. [64] and Mendoza et al. [65].

The model proposed by Chen et al. is similar to the one described by Marche et al.: they are both designed for a Social IoT environment where each node computes the trust value of the rest of the network by itself, so that each device has a subjective values of trust towards the rest of the network. Adewuyi et al. and Mendoza et al. are non-social algorithms, based on a distributed approach, which rely on recommendations from neighbour nodes and QoS parameters: trust is computed making use of a weighted sum among these parameters, but while Adewuyi et al. focuses on historical interactions, making use of a time window of past interactions, Mendoza et al. concentrates only on the last transaction and the type of service. Differences in the performance of the models can depend on the structure of the network considered and on the types of service/information requested. To this, we did not consider our ad-hoc network, but we have adopted the IoT dataset opportunistically re-scaled to a size comparable to their experiments, as described in the previous subsection. Moreover, we have considered the same requests for all the four models, so we are confident that the obtained results are consistent with those obtained by the authors.

Figure 6.5 shows the comparison of the four trust models considering a number of providers $M = 10$. We can notice that all the algorithms have a high success rate of over 90%, but Chen et al. and Mendoza et al. show a SR decreasing over time. This is due to the collusive attack that is able to confuse the network. The two mentioned algorithms have a short dynamic, i.e. that trust values of both benevolent and malicious nodes, are concentrated around 0.5, so they are more prone to errors. Whenever there is a transaction where the collusive malicious attack is successful, the malicious nodes are able to obtain the highest value of feedback, since the reference value would correspond with the malicious value provided by all the nodes. So, even

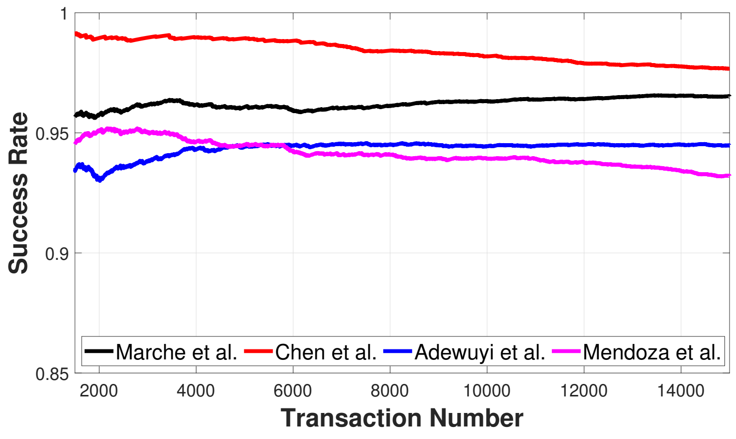

Figure 6.5: Transaction success rate for different trust management models.

if really slowly, these nodes would impact more and more transactions, thus making the success rate decreasing over time.

Chapter 7

Modelling Nodes' Interactions

In the previous Chapter, we have illustrated the importance of a feedback model in the trust processes. However, we overlook an important question: "Why do nodes have to collaborate with each other?". In this regard, in this Chapter, we model the trust game between the generic *Requester* and the generic *Provider*, which will be used to provide the guidelines to build a trust management algorithm able to detect malicious behaviours.

According to the previou chapters, in the IoT scenario, the requester has the role of the trustor and has to trust that the provider, which is then the trustee, will provide the required service. For every service request, both requester and provider have costs and benefits associated with it, so each node needs to find a tradeoff between the cost and the benefit related to a request. From the point of view of the requester, it has a cost c_r associated with its request, which can be related for example to the delay in providing the service back to the user, but it has an obvious benefit b_r related to obtaining the desired service. The provider, instead, has to consider the cost c_p to solve the request, which can be tied for example to the energy consumption to make a sensing measurement, and a benefit b_p for its reputation, which can be increased or decreased according to its behavior.

In our scenario, the requester has to select one of the providers based on their level of trust: the higher the level of trust, the higher the probability to receive the desired service, and thus to maximize the payoff. The trust level is computed according to the trustworthiness management model implemented, which has the fundamental role to identify malicious nodes. The goal of this chapter is to study the trust game between the requester and the provider as a framework and propose guidelines to design suitable trust models.

7.1 Game and Payoffs Definitions

The proposed trust game consists of a finite set of devices acting as players, where a link between two devices denotes the possibility of interactions or transactions be-

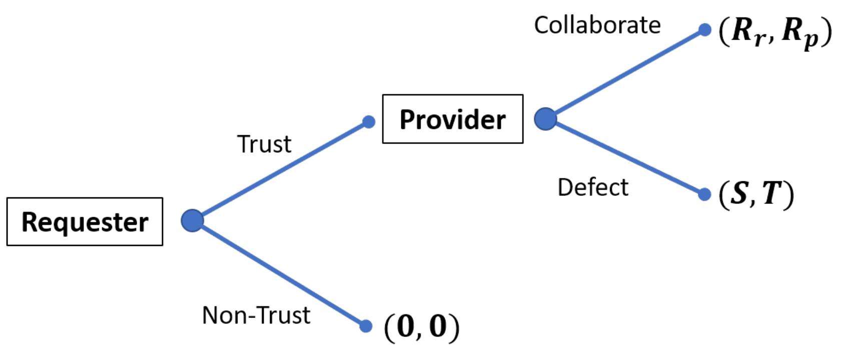

Figure 7.1: Decision tree for the trust game.

tween them. The game is based on pairwise interactions, i.e., every device interacts or transacts with other directly connected devices in pairs. Pairwise interactions proceed in two phases. A requester needs a service and can choose whether to select a provider, the trustee, to retrieve it (i.e. being trustful) or to do nothing (i.e. being not trustful). In the latter case, the game ends and both the player gets a zero payoff: the trustor has not received the service, and the trustee did not have any chance to take part in the game. In the former case, the trustor needs to consume some of its resources in order to send the request to the provider, which in return has to decide to defect or collaborate. If it collaborates, both players receive a reward, respectively R_r for the requester and R_p for the provider. But if the provider defects, i.e. it behaves maliciously, it receives a greater reward, equals to T (temptation), while the requester receives a negative payoff S. The negative payoff is due to the false service received by the requester that must be discarded, while the malicious provider safeguards its resources and obtains a greater reward. During a single interaction, for the provider, defection always results in a better payoff than cooperation, since the requester can not punish it, and so it represents a dominant strategy. The best strategy for the requester is then to be not cooperative and not to ask for services. Mutual defection is the only strong Nash equilibrium in the game, so this results in a network where nodes do not interact with each other.

Figure 7.1 illustrates the decision tree for the evaluated trust game.

However, two IoT nodes can interact more than once in succession and they can remember the previous actions of their opponents and change their strategy accordingly. Under these conditions, the game becomes iterated: nevertheless, if the game is played exactly N times and both players know the number of transactions, then it is still optimal to defect in all rounds and the Nash equilibrium is to always defect. The proof is inductive: the provider could defect on the last transaction since the requester will not have a chance to change later its strategy. Therefore, both will defect on the last transaction. Thus, the provider might as well defect on the second-to-last interaction, since the requester will defect on the last no matter the provider's behavior, and so on.

For these reasons, we consider an iterated trust game with an unknown number of transactions, where the complete payoff is determined according to the strategy adopted in each game. The incentive to behave maliciously in the short-term is compensated by the possibility for the requester to punish the provider after abuse of trust. For cooperation to emerge between game rational players, the total number of rounds must be unknown to the players. In this case "always defect" may no longer be a strictly dominant strategy, but it remains a suboptimal Nash equilibrium. Based on the previous considerations on cost and benefit for the two players, payoffs are determined as follows:

$$\begin{cases} S = -b_r - c_r \\ T = b_p \\ R_r = b_r - c_r \\ R_p = b_p - c_p \end{cases} \tag{7.1}$$

The punishment S reflects the request's cost c_r and the false benefit received, specified by $-b_r$. Moreover the temptation T is related to the benefit for the malicious behavior b_p, without any cost. In addition, the payoffs R_r and R_p are the results of the collaboration between the players: the first depends on the request's cost c_r and on the benefit of the received service b_r, whereas the second payoff concerns the provider with the cost to provide the service c_p and the benefit in terms of its reputation b_p. From this, we model the payoffs' constraints as follows:

$$\begin{aligned} &\textbf{Requester} : R_r > 0 > S \\ &\textbf{Provider} : T > R_p > 0 \end{aligned} \tag{7.2}$$

The punishment S is the worst payoff for a requester: this is due to the resources used for the request and to the false service received. Regarding the provider, the temptation T is greater than the payoff resulting from collaboration and the related reward. Moreover, the requester and provider payoffs are related by the following:

$$R_r > T \tag{7.3}$$

This relation shows how the requester obtains a greater payoff than the provider since the requester receives the desired service, while the requester increases its reputation.

The relations in terms of benefit and cost is shown below:

$$b_r - c_r > b_p \tag{7.4}$$

where it is remarked how the requester has a different benefit/cost greater than the provider; this is due to the requester that is more interested in the communication to receive the needed service.

7.2 Strategies and Attacks

In order to provide general guidelines for the development of trust algorithms, we consider a generic trust model. Due to the iterated nature of the considered trust game, the nodes' strategies must reflect their past interactions. For each interaction k, the generic trust value assigned to the provider can be expressed as:

$$T(k) = T(k-1) + \Delta \tag{7.5}$$

where $T(k-1)$ represents the trust value during the previous transaction, while Δ indicates the trust variation based on the provider's behavior in the last round:

$$\Delta = \begin{cases} -\Delta^-, & \text{malicious behavior} \\ \Delta^+, & \text{benevolent behavior} \end{cases} \tag{7.6}$$

When there is no previous information, i.e. during the first interaction, an initial trust value is assigned to each provider, so that $T(0) = T_{init}$.

The requester changes its behavior based on the computed trust value and chooses a strategy in the iterated games in order to increase its total payoff. To simplify the analysis, all the parameters are in the range $[0, 1]$, so that, e.g., a trust value of 0 indicates untrustworthy nodes, while a trust value of 1 is used for completely trustworthy nodes. Moreover, whenever the trust value of a device is lower than a given threshold T_{th}, the node is immediately labeled as malicious.

The evolution of trust depends on the strategy adopted by the provider in the previous interaction, which has a direct impact on the trust variation Δ. Two different behaviors can be considered in a network: one is always benevolent (or cooperative) and provides only good services; therefore its trust is always increased after a transaction and $\Delta = \Delta^+$. The other behavior is a strategic one corresponding to an opportunistic participant who cheats whenever it is advantageous for it to do so. A node that performs maliciously, usually provides false or scarce services in order to save its resources. Below we model the most studied attacks to the trust in the literature and study the strategies that can be used to develop a suitable trust model.

A node performing **Malicious with Everyone (ME)** acts maliciously with everyone. Regardless of the interaction, the node sends only false services and its trust value is then always reduced ($\Delta = -\Delta^-$). We can then model its trust evolution as follows:

$$T(k) = T(k-1) - \Delta^- \tag{7.7}$$

Another malicious attack is the **Whitewashing Attack (WA)**, which shows a very simple dynamic behavior. This behavior is similar to the ME, which provides only bad services, with the difference that a node performing this attack can re-join the network and then re-initialize its trust value. Similarly to the previous case, $\Delta = -\Delta^-$ and the trust evolution is the same as Equation 7.7.

A more complex dynamic attack is the **On-Off attack (OOA)**, where the malicious node changes its behavior from benevolent to malicious and vice versa every M interactions. In this case, we can model the trust variation Δ based on the first behavior adopted by the provider. If the node starts with malicious behavior, Δ can be expressed as:

$$\Delta = \begin{cases} -\Delta^-, & \text{if} \quad 2nM < k \leq (2n+1)M \\ \Delta^+, & \text{if} \quad (2n+1)M < k \leq (2n+2)M \end{cases} \tag{7.8}$$

Otherwise, if the node initially acts as a benevolent node, Δ can be computed as:

$$\Delta = \begin{cases} \Delta^+, & \text{if} \quad 2nM < k \leq (2n+1)M \\ -\Delta^-, & \text{if} \quad (2n+1)M < k \leq (2n+2)M \end{cases} \tag{7.9}$$

with $n \in \mathcal{N}$. The two Equations 7.8 and 7.9 illustrate the oscillatory behavior of this attack.

The last strategy is represented by the **Opportunistic Service Attack (OSA)**. A node performing this attack provides bad service only when its trust is at an acceptable level. It represents a rational player that performs attacks with the aim to maximize its own payoff. The node adapts its strategy in order to not be detected: to do this it defines its own trust limit T_{OSA} and behaves so that its trust value is always higher than this limit. Indeed, this limit must be greater than the threshold T_{th} in order for the node to be considered as benevolent ($T_{OSA} \geq T_{th}$). The trust variation can then be described as:

$$\Delta = \begin{cases} \Delta^+, & \text{if} \quad T(k-1) - \Delta^- < T_{OSA} \\ -\Delta^-, & \text{if} \quad T(k-1) - \Delta^- \geq T_{OSA} \end{cases} \tag{7.10}$$

When the node senses that its trust is dropping below the trust limit, it sends good services and then $\Delta = \Delta^+$. Otherwise, the node continues to provide bad services ($\Delta = -\Delta^-$).

Table 7.1 shows all the parameters used in the proposed investigation.

7.3 Trust management model guidelines

The goal of a trust model is to detect malicious nodes without discarding the cooperative nodes. With ideal conditions, a cooperative node will always provide good services and thus it will receive a positive trust variation (i.e. $\Delta = \Delta^+$, which as

Parameter	**Description**
k	Generic interaction between a requester and a provider
K	Total number of interactions between the two players
$Tinit$	Initial trust value for a provider node
T_{th}	Threshold to consider a node as "benevolent"
$\Delta = \begin{cases} \Delta^- \\ \Delta^+ \end{cases}$	Decrement/increment of the trust value based on the provider's behavior
k'	Re-join interaction for a WA node in which the trust is re-initialized
M	Number of interactions after an OOA node change its behavior
T_{OSA}	The lowest trust value accepted for a OSA node for itself

Table 7.1: Trust Model Parameters

described before can have a value in the range $[0, 1]$). For this reason, the only condition to not discard any cooperative node is trivial:

$$T_{init} \geq T_{th} \tag{7.11}$$

The initial trust represents a crucial parameter for a trust model, it establishes the number of interactions that a malicious node can take advantage of in order to act maliciously. A high value confers the best trust to the nodes, while a low value makes the model suspicious. In the community, many works take on the choice of the initial value using static characteristics of the nodes, e.g. computation capabilities of the nodes [66] or social relationships between the objects' owners [67].

However, due to the presence of malicious behaviors, stricter conditions are necessary. Indeed, the goal of any trustworthiness management model consists of maximizing the payoff for the cooperative nodes and thus isolating malicious objects as quickly as possible, i.e. with the lowest number of transactions, so that they are not selected as providers. Since a requester will discard a provider if its trust value drops below a certain threshold T_{th}, we can express the goal of a trust algorithm as:

$$max\{payoff\}^{req} \rightarrow min(k) : T^{prov}(k) < T_{th} \tag{7.12}$$

Ideally, the highest payoff for the requester is achieved if the model is able to detect a malicious node at the first malicious transaction. Starting from Equation 7.12, we derive the most suitable configurations for the trust model parameters: the initial value of trust T_{init}, the trust variation Δ^+ and Δ^- and the threshold T_{th}.

In order to detect a ME attack after the first malicious transaction, the following relation must be true:

$$T(1) = T_{init} - \Delta^- < T_{th} \tag{7.13}$$

where the reader should remember that $T(0) = T_{init}$. From Equation 7.13 we have three parameters, so we need to set two of them and calculate the last. Three conditions can then be set as follows:

$$\begin{aligned} &T_{th} > T_{init} - \Delta^- \\ &T_{init} < T_{th} + \Delta^- \\ &\Delta^- > T_{init} - T_{th} \end{aligned} \tag{7.14}$$

Similar considerations can be derived for a node implementing a WA attack. However, a node performing this attack re-initializes its trust value every k' transactions, so the conditions to maximize the payoff of the requester has to be generalized as follows:

$$min\{k\} \rightarrow k = k' + 1 \tag{7.15}$$

$$\forall (entry) : (T_{init} - \Delta^-) < T_{th} \tag{7.16}$$

where $T(k') = T_{init}$ and for each re-initialization the malicious node performing WA is identified at transaction $k' + 1$. The conditions found in Equations 7.14, calculated for a ME attack, are still valid also for WA attack.

Other conditions can be obtained considering other attacks. As described before, a node performing an OOA attack has two distinct behaviors: the node starts with M malicious interactions or with M cooperative interactions. The first behavior is similar to the ME attack: a trust algorithm can maximize the payoff of the requester by detecting the malicious node during the first (malicious) transaction and thus we obtain again the same conditions as Equation 7.14. However, if the malicious node starts with M cooperative transactions, the first malicious interaction is the $k = (M+1)$-th. In order to identify malicious behavior, the trust model's parameters should be set in order to satisfy the following:

$$T(M + 1) = T_{init} + M\Delta^+ - \Delta^- < T_{th} \tag{7.17}$$

which takes into account that the first M positive interactions have increased the trust of the node. However, M is a typical parameter of the OOA attack, so in order to avoid leaving any degree of freedom to the malicious node, it is important to set a condition that is independent of M and that can be obtained with:

$$\Delta^+ = 0 \tag{7.18}$$

With this condition, both malicious and cooperative nodes are never rewarded when providing good services, but they are still punished when delivering bad ones. Applying this condition to 7.17, it is possible to obtain the same relation as in 7.13, that can be solve applying the conditions in 7.14.

Another condition can be obtained by analyzing nodes performing the OSA

Parameter	**Value**
T_{th}	1
T_{init}	1
Δ^+	0
Δ^-	$x \in (0, 1]$

Table 7.2: Adequate value for the trust model to detect the malicious behaviors

attack. As mentioned earlier, in the OSA attack, the malicious nodes set a trust limit T_{OSA} higher than the threshold T_{th} and uses this limit to decide its behavior. From this, we can set an ideal value for T_{th}: a value lower than 1 allows the node to act maliciously, while with a value equals to 1, regardless of the trust limit T_{OSA} set by the malicious node, the node performing OSA is forced to provide only good services and it is unable to assume malicious behavior.

Table 7.2 summarizes the evaluated parameters. In order to prevent OSA attacks we must have the threshold T_{th} to 1. As a consequence of Equation 7.11, the initial value of trust must be set to $T_{init} = 1$. Moreover, from the considerations on the OOA attack, Δ^+ is set to 0 while, from the third condition in 7.14, Δ^- has to assume any value greater than 0 in order to detect immediately any malicious behaviors.

7.4 Probability of error

The conditions obtained in the previous Section represent an ideal scenario where the benevolent node will always cooperate. However, in a real IoT system, a cooperative device can be discarded from a network due to errors related to several reasons: well-behaving devices can show poor performance, due to errors, scarce accuracy, or technical problems in general. This problem is usually overlooked by trust algorithm models while it should be fundamental for them to be able to discern a malicious node from a poor behaving one. A trust model should be designed to take into account the errors of cooperative nodes, according to some admissible error rate for the model. If we consider that a benevolent node has a probability p to provide an unintentional bad service, then we can express the trust value calculated by the generic trust model as:

$$T(k) = T(k-1) + (1-p)\Delta^+ - p\Delta^- \tag{7.19}$$

In order to not isolate any benevolent node, a trust model should always be sure that $T(k) \geq T_{th}$. To this the following conditions must be met:

$$\begin{aligned} (1-p)\Delta^+ - p\Delta^- &\geq 0 \\ \Delta^+ &\geq \frac{p}{(1-p)}\Delta^- \end{aligned} \tag{7.20}$$

and, since $\Delta^- \in (0,1]$, it is possible to observe a first difference w.r.t. the errorless scenario: Δ^+ can not be equal to 0, in order for a trust model to balance any error from cooperative nodes.

However, condition 7.19 can still be violated and some cooperative nodes can be discarded: in the worst-case scenario, the first $\overline{k}$ interactions are all affected by errors. Considering consecutive transactions among nodes as independent, we can compute the probability of such a scenario as $p^{\overline{k}}$. It is then possible to set a maximum admissible error A_{max} for the trust model as:

$$A_{max} = p^{\overline{k}+1} \tag{7.21}$$

so that, by knowing the admissible model error A_{max} and the probability of error on the single transaction p, we can compute how many consecutive transactions $\overline{k}$ can be tolerated by the trust model as:

$$\overline{k} = \log_p A_{max} - 1 \tag{7.22}$$

This value determines that any node should not be discarded before $\overline{k}$ transactions, even if they are all malicious/with errors. This condition can be expressed as:

$$T(\overline{k}) = T_{init} - \overline{k}\Delta^- \geq T_{th} \tag{7.23}$$

The goal of the trust model is still to isolate the malicious nodes; similarly to what we have done in the previous section, we aim to find the conditions that allow the trust model to isolate the malicious nodes as soon as possible. For nodes performing ME and WA attacks, the first useful transaction to detect the malicious nodes is the $(\overline{k}+1)$-th transaction, and then the following condition must be true:

$$T(\overline{k}+1) = T_{init} - (\overline{k}+1)\Delta^- < T_{th} \tag{7.24}$$

so that the first $\overline{k}$ transactions allow errors to occur, while the system recognizes a malicious behavior afterwards. Similarly to 7.14, three conditions can be set as follows:

$$\begin{aligned} T_{th} &> T_{init} - (\overline{k}+1)\Delta^- \\ T_{init} &< T_{th} + (\overline{k}+1)\Delta^- \\ \Delta^- &> \frac{T_{init} - T_{th}}{\overline{k}+1} \end{aligned} \tag{7.25}$$

where each parameter is described based on the other two and $\Delta^- \in (0, \frac{1}{\overline{k}+1})$.

Another condition can be obtained by analyzing the OOA attack. If the node implementing OOA starts with a malicious behavior performing M malicious transactions, it is possible to devise the same condition as Equation 7.25. However, if the

Parameter	**Value**
T_{th}	$T_{th} > T_{init} - (\overline{k}+1)\Delta^-$
T_{init}	1
Δ^+	$\frac{p}{(1-p)}\Delta^-$
Δ^-	$x \in (0, \frac{1}{\overline{k}+1})$

Table 7.3: Final parameters' value for the trust model to detect the malicious behaviors in the scenario with errors.

Parameter	**Description**
p	Probability of error in a generic interaction
A_{th}	Maximum tolerable error for the trustworthiness management model
A_{max}	Maximum error for a trust model
$\overline{k}$	Number of consecutive malicious behaviors permitted by the trust model

Table 7.4: Trust Model Parameters considering an error probability

malicious node starts with M benevolent transactions, the trust model can identify the malicious node at the $k = (M + \overline{k} + 1)$-th transaction, if the following condition is satisfied:

$$(T_{init} + M\Delta^+ - (\overline{k}+1)\Delta^-) - T_{th} < 0 \tag{7.26}$$

In order to minimize the impact of M and the number of transactions needed to detect the OOA attack, Δ^+ should be set at the minimum admissible value, which can be derived from Equation 7.20 as:

$$\Delta^+ = \frac{p}{(1-p)}\Delta^- \tag{7.27}$$

which allows cooperative nodes to avoid being discarded due to errors but, at the same time, enables the trust model to identify quickly the OOA attack.

Finally, it is not possible to devise any further conditions by analyzing the OSA attack. In order to satisfy Equation 7.25, it is not possible to set T_{th} equal to 1, as in the errorless scenario. This allows a node implementing OSA to perform malicious transactions with a rate equal to the error probability p, since performing a higher number of malicious transactions would cause the node to be detected. In this way, the value for T_{th} must follow Equation 7.25 correlated to the probability of error p, while T_{init} is set equal to 1 in order to overcome the scenario without error as well. Table 7.3 shows the adequate parameters' values in order to design a suitable trust management model, both without or with an error probability for cooperative nodes.

Table 7.4 shows all the parameters used to study the trust management models with an error probability on cooperative nodes.

7.5 Game Evaluation

This section presents the setup of the game's parameters and illustrates the performance of the proposed guidelines.

7.5.1 Game Setup

In order to test the effectiveness of the guidelines for a trustworthiness model examined in the previous section, we need to simulate the binary game and all the possible behaviors. To this, we make use of a full-mesh network of $N = 100$ devices, where each device interacts K times, not known beforehand by the devices, with each other device, alternating the roles of provider and requester. This way, all nodes can play all the possible strategies in the game. For each pairwise interaction, two players, one acting as a requester and the other as a provider, play the game and change their strategies according to their behavior in the previous rounds and according to the adopted trust model.

According to Equation 7.4, we can set the values of all the payoffs. Different values might be assigned to the parameters, however, if they respect the exposed relations the final game would be the same. In this case, the greatest value is assigned to the requester's benefit since it receives the required service, while the provider has a minor benefit related to its reputation. Taking into account the cost, the provider needs to use its own resources in order to solve the request, and then the cost is higher w.r.t. the requester, where the cost is associated with the time spent to obtain the service and to the resources needed to send the request. For each game, the payoff for the two nodes, requester and provider, is computed according to the payoffs' values and the total payoff for a node is the sum of all the games.

The interactions follow the trust model based on the guidelines exposed in Section 7.3 in order to detect the malicious behaviors and guarantee a high payoff for the benevolent. To do this, we set Δ^- equal to 0.1 in order to satisfy the condition $\Delta^- > 0$ and $T_{init} = 1$ to grant the highest possible initial trustworthiness to all the nodes, while Δ^+ and T_{th} are consequently evaluated for the specific set of simulations. The maximum admissible error for the algorithm is set to $A_{max} = 10^{-3}$ in order to reach a compromise between the errors due to the cooperative nodes and the bad services provided by malicious nodes: considering an error probability equal to $p = 0.2$, $\overline{k} = 3.29$ that allows a number of consecutive errors starting from T_{init} equal to 3.

Moreover, malicious nodes are designed according to the description supplied in Section 7.2. All the behaviors, both benevolent and malicious, are used to measure the effectiveness of the guidelines for the trust management model. ME behavior always provides bad services and defects in all transactions. Similarly, a node implementing the WA attack acts maliciously with everyone, but after a fixed number

Parameter	**Description**	**Value**
N	Number of nodes	100
K	Number of interactions between two players	100
b_r	Benefit value for the requester	0.5
b_p	Benefit value for the provider	0.3
c_r	Cost value for the requester	0.1
c_p	Cost value for the provider	0.2
Δ^-	Decrement of the trust value	0.1
T_{init}	Initial trust value	1
A_{max}	Maximum tolerable error for the model	0.001
p	Probability of error	0.2
$\overline{k}$	Consecutive malicious behaviors permitted	3
Δ^+	Increment of the trust value	0.025
T_{th}	Decision threshold	0.625

Table 7.5: Simulation Parameters

of transactions, 25 for our simulations, it resets its trust value by leaving and re-entering the network. Nodes performing the OOA attack change their state from ON to OFF and vice versa every $M = 5$ interactions, starting from the cooperative behavior which is harder to detect. Finally, the OSA node represents a smart attacker that modifies its T_{OSA} threshold, which is used in order to choose a behavior. In the first set of simulations, the OOA node sets $T_{OSA} = T_{th}$ so as to have more possibilities to act malicious and to increase its payoff.

Table 7.5 shows all the configuration parameters for the proposed simulations, the different payoffs used for the game, and the trust model details.

7.5.2 Experimental Game Results

We evaluate the performance of the proposed guidelines by analyzing the binary trust game in the simulated IoT network. Each device is alternately a requester or a provider and has interactions with all the other nodes.

We first examine a scenario with a population composed of only cooperative nodes and no trust management model: the goal is to understand which is the payoff that can be achieved in an ideal network. Each requester trusts all the providers, while providers have benevolent behavior and collaborate in all interactions. The payoff average value for a single node is 24.75. This value consists of the maximum payoff achievable in a game of 100 interactions by a cooperative node. The node is not interested to preserve its own resources and then collaborates in each game.

Starting from the case with only cooperative nodes, we add malicious nodes to the network to illustrate how the attackers can obtain a greater payoff. Out of the total number of nodes in the network, we replace 5% of the nodes for each type of attack's behavior, so that the final network is composed of 80 cooperative nodes and

	Benevolent	**ME**	**WA**	**OOA**	**OSA**
Average Payoff	16.00	26.40	26.40	21.20	26.40

Table 7.6: Average payoffs of nodes without any trust model.

	Benevolent	**ME**	**WA**	**OOA**	**OSA**
Average Payoff	21.73	17.28	17.57	17.76	21.73

Table 7.7: Average payoffs of nodes in a no error scenario with a trust model.

20 malicious nodes, evenly distributed among the four types of attacks. No trust algorithm is implemented, so the requesters will always choose to play the game, while malicious providers will behave according to the implemented attack. Table 7.6 illustrates the resulting average payoffs for all the behaviors: the results show how cooperative nodes achieve the minimum payoff, which is lower than the previous scenario due to the negative payoff S. Indeed, a requester interacting with a malicious provider will not receive any service, so its average payoff decreases from 24.75 to 16.00. On the other hand, malicious nodes receive the best payoff by acting maliciously: this payoff is higher w.r.t. the case with only cooperative nodes, because malicious nodes do not have to use their resource to produce the service (sense the environment or act on something). ME, WA, and OSA can always defect without being detected due to the absence of the trustworthiness management model. The OOA behavior presents a slightly lower payoff due to a certain number of transactions during the ON state, where the node performs benevolently; anyway, the malicious interactions allow them to receive a greater payoff w.r.t. the cooperative nodes.

The employment of a suitable trust model is essential in order to detect the malicious nodes and increase the payoff of the cooperative nodes. To this, the next set of experiments examines the same network used in the previous scenario, i.e. with both cooperative and malicious nodes, but adopting a trust management model. Starting with the scenario where cooperative nodes always deliver the right service, i.e. they are not subject to errors, we design the trust model according to Section 7.5.1: $T_{init} = 1$ and $\Delta^- = 0.1$ in order to trust all nodes at start and to detect as fast as possible the malicious behaviors. Moreover, T_{th} and Δ^+ are set based on Table 7.2: for the case without errors for cooperative nodes, $T_{th} = T_{init} = 1$ to detect the attackers at their first malicious transactions and $\Delta^+ = 0$ to never increase the trust for intelligent malicious nodes, such as the OSA. Table 7.7 illustrates the average payoffs of the nodes using the trust model previously described. The trust model is able to detect ME, WA and OOA behaviors thus reducing their average payoff with an advantage for the cooperative nodes. Even with a very low value of Δ^-, the model discards these malicious nodes after the first malicious transaction. The result is that when a malicious node acts as a provider, it will not be trusted and then it will not receive a payoff from any of the benevolent requesters: it will

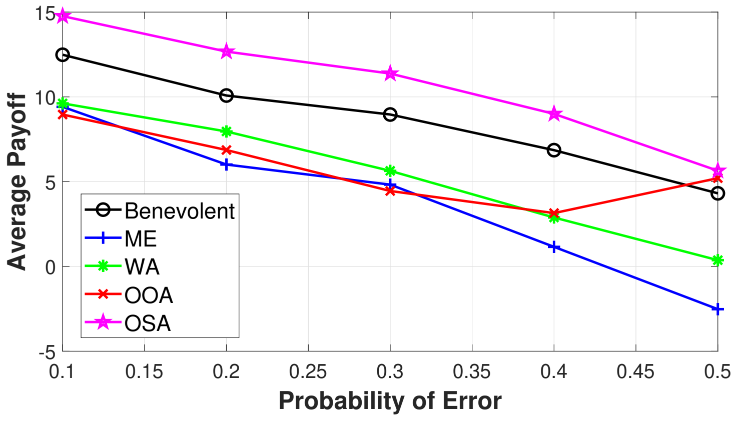

Figure 7.2: Average payoffs for different probability of error for the cooperative nodes.

only be able to accumulate payoff when it acts as a requester and trusts other nodes. The WA behavior resets its trust reputation after 25 interactions and then it has a slightly higher payoff than the ME attack but, nevertheless, it is detected immediately after the first malicious interaction. The OOA behavior can achieve an even higher payoff compared to the other two behaviors by acting as a benevolent node and providing good services in its ON state (the first M transactions); however, when the node switches to the OFF state, it is immediately detected at the first malicious interaction. Finally, the OSA behavior exhibits the highest payoff equal to the cooperative nodes. The node performing the OSA attack changes its behavior in order to be chosen as the provider and to not be discarded. However, since the model can detect a malicious node at its first interaction, the malicious node must always perform benevolently and thus achieve the same payoff as the cooperative nodes. Finally, we can observe how the average payoff of a cooperative node in this scenario is equal to 21.73, which is lower when compared to the payoff of 24.75 of the benevolent node in a completely cooperative network. This is tied to the presence of malicious nodes that decreases the average payoff for cooperative nodes even though the trust model detects them at the first interaction.

We now want to analyze the results when cooperative nodes can send bad services to the requester due to unintentional errors. The focus of the next test of simulations is to test how the trust model can be designed in order to take into account an error probability. Figure 7.2 shows the average payoffs for different values of the error probability p. For each value, and considering $T_{init} = 1$, the model can set T_{th} according to Equation 7.24. The Figure shows how the average payoff for all the possible behaviors. ME and WA behaviors show a lower payoff w.r.t. cooperative nodes, thus indicating how they are always detected: while the cooperative nodes make errors with a certain probability, ME and WA send always scarce services and the model can easily detect them. Similarly, the OOA attack is discarded until an

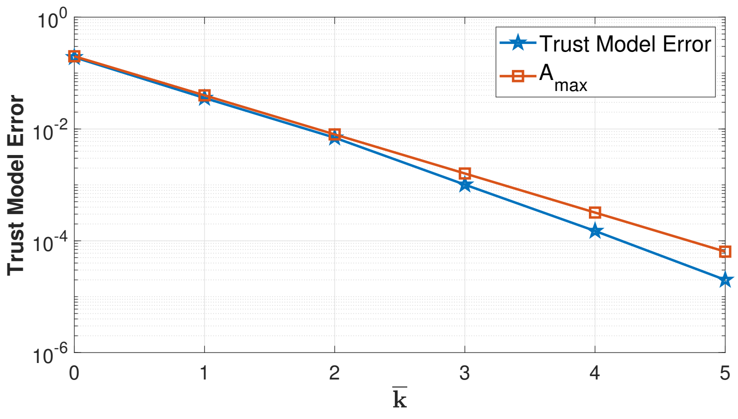

Figure 7.3: Trust model error for different values of $\overline{k}$.

error probability of 0.5, because this behavior is similar to a condition of a percentage of error equal to 50%. This results in a greater payoff of the OOA nodes with an error probability greater than 0.5. Furthermore, the OSA is able to reach the best payoff in all the simulations sending scarce services in a percentage equal to the probability of error (e.g. for a probability of error equal to 0.2, the OSA is able to act maliciously for the 20% of its interactions). Moreover, OSA takes advantage of the admissible error set by the model sending for each requester a number of $\overline{k}$ bad services in the first interactions.

Figure 7.3 shows the trust model error in discarding cooperative nodes, considering a scenario with $N = 10^5$ cooperative nodes and a probability of error equal to 0.2. The orange line shows the computed value for the maximum admissible error A_{max} as referenced. The Figure shows how the value of $\overline{k}$ is essential to overcome the probability of error of the cooperative nodes and obtain an acceptable error of the trust algorithms. By increasing the value of $\overline{k}$, the model can decrease the number of discarded cooperative nodes, i.e. the trust model error, at the cost of increasing the vulnerability to attacks.

Furthermore, the importance of the parameter Δ^+ is illustrated in Figure 7.4. At the end of the simulation, i.e. after 100 transactions, the Figure shows how the worst error is when $\Delta^+ = 0$ since the cooperative nodes are all discarded and no errors are allowed. The minimum value that allows a maximum admissible error A_{max} equals to 0.001 is $\Delta^+ = \frac{p}{1-p}\Delta^-$: this value allows cooperative nodes to avoid being discarded due to errors and, at the same time, enable the trust model to identify quickly the attacks.

Finally, the last two sets of simulations are aimed to understand how malicious nodes can change their parameters in order to overcome the trust model. According to the previous simulations, the probability of error p is set equal to 0.2 and the simulations focus on an individual malicious node. Each malicious node tries to bypass the trust model in order to increase its own payoff at the expense of the cooperative

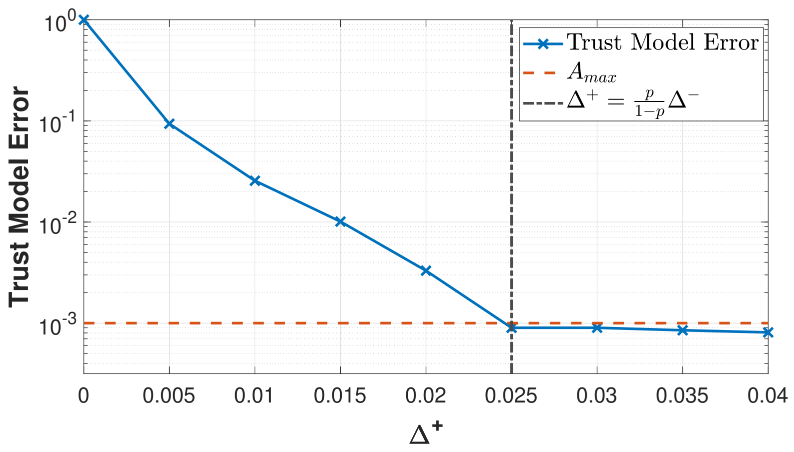

Figure 7.4: Model error for different values of Δ^+.

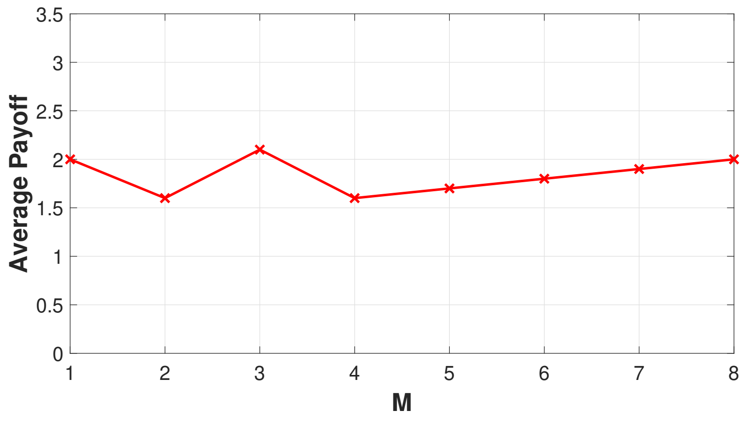

Figure 7.5: Average payoff for a OOA node for different values of M.

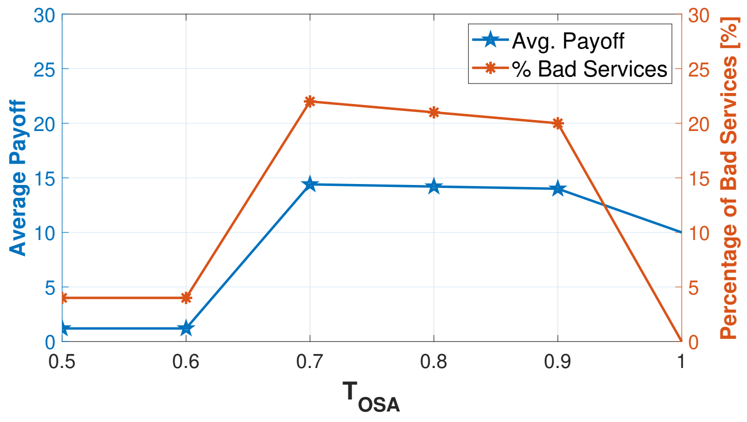

Figure 7.6: Average payoffs and percentage of scarce services for the OSA node at the OSA threshold variation.

nodes. ME attack has no way to modify its behavior and with a probability of error $p < 1$ it is always detected during the first transaction by the trust model. Concerning the WA attack, a node can change how often it re-enters the network, but since its behavior is similar to the ME, the node is always detected at the first malicious transaction. The worst-case scenario is when a node implementing the WA attack re-enters the network after each malicious interaction, and the WA attack can be never detected. However, the high cost of leaving and re-entering the network with a different identity limits this behavior.

Analyzing the OOA behavior, a node has two choices available: which state it is used for the first transactions and the duration of each state, i.e. the value of M. As stated in the previous section, the best choice for a node is to start with a benevolent behavior in order to increase its trust value. Figure 7.5 illustrates then the average payoff of a node implementing the OOA attack with different values of M. With an error probability equal to 0.2 and A_{max} sets to 0.001, the model can allow a number of $\overline{k} = 3$ consecutive error starting from the first interaction. This means that a node performing OOA will be detected as malicious after 4 malicious interactions. The average payoff is then tied by the number of benevolent interactions the node performs when it is in the ON state, so that the average payoff increases for $M > \overline{k}$. Before this condition, the average payoff shows an oscillatory behavior due to the variation in the number of cooperative transactions.

At last, the behavior of the OSA node is described in Figure 7.6. The model threshold T_{th} is set according to Equation 7.24 with a value near 0.7, because of $\Delta^- = 0.1$. The Figure shows how the best value for T_{OSA} is equal to T_{th}, where the node can perform the highest number of bad services, and thus its average payoff is maximum. We can also note how the percentage of bad services the OSA behavior is able to deliver is higher than the error probability of a node ($p = 0.2$): this is due to the ability of OSA to exploit the tolerance margin due to the maximum admissible error by the algorithm, A_{max}. With $T_{OSA} < T_{th}$ the node is immediately detected and its payoff is lower, while with $T_{OSA} > T_{th}$, the node does not fully exploit malicious opportunities and then it can not achieve the maximum payoff.

Chapter 8

Conclusions

In this thesis, we have proposed an exhaustive analysis of the trustworthiness management in the IoT and SIoT. At first, we have illustrated a query generation model that can be used to analyze the performance of search and discovery mechanisms in the SIoT. To define the model, we have generated a dataset, which is based on real IoT objects available in the city of Santander and makes use of people mobility models. The dataset and the resulting social network are made available to the research community in order to test several SIoT management algorithms. Moreover, we have analyzed the possible types of attacks that nodes can implement to disrupt an IoT system and then proposed a trust management model based on a Machine Learning algorithm for a SIoT scenario. However, through an analysis of the literature, we have identified two important deficiencies of the scenarios used to test TMSs. The first concerns errors in providing services: a node provides the wrong service due to malicious behaviours or malfunctions and poor accuracy (errors in service providing). Moreover, the requester node usually is not able to perfectly evaluate the received service; thus, benevolent providers could be poorly evaluated (errors in service requesting). We have shown how these essential assumptions must be considered in the scenario used to test a trust model and how any complex algorithm is necessary otherwise. Therefore, to solve these deficiencies, we have defined the problem of feedback evaluation in the IoT and proposed different metrics to evaluate a reference value that should be used to rate the service received by the providers, with or without the presence of errors. Finally, we have modelled the interactions among nodes in the IoT following a binary trust game to study how trust can arise between them. In particular, we have analyzed the interactions between a service requester, which acts as the trustor, and a service provider, the trustee. Based on this model, we have proposed guidelines that can be used to design trust management algorithms.

Bibliography

[1] A. Al-Fuqaha et al. "Internet of things: A survey on enabling technologies, protocols, and applications". In: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 2347–2376.

[2] K. Li et al. "Incorporating social interaction into three-party game towards privacy protection in IoT". in: *Computer Networks* 150 (2019), pp. 90–101.

[3] L. Atzori et al. "The social internet of things (siot)–when social networks meet the internet of things: Concept, architecture and network characterization". In: *Computer Networks* 56.16 (2012), pp. 3594–3608.

[4] D. Gambetta et al. "Can we trust trust". In: *Trust: Making and breaking cooperative relations* 13 (2000), pp. 213–237.

[5] W. Meng et al. "Towards Bayesian-based trust management for insider attacks in healthcare software-defined networks". In: *IEEE Transactions on Network and Service Management* 15.2 (2018), pp. 761–773.

[6] M. Nitti, R. Girau, and L. Atzori. "Trustworthiness management in the social internet of things". In: *IEEE Transactions on knowledge and data engineering* 26.5 (2014), pp. 1253–1266.

[7] H. Xia et al. "An efficient social-like semantic-aware service discovery mechanism for large-scale Internet of Things". In: *Computer Networks* 152 (2019), pp. 210–220.

[8] A. M. Ortiz et al. "The Cluster Between Internet of Things and Social Networks: Review and Research Challenges". In: *IEEE Internet of Things Journal* 1.3 (2014), pp. 206–215.

[9] L. Atzori et al. "The social internet of things (siot)–when social networks meet the internet of things: Concept, architecture and network characterization". In: *Computer networks* 56.16 (2012), pp. 3594–3608.

[10] R. Chen, F. Bao, and J. Guo. "Trust-based service management for social internet of things systems". In: *IEEE transactions on dependable and secure computing* 13.6 (2015), pp. 684–696.

[11] K. Zhao and L. Pan. "A machine learning based trust evaluation framework for online social networks". In: *2014 IEEE 13th International Conference on*

Trust, Security and Privacy in Computing and Communications. IEEE. 2014, pp. 69–74.

[12] A. M. Kowshalya and M. Valarmathi. "Trust management for reliable decision making among social objects in the Social Internet of Things". In: *IET Networks* 6.4 (2017), pp. 75–80.

[13] B. Jafarian, N. Yazdani, and M. S. Haghighi. "Discriminative-Aware Trust Management for Social Internet of Things". In: *Computer Networks* (2020), p. 107254.

[14] R. Chen et al. "Trust-based service management for mobile cloud IoT systems". In: *IEEE transactions on network and service management* 16.1 (2018), pp. 246–263.

[15] B. Gong, Y. Zhang, and Y. Wang. "A remote attestation mechanism for the sensing layer nodes of the Internet of Things". In: *Future Generation Computer Systems* 78 (2018), pp. 867–886.

[16] P. De Meo et al. "A reputation framework to share resources into iot-based environments". In: *2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC).* IEEE. 2017, pp. 513–518.

[17] W. Li, H. Song, and F. Zeng. "Policy-based secure and trustworthy sensing for internet of things in smart cities". In: *IEEE Internet of Things Journal* 5.2 (2017), pp. 716–723.

[18] H. Al-Hamadi and R. Chen. "Trust-based decision making for health IoT systems". In: *IEEE Internet of Things Journal* 4.5 (2017), pp. 1408–1419.

[19] B. Li et al. "PHAT: A preference and honesty aware trust model for web services". In: *IEEE Transactions on network and service management* 11.3 (2014), pp. 363–375.

[20] Z. Chen et al. "A scheme of access service recommendation for the Social Internet of Things". In: *International Journal of Communication Systems* 29.4 (2016), pp. 694–706.

[21] N. B. Truong et al. "Toward a trust evaluation mechanism in the social internet of things". In: *Sensors* 17.6 (2017), p. 1346.

[22] D. Airehrour, J. A. Gutierrez, and S. K. Ray. "SecTrust-RPL: A secure trust-aware RPL routing protocol for Internet of Things". In: *Future Generation Computer Systems* 93 (2019), pp. 860–876.

[23] X. Fan et al. "Decentralized Trust Management: Risk Analysis and Trust Aggregation". In: *ACM Computing Surveys (CSUR)* 53.1 (2020), pp. 1–33.

[24] A. Altaf et al. "Trust models of internet of smart things: A survey, open issues, and future directions". In: *Journal of Network and Computer Applications* 137 (2019), pp. 93–111.

[25] M. A. Azad et al. “Decentralized self-enforcing trust management system for social Internet of Things”. In: *IEEE Internet of Things Journal* 7.4 (2020), pp. 2690–2703.

[26] D. Wang et al. “Towards robust and effective trust management for security: A survey”. In: *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE. 2014, pp. 511–518.

[27] J. Caminha, A. Perkusich, and M. Perkusich. “A smart trust management method to detect on-off attacks in the internet of things”. In: *Security and Communication Networks* 2018 (2018).

[28] A. Jøsang and J. Golbeck. “Challenges for robust trust and reputation systems”. In: *Proceedings of the 5th International Workshop on Security and Trust Management (SMT 2009), Saint Malo, France*. Vol. 5. 9. Citeseer. 2009.

[29] M. Rashmi and C. V. Raj. “A review on trust models of social Internet of Things”. In: *Emerging Research in Electronics, Computer Science and Technology*. Springer, 2019, pp. 203–209.

[30] J. Guo, R. Chen, and J. J. Tsai. “A survey of trust computation models for service management in internet of things systems”. In: *Computer Communications* 97 (2017), pp. 1–14.

[31] R. Chen and J. Guo. “Dynamic hierarchical trust management of mobile groups and its application to misbehaving node detection”. In: *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*. IEEE. 2014, pp. 49–56.

[32] K. Zaidi et al. “Host-based intrusion detection for vanets: a statistical approach to rogue node detection”. In: *IEEE transactions on vehicular technology* 65.8 (2015), pp. 6703–6714.

[33] R. Chen, J. Guo, and F. Bao. “Trust management for SOA-based IoT and its application to service composition”. In: *IEEE Transactions on Services Computing* 9.3 (2014), pp. 482–495.

[34] R. Girau, S. Martis, and L. Atzori. “Lysis: A platform for IoT distributed applications over socially connected objects”. In: *IEEE Internet of Things Journal* 4.1 (2017), pp. 40–51.

[35] M. Nitti et al. “The Virtual Object as a Major Element of the Internet of Things: a Survey”. In: *IEEE Communications Surveys Tutorials* PP.99 (2015), pp. 1–1.

[36] I. Grønbæk. “Architecture for the Internet of Things (IoT): API and interconnect”. In: *Sensor Technologies and Applications, 2008. SENSORCOMM’08. Second International Conference on*. IEEE. 2008, pp. 802–807.

[37] Q. Z. Sheng et al. *Service-Oriented Computing: 14th International Conference, ICSOC 2016, Banff, AB, Canada, October 10-13, 2016, Proceedings.* Vol. 9936. Springer, 2016.

[38] V. Issarny et al. “Revisiting service-oriented architecture for the IoT: a middleware perspective”. In: *International conference on service-oriented computing.* Springer. 2016, pp. 3–17.

[39] R. Girau, S. Martis, and L. Atzori. “Lysis: A platform for IoT distributed applications over socially connected objects”. In: *IEEE Internet of Things Journal* 4.1 (2016), pp. 40–51.

[40] I. Sason and S. Verdu. “f-divergence Inequalities”. In: *IEEE Transactions on Information Theory* 62.11 (2016), pp. 5973–6006.

[41] *FIWARE Data Models.* 2018.

[42] L. Sanchez et al. “SmartSantander: IoT experimentation over a smart city testbed”. In: *Computer Networks* 61 (2014), pp. 217–238.

[43] TUS-Santander. *TUS Santander.* 2018.

[44] G. W. Index. *GWI Social Q1 2017.* 2017.

[45] S. Kosta, A. Mei, and J. Stefa. “Small world in motion (SWIM): Modeling communities in ad-hoc mobile networking”. In: *Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on.* IEEE. 2010, pp. 1–9.

[46] J. Leguay et al. *CRAWDAD trace upmc/content/imote/cambridge (v. 2006–11–17).* 2006.

[47] R. Girau, S. Martis, and L. Atzori. “Neighbor discovery algorithms for friendship establishment in the social Internet of Things”. In: *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT).* IEEE. 2016, pp. 165–170.

[48] C. Marche, L. Atzori, and M. Nitti. “A dataset for performance analysis of the social internet of things”. In: *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC).* IEEE. 2018, pp. 1–5.

[49] C. Marche et al. “Navigability in social networks of objects: The importance of friendship type and nodes’ distance”. In: *2017 IEEE Globecom Workshops (GC Wkshps).* IEEE. 2017, pp. 1–6.

[50] L. Zhu et al. “The Barabasi and Albert scale-free network model”. In: *Journal of Intelligent & Fuzzy Systems* 35.1 (2018), pp. 123–132.

[51] Y. Ma and G. Guo. *Support vector machines applications.* Vol. 649. Springer, 2014.

[52] J. Xu et al. “New incremental learning algorithm with support vector machines”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49.11 (2018), pp. 2230–2241.

[53] S. Lee and C.-H. Jun. “Fast incremental learning of logistic model tree using least angle regression”. In: *Expert Systems with Applications* 97 (2018), pp. 137–145.

[54] P. Reiner and B. M. Wilamowski. “Efficient incremental construction of RBF networks using quasi-gradient method”. In: *Neurocomputing* 150 (2015), pp. 349–356.

[55] H. Xia et al. “Trustworthiness inference framework in the social Internet of Things: A context-aware approach”. In: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE. 2019, pp. 838–846.

[56] C. Boudagdigue et al. “Trust management in industrial Internet of Things”. In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 3667–3682.

[57] M. S. Abdalzaher and O. Muta. “A game-theoretic approach for enhancing security and data trustworthiness in IoT applications”. In: *IEEE Internet of Things Journal* 7.11 (2020), pp. 11250–11261.

[58] S. Suhail et al. “Data trustworthiness in iot”. In: *2018 International Conference on Information Networking (ICOIN)*. IEEE. 2018, pp. 414–419.

[59] G. Chen et al. “An adaptive trust model based on recommendation filtering algorithm for the Internet of Things systems”. In: *Computer Networks* 190 (2021), p. 107952.

[60] R. Su et al. “PDTM: Phase-based dynamic trust management for Internet of things”. In: *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE. 2021, pp. 1–7.

[61] Z. Lv et al. “Trustworthiness in industrial IoT systems based on artificial intelligence”. In: *IEEE Transactions on Industrial Informatics* 17.2 (2020), pp. 1496–1504.

[62] Z. Ma, L. Liu, and W. Meng. “Towards multiple-mix-attack detection via consensus-based trust management in IoT networks”. In: *Computers & Security* 96 (2020), p. 101898.

[63] M. Salimitari et al. “A prospect theoretic approach for trust management in IoT networks under manipulation attacks”. In: *ACM Transactions on Sensor Networks (TOSN)* 16.3 (2020), pp. 1–26.

[64] A. A. Adewuyi et al. “CTRUST: A dynamic trust model for collaborative applications in the Internet of Things”. In: *IEEE Internet of Things Journal* 6.3 (2019), pp. 5432–5445.

[65] C. V. L. Mendoza and J. H. Kleinschmidt. “A distributed trust management mechanism for the Internet of things using a multi-service approach”. In: *Wireless Personal Communications* 103.3 (2018), pp. 2501–2513.

[66] U. Jayasinghe et al. “Machine learning based trust computational model for IoT services”. In: *IEEE Transactions on Sustainable Computing* 4.1 (2018), pp. 39–52.

[67] L. Militano et al. “NB-IoT for D2D-enhanced content uploading with social trustworthiness in 5G systems”. In: *Future Internet* 9.3 (2017), p. 31.