Distributed on-line network monitoring for trust assessment

Jorge López

THESE DE DOCTORAT
de
Universidad Galileo
et de
DE L'UNIVERSITE PARIS-SACLAY,
préparée à Télécom SudParis

ÉCOLE DOCTORALE N° 580
Intitulé complet de l'école doctorale

Spécialité de doctorat : Informatique

Par

M Jorge López

Monitorage En-Ligne et Distribué de Réseaux pour L'Évaluation de la Confiance

Thèse présentée et soutenue à Évry, le 2 décembre 2015 :

Composition du Jury :

M. L. Denoyer	Professeur	Université Pierre et Marie Curie	Président
Mme N. Yevtushenko	Professeure	Tomsk State University	Rapporteuse
M. A. Mellouk	Professeur	Université Paris-Est Creteil	Rapporteur
Mme A. Cavalli	Professeure	Télécom SudParis	Examinatrice
M. A. Silva	Chercheur	Bell Labs Alcatel-Lucent	Examinateur
M. F. Lalanne	Chercheur	Inria Chile	Examinateur
M. S. Maag	Professeur	Télécom SudParis	Directeur de thèse
M. G. Morales	Professeur	Université Galileo	Co-directeur de thèse

"Happy life with the machines, scattered around the room... Look what they made, they made it for me! Happy technology!"

Joel Zimmerman & Chris James

Abstract

Distributed On-line Network Monitoring for Trust Assessment

by Jorge López

Collaborative systems are growing in use and in popularity. The need to boost the methods concerning the interoperability is growing as well; therefore, *trustworthy* interactions of the different systems are a priority. Trust as a computer science concept has been studied in the recent years. Nevertheless, in the literature, very little focus is placed on how to assess the correctness of the interactions between the entities; even if most approaches rely on the estimation of trust based on the accumulated measures of these values. To broadly determine the correctness of interactions without targeting a specific domain or application, an approach using Distributed On-line Network Monitoring (DONM) was proposed. Furthermore, a prototype tool-set was developed to automatically test the *trust properties*. DONM is a form passive testing; it analyzes systems' responses and test the correctness of the interactions via network traces. Since it relies on the stated *properties* to test, a novel approach was proposed to automatically extract relevant properties to test. Our approach deeply relies on the operation of On-line Monitoring Systems. That is the reason why we propose new methods to enhance the state of the art techniques to: a) efficiently evaluate properties in $O(n)$ time complexity using an Extended Finite State Automata (EFSA) auxiliary data structure; and b) to expand the language expressiveness to properly express the constraints of such systems, such as, *timeouts* in order to avoid resource starvation. Finally, using the evaluation of the entities' interactions provided by our approach, trust management engines will help *trustors* to decide with whom and how to interact with other users or applications. We propose a new framework that is flexible for any domain, allowing trustors to define the *trust features* used to evaluate trustees in different contexts. Furthermore, with the evaluations of the trust features, we propose a trust model which achieves close-to-human inference of the trust assessment, by using a machine learning based trust model, namely solving a multi-class classification problem using Support Vector Machines (SVM). Using the SVM-based trust model, experiments were performed with simulated trust features to estimate *trust level*; an accuracy of more than 96% was achieved.

Publications

Journal articles

- Jorge López, Stephane Maag, Gerardo Morales, "Behavior Evaluation for Trust Management based on Formal Distributed Network Monitoring", *World Wide Web*, DOI: 10.1007/s11280-015-0324-6, P: 1-19, Rank A, IF: 1.623/SJR 0.754, Included in Springer High Impact Factor Journals in Computer Science, 2015

- Xiaoping Che, Jorge López, Stephane Maag, Gerardo Morales, "Testing trust properties using a formal distributed network monitoring approach", *Annals of Telecommunications*, DOI: 10.1007/s12243-014-0454-3, Volume 70, Issue 3-4, P: 95 - 105, IF: 0.408, 2015

- Jorge López, Stephane Maag, "Distributed On-line Network Monitoring for Trust Management", Proceedings of the Russian Physics Journal, Accepted to be published, 2015

- Jorge López, Stephane Maag, Gerardo Morales, "Scalable Evaluation of Distributed On-line Network Monitoring for Behavioral Feedback in Trust Management", *Proceedings of the Institute for System Programming of the Russian Academy of Sciences*, DOI: 10.15514/ ISPRAS-2014-26(6)-11, Volume 26, Issue 6, P: 125 -140, 2014

Conferences & workshops

- Jorge López, Stephane Maag, **??**"Towards a Generic Trust Management Framework using a Machine-Learning-Based Trust Model", *The 14th IEEE Internationtal Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-15)*, DOI: 10.1109/Trustcom.2015.528, P:1343–1348 August, Helsinki, Finland, Rank A, 2015

- Jorge López, Stephane Maag, Gerardo Morales, "TEAR: a Multi-purpose Formal Language Specification for TEsting At Runtime", *The First International Workshop on Security Testing And Monitoring (STAM)*, DOI: 10.1109/ARES.2015.90, P:727–734, August, Toulouse, France, 2015

- Jorge López, Stephane Maag, Cecilia Saint-Pierre, Javier Bustos, Ana Cavalli, "Process Mining for Trust Monitoring", *29th IEEE International Conference on Advanced Information Networking and Applications (AINA) Workshops*, DOI: 10.1109/WAINA.2015.71, P: 605-610, Gwangju, Korea, Rank B, 2015

- Xiaoping Che, Stephane Maag, Jorge López, Ana Cavalli, "Testing Network Protocols: Formally, at runtime and on line", *26th IEEE International Conference on Software Engineering and Knowledge Engineering (SEKE)*, P: 90-93, Vancouver, Canada, Rank B, 2014

- Xiaoping Che, Jorge López, Stephane Maag, "Online Testing: A Passive Approach for Protocols", *Revised Selected Papers of the 9th INSTICC International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, DOI: 10.1007/978-3-319-27218-4_6, P: 79–92, Lisbon, Portugal, Rank B, 2015

- Xiaoping Che, Stephane Maag, Jorge López, "A Formal Online Monitoring approach to Test Network Protocols", *Eighth International Conference on Advanced Engineering Computing and Applications in Sciences*, Rome, Italy, 2014

- Jorge López, Xiaoping Che, Stephane Maag, "An Online Passive Testing Approach for Communication Protocols", *9th INSTICC International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, DOI: 10.5220/0004885 501360143, P: 136-143, Lisbon, Portugal, Rank B, 2015

- Jorge López, Xiaoping Che, Stephane Maag, Gerardo Morales, "A Distributed Monitoring Approach for Trust Assessment Based on Formal Testing", *28th IEEE International Conference on Advanced Information Networking and Applications (AINA) Workshops*, DOI: 10.1109/WAINA.2014.114, P: 702-707, Victoria, Canada, Rank B, 2014

- Jorge López, Stephane Maag, Gerardo Morales, "A formal distributed network monitoring approach for enhancing trust management systems", *Fifth ACM International Conference on Management of Emergent Digital EcoSystems (MEDES)*, DOI: 10.1145/2536146.253 6160, P: 76-83, Luxembourg, Luxembourg, 2013

Acknowledgements

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Even though the future seems far away, it is actually beginning right now." — *Mattie Stepanek.*

The continuous expansion of computer network communications has encouraged and entitled users and developers to create new ways to interact. In general, new applications and systems can collaborate to provide a single composite service to the end user. In addition to that, the interactions between the systems can become complex. Also, sensitive transactions and critical operations are performed. For instance, external payment gateways used by companies to process payments or external data storage using cloud services. Furthermore, collaborative systems might deeply rely on peer systems or users' correct data from these interactions to operate correctly. Many examples can be listed in that category, some can be: a) crowd-sourcing applications [Yu et al., 2012], that without the correct information, their operation becomes worthless, and b) for Mobile Ad-Hoc Networks (MANets) [Mellouk et al., 2014], to properly route data with other nodes. Given the previously mentioned facts, the need to guarantee the correctness of those interactions is of great priority. Deciding with whom and how to interact with the different entities is an important task that the systems and users need to accurately be able to perform. A Human cognitive concept of this is *trust*.

The concepts of trust have been brought to computer science. The systems need to interact with users and with other applications. The decision regarding with who and how to interact with other users or applications depends on each application or system. Probably, the most basic notion of trust is that, trust is a relationship; a relationship between an entity which expects a behavior, the *trustor*, and the entity performing the action, the *trustee*. One of the first works that introduced trust as a computer science concept is the one realized in [Marsh, 1994]. In this work the authors assign a range of

trust and distrust going from -1 to 1; -1 representing full distrust, and 1 representing blind trust. While these fuzzy logic values are very intuitive for the characterization of trust as a computer science concept, deciding when to consider an entity trustworthy or not is done with a simple threshold. Nevertheless, this is a first interesting way of formalizing trust concepts. Instead of using a binary approach of complete trust or distrust, these values are considered as fuzzy logic values. Fuzzy logic variables or values handle the concept of partial truth. For example to describe "how black is a gray color", you can use these type of values; a 0.1 value will indicate a pale gray. The same concept applies for a trust value of 0.95, this means a strong trust from the trustor to the trustee. Many systems have adopted this methodology as well. For example, in the works [Ray and Chakraborty, 2004; Toumi et al., 2012], the functions of trust associate the trust and distrust values on the same ranges from -1 to 1.

There are many definitions of trust in the literature, but the one we adopt is the one commonly accepted, widespread and defined in [Grandison and Sloman, 2000] as: "the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context".

From the definition we note that it involves several fundamental factors:

- The trustor trusts an entity called the trustee. This implies a mechanism for identification or authentication

- Belief is a subjective concept for the trustor. It is regarding an expectation on the future, but, it could be influenced by the past events with the trustee, i.e., experience

- The trustor expects **behaviors** from the trustee. Behaviors considered dependably, secure and reliable

- A trustee might have different levels of trust for different contexts, that is, situations

The statements presented above are essential in the development of trust management systems. From those points, many researchers have created their approaches of trust. For instance, several trust management systems use security policies and authentication in order to provide the concept of trust. In these types of systems, the entity called trustee is related to an authentication mechanism. The security policy rules express the actions allowed for each trustee. This is called "hard trust" because the actions can only be permitted or denied, i.e., *who can do what*. The action on the security policy implies a context. For example, trust management systems as PolicyMaker[Blaze et al., 1996],

KeyNote[Blaze et al., 1999], REFEREE[Chu et al., 1997], and SD3[Jim, 2001] use policy languages and authentication to provide the concept of trust. These works were some of the pioneers in trust management systems.

"Soft trust" management systems, on the other hand are trust management systems that are based on features as experience, reputation, risk, and others. For this purpose, the evaluation of the trustee's behaviors are added to assess the trustee experience. This approach is somewhat closer to the human cognitive notion of trust. Nevertheless, most of the works dedicated to trust estimations utilize only local observations. A crucial aspect is that those trust management engines assume that the behavior evaluation is a given system capability, and do not take into consideration how to perform these evaluations. Additionally, to the best of our knowledge, no comprehensive methods to evaluate behaviors are found in the trust literature. Likewise, no formal approaches have been defined considering several points of observation. Normally, approaches consider the local interactions between the trustor and the trustee. If beyond the local observations there could exist untrustworthy behaviors, a local-only perspective will fail to detect them; we expose some of these behaviors in this work. We tackle the previously mentioned issue, and as the first contribution, we propose to provide behavioral evaluation, testing if the interactions are trustworthy or not, based on the inspection and the correlation of the network packets at different points of observation.

We aim at providing trust information in a comprehensive way so that, any generic framework can use the information about these behaviors and incorporate them into the trust estimation process. It is our point of view that trust systems will benefit from different techniques inputs. That is the reason why in this first contribution we do not aim to provide another approach how to assess trust, but, rather providing a comprehensive way to perform behavior evaluation of the interactions. Given this motivation, we present a distributed on-line monitoring tool-set to provide feedback regarding the interactions between the entities, and a formal language to express trust properties. It is extremely important to note that, we consider that any entity which needs to trust another one, commonly, are systems that communicate through network protocols. Therefore, our approach can be said to be deeply comprehensive for any system who's traffic can be inspected. This can be considered as the lower layer of abstraction if considering trust as a layered approach.

An important remark to clarify our contribution is that, in order to guarantee the *correct* system interactions, the previously mentioned approach was proposed. This approach is formulated as the well-known and widely-adopted approach of guaranteeing correctness of given invariants (named throughout this thesis trust properties or simply properties) as employed in [Bayse et al., 2005]. Even if other different approaches to guarantee the

correctness of the interactions could be considered, those are left out of the scope of this thesis.

Regarding our first contribution, the main aspects which can be highlighted are the following:

- Defining a formal model to express trust properties in computer communication networks

- Proposing a distributed monitoring approach to analyze trust properties in distributed networks

- Developing a set of tools for performing on-line distributed network monitoring for evaluating behaviors in order to provide trust verdicts

- Successfully applying our methodology and tools to a real industrial DNS use case scenario to evaluate the trustworthiness of the servers responding the queries from DNS resolvers. Furthermore, to be able to correctly detect untrustworthy replies in real time.

One important characteristic which is always desirable in trust systems is to have the trust information as fast as possible, since guaranteeing sensitive or critical operations can be done timely; that is the reason why our proposed mechanism focuses on monitoring the systems on real time. Being on-line network monitoring so important for our approach, as the second contribution of our work is to propose enhancements to on-line monitoring systems, namely:

- Identifying the areas of opportunity and motivation to enhance our on-line network monitoring system, the one exposed as the first contribution

- Proposing a scalable evaluation method for any on-line network monitoring system, by using an auxiliary model, an extended finite state automata (EFSA), and as well as other known methods to reduce the time complexity of the evaluation algorithm

- The modification of the language presented in the first approach to ease the usage of our tools to avoid issues as resource starvation

- Consolidating our enhancements, using the newly proposed language with the scalable algorithm to construct the corresponding EFSA from a property stated in the language

Once having a solid and enhanced approach on how to evaluate behaviors to provide trust verdicts, we noticed that, the traditional methods to find trust properties can be laborious and the process can be highly time consuming. Given that motivation, in this work we propose a novel approach to automatically generate trust properties by employing techniques from the Process Mining (PM) research domain. In an intuitive way, communicating systems and their intrinsic network protocols are defined as sequence network packets or messages. Nonetheless, they can also be seen as sequences of structured activities, queries and responses that are executed by entities (e.g., clients and servers), with the objective of exchanging some valuable information. From this point of view, we propose to model the system's behavior as a process flow and use the PM techniques to analyze them. We aim at comparing formal models of the protocol specification and the models obtained from the PM techniques. Using these *comparisons*, we propose a new algorithm to generate the suitable set of trust properties for any *fault* found. The automatic trust property generation is our third contribution in this work.

One of the major challenges in the trust domain in computer science is the lack of a generic framework. It can be appreciated in the surveys shown in [Artz and Gil, 2007] and [Momani and Challa, 2010]. Most frameworks force users to adopt a policy language, or a fixed set of parameters, predefined by the trust management engine. Moreover, they have the rigid approach of having a single trust threshold to distinguish between trusted and untrusted entities. Our goal is to allow any existing software implementations, systems, and devices to incorporate the trust concepts and use trust management engines. One key fact that is usually overlooked is that, there is a huge range of devices with very different processing capabilities. The interaction with such engines, modules, and systems must be as less intrusive (transparent) and flexible as possible. The flexibility concept is inherent to the trust nature; each trustor decides "how" to trust the trustees.

After our proposed approach to comprehensively provide feedback regarding the entities' interactions, we noticed that no trust management engines or frameworks are capable of easily incorporating external feedback, nor adding new or extending trust properties to monitor a particular behavior. In this work, as the fourth contribution we propose a flexible an extensible framework by employing a RESTful web-service-based architecture to exchange the trust feedback and to get the trust information from the trust management engine. Furthermore, we entitle the trustor applications to define their own trust contexts by specifying the trust features related to the context, and how to measure them. In fact, in the literature, the capability of defining the trust contexts in the trust management framework during execution time is not considered. It is our belief that this contribution can encourage existing systems to incorporate trust concepts at a large

scale. Additionally, we discuss the architecture, design, and proposed inputs for the associated trust model.

When proposing an extensible and flexible trust management framework, the associated trust model needs to be capable of incorporating various different inputs, indiscriminately. With all the inputs, the trust model needs to determine to which "*trust zone*" the entity belongs to, considering that the feature space for the entities' is evaluated differently in different zones. In order to obtain this, the fifth contribution of this work is to present a trust model which is flexible and capable of accurately determining if the values of the inputs can yield trustworthy interactions. Commonly, trust models are based on a linear combination of inputs such as, knowledge, experience, and reputation. The associated weights of this linear combination determine how important each parameter is. In contrast, the concept of trust is much more complex. For instance, if a trust feature which is normally desired to have a high value is low, but the entity has a rare high combination of two other less important parameters for which it should be trusted, a simple linear combination is not capable to correctly model and express this set of constraints. It is our vision that trust management systems must have their criteria as close as we humans have in regards to trust. Likewise, trust models use a threshold to determine whether the entity is trusted or not, which logically implies that entities can only be binary classified. It is desirable to have multiple classes, in that way, the trust level of the entities can be accomplished in a finer manner. To achieve the close-to-human inference and a finner classification, we propose the use of machine learning techniques, namely solving a multi-class classification problem where the classes are: trustworthy, neutrally trusted, and untrustworthy; the inputs of the algorithm are the different trust features and the labels, i.e., evaluations of the trust level given the trust features. We present the trust model, and propose to solve it using Support Vector Machines (SVM) optionally using a Gaussian Kernel. We experiment our approach by simulating values from different experiences and their associated trust levels. The results are promising, they show high accuracy for determining the trust level that an entity has. With an accurate trust level assessment, entities can decide how to interact with their peers. For example a "neutrally trusted" entity, will be allowed to execute a limited set of operations, while a "trustworthy" entity will be allowed to execute the complete set of available operations.

The remainder of this thesis is organized as follows, in Chapter 2 we present the state of the art regarding trust models and frameworks, on-line network monitoring systems, and process mining. The Chapter 3, describes our generic approach to provide feedback of desired trust properties using the formal distributed on-line network monitoring approach. Then, in Chapter 4 we show our proposed method to automatically extract the trust properties to test. In Chapter 5 we show the enhancements proposed to on-line

network monitoring systems. Later, in Chapter 6 we detail a generic trust management framework, its architecture, design, and trust model to assess the trust level of the entities. Finally, in Chapter 7 we conclude and provide perspectives for our future work.

In general terms, in Chapter 3, known techniques for conformance and performance black box passive testing, using network packet captures were applied and extended to use on-line and distributed concepts to fit the trust management domain requirements and open problems of that domain. In the subsequent chapters, Chapter 4 and Chapter 5, solutions to enhance the previously mentioned state of the art techniques were proposed to ease the usage of them and to make them more scalable. Finally, in Chapter 6, open problems in the trust management domain were identified and solutions to them were proposed. Throughout the content of this thesis, we reference the associated contributions with respect to these general terms and to the particularities previously exposed in this chapter.

Chapter 2

Related Work

"If I have seen further it is by standing on the shoulders of Giants." — Sir Isaac Newton, Letter to Robert Hooke, February 5, 1676.

In the introductory chapter(Chapter 1), we showed an overview of what the work realized by this thesis is about. In this chapter, we depict the state of the art solutions regarding the related topics, in order to clearly distinguish the contributions of our work. More than that, based on the exposition of the different works, we show that the existing solutions leave some areas where some remaining open issues are encountered and a contribution is needed; those areas where contributions are needed are the ones we tackle throughout this thesis, making them our core contribution to the state of the art.

This Chapter is divided into two main related work sections. As stated in the introduction, our first contribution of this thesis is providing feedback on behavioral evaluation using distributed on-line network monitoring. Therefore, we first explore the related works on on-line network monitoring approaches. This is exposed in Section 2.1. Then, in Section 2.2 we expose general trust frameworks to compare how our work could fit into them and as well to compare our proposed contributions in the Chapter 6. Finally, to clearly understand our contribution, this chapter finishes with an overview of the remaining open issues given the exposed works, as well as our proposed solutions to address the remaining open issues. Further, we discuss the relation between those open issues and the specific content of the chapters of this thesis.

2.1 Distributed On-line Network Monitoring for behavioral feedback related work

In general terms, in the trust literature, there is little to no consideration on how to evaluate the behaviors; they do not consider how to obtain them, they simply assume they are present at the system. If considering trust as a layered and holistic approach, overlooking at the behavioral evaluation is highly undesirable. In general the objective of the behavioral evaluation is to provide an assessment of the outcome of an interaction. When two entities are interacting, the trustworthiness of those interactions in question needs to be rapidly evaluated. The desired behaviors of the interactions between the entities need to enforce, therefore, they behaviors need to be specified and proven to be the correct ones. The data extracted from the interactions need to be collected in some points of observation in such a way that the approach is useful for most cases. Also, the collected data need to be sufficient so that no untrustworthy behaviors can occur.

As stated before, while evaluating interactions, in order to provide a good behavioral evaluation, at least two points need to be stated: i) the feedback should be provided online to rapidly determine how to interact with the entities, and ii) we focus on network traffic inspection, based on the assumption that systems that interact mostly via network protocols. Based on these points we present some related work.
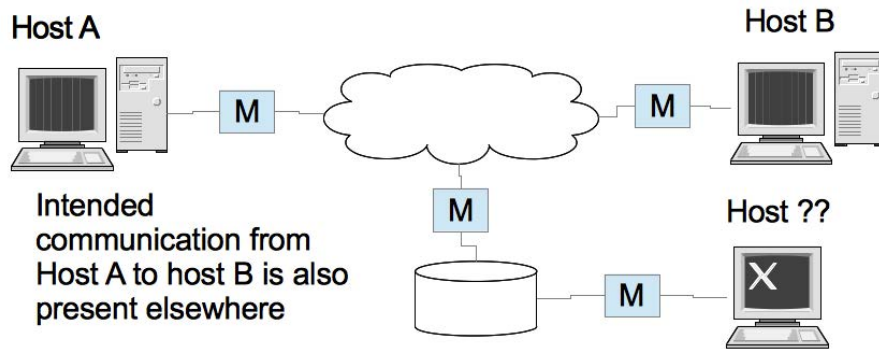
Figure 2.1: Data forwarded towards unknown destination

One might consider that a first approach to do this is to use well known Intrusion Detection System (IDS) software to evaluate the interactions. The IDS software meets both requirements stated above. The most used IDS software and the industry standard is Snort [Roesch et al., 1999]. Snort has many advantages when used as an IDS, nevertheless, trust properties can be by far more complicated. Briefly described, untrustworthy behaviors can be undetected, when the observations are coming from a single point of observation, for instance consider the case depicted in the Figure 2.1. In that figure, a message "M" that was intended only from "Host A" to "Host B" is detected at an external point of observation in a path which does not belong to the path to reach

"Host B". Further information of how we tackle this issue is described in Chapter 3, and specifically in Section 3.1. Furthermore, Snort utilizes regular expressions or signatures to describe the network packets. However, while constructing trust properties, granular access to the data is required given the network protocols' semantics, and also to make complex relationships between one observed network packet and the others. As an example, the DNS packet format allows for compression. In a nutshell, in order to avoid repeating some parts of a string in the DNS replies, DNS can encode a string with a pointer to previous data within the same packet. Let us assume the DNS string encoding as shown in Figure 2.2; the data is supposed to start in the bit 0 for the sake of simplicity. DNS string encoding compression will allow to use a further reference, for instance to [3]lor[ptr-to 4], in order to make [3]lor[16]telecom-sudparis[2]eu[0] without repeating the last part. If the regular expression language was used or any other signature mechanism, the semantics behind the DNS compression will make impossible for the signature to be correctly detected, for example when searching for a value that is not present completely, but referenced to another part of the packet. Our proposed approach is capable of processing the network protocols' semantic constructions and test those taking them into consideration.

0-7	8-15	16-23	24-31
3	w	w	w
16	t	e	l
e	c	o	m
-	s	u	d
p	a	r	i
s	2	e	u
0			

FIGURE 2.2: DNS string encoding

Additionally, different distributed approaches have been proposed for Intrusion Detection Systems(IDS). Several works like [Zargar et al., 2011], [Roschke et al., 2010] and [Lo et al., 2010] using different methodologies have exposed IDS that are collaborative or distributed as a solution to detect possible attacks, such as, to detect a distributed denial of service (DDoS). Collaborative or distributed IDSs can observe that different hosts located at different networks are sending network packets to the final common target. Thus, concluding that a distributed denial of service attack occurs. This correlation performed is without a doubt highly valuable. But, as trust properties are very complex, and as shown by some studies, such as [Dagon et al., 2008], untrustworthy replies can be produced without any security attacks. For example, let us assume a software implementation that does not suffer from any security breach. If the software has some implementation faults (software bug) that can only be observed when several rare circumstances come together and when this rare condition occurs, the implementation does not provide a trustworthy response, making the interaction untrustworthy. Our

approach enforces the trust properties regardless of the cause being a security breach or a mistake. Furthermore, the granularity of accessing the data is not considered. Likewise, the issue regarding the network protocols' semantic can be overlooked if the methodology does not consider granular access to the data.

As exposed by the previous works, IDS tools and methodologies cannot be directly applied for behavioral evaluation given the set of constraints of the trust properties. Mainly due to two reasons, i) The accuracy and granularity in the data inspection to consider the network protocols' semantics, and ii) the trustworthiness of the data can be unrelated to any intrusion attempt. There are other approaches that perform passive testing that can also be considered to be applied for evaluating the behavior of the entities interaction for the trust domain. We present some related work next.

Regarding passive testing techniques, the basic idea is to check for the correct functioning of the system without stimulating the system via any inputs. In that sense, passive testing has beyond any doubt a characteristic that is desirable when evaluating the trustworthiness of the interactions between the entities. Moreover, there exist works which also propose the use of network packets to inspect the correct functioning of the systems as our approach does. One interesting work is the one presented in [Cavalli et al., 2009], in that work, the authors experimented the correctness of the OSLR protocol [Clausen and Jacquet, 2003] by using network traces. This approach cannot be directly used for our purposes, the main reasons are that the approach only uses off-line network captures and that the observations are local only. The disadvantage with the approach being off-line is that untrustworthy interactions can go undetected until the captures are collected and the tests are executed. The delay in obtaining the trust verdicts can have a high impact for critical systems. For instance, when a sensitive operation depends on a previous (and also potentially sensitive) set of operations if obtaining the verdicts without a considerable delay, the last operation could be refused to the untrustworthy entity. On the other hand, if the time delay for obtaining verdicts is substantial, the untrustworthy entity could interact and execute potentially critical sensitve operations that ideally would have been refused.

Another interesting work from the passive testing domain is the one found in [Lee et al., 2006]. In this work, the authors tested various different network protocols. Indeed, making the approach as generic as possible so all network protocols can be tested with the same methodology is one of our aims. However, the approach presented in the previously mentioned work do not focus on some important aspects which are important for trust properties. First, the data collection is done in an off-line manner using *tcpdump*[1]; it has the same pitfalls as stated before for any off-line collection method. Second, in order

---

[1]http://www.tcpdump.org

to use their approach the generation of a formal model using an Extended Finite State Machine (EFSM) is necessary. It is known to be a hard task to derive a model. To begin with, a model requires without a doubt an expert in the protocol that will derive this model. Furthermore, when deriving a model, any mistake in the model specification can make the test results fruitless. Additionally, deriving any specification model can be a highly time consuming task. Finally, the approach only considers local interactions and certainly suffers from the same drawbacks as stated before.

An approach that tackled the issues how to finely describe the network packets, achieves complex relationships is the tool presented in [Che et al., 2012a; Lalanne and Maag, 2013]. In fact, our first approach takes the base syntax of the language from this approach. Further, given the lack of expressiveness of the language we change it in Section 5.4. The main shortcomings of this approach when evaluating behaviors for trust feedback are: the absence of on-line functionality, and the fact the approach uses a single point of observation.

Finally, in [Toumi et al., 2014], the authors also employ the concept of monitoring network traces to provide behavioral feedback for trust management. This approach has some shortcomings as well. First, it is not distributed and as stated before, untrustworthy behaviors can go undetected when considering a single point of observation. The underlying tool the authors use to do the network evaluation is the Montimage Monitoring Tool (MMT). The tool is very versatile and has many features, as stated from their website[2], the tool is capable of capturing and analyzing network traffic. It can be used to verify network functional properties, QoS and security properties and is composed of NetCapture probes and a NetOperator application that allow deploying a mixed distributed/centralized network monitoring solution. It incorporates machine-learning algorithms for the identification and statistical analysis of network flows and the detection of unexpected behavior and security flaws. Also, the language allows fine granularity by defining a function that determines how to extract any required value. Nonetheless, the MMT language formalism defines an "MMT Security Rule" as an if-then expression, as shown in definition 10 of [Mallouli et al., 2012]. Therefore, it cannot express several causalities and consequences. As for trust properties, in order to correctly expressing many trust properties, several implications are required. For instance, a trust property with the construction "if A is observed, then, B must be observed, then C should have occurred before A". As a result, the formalization using the MMT tool is not sufficient to express the desired trust properties.

Aside from the specific points that other passive testing approaches do not focus on as stated before, and that are necessary for our behavioral evaluation, we also note that

---

[2]http://www.montimage.com/en/products.php

all of them may not target and that we consider important. The first is that the on-line functioning needs to have great performance, due to the fact that the resources on the testers are limited, even if the testers are large clusters. The second is that passive testing techniques depend on the desired properties to enforce. The properties to enforce can be somehow difficult to discover and also a highly time-consuming process.

Our approach relies on the monitoring of correct functioning of systems, therefore, techniques for systems conformance can be compared. One of the most popular techniques for ensuring the correctness of systems is Model Based Testing [Holzmann, 2004]. Those techniques cannot always be directly applied to achieve our goal, the reason is that a model of the system under test has to be derived in advance, and furthermore a set of properties can be verified for corresponding violations. Typically, the system description is omitted when performing on-line monitoring/passive testing, hence, this issue is left out of the scope of this thesis. On the other hand, a formal specification of the system under test can be inferred by observing input/output traces and applying machine learning techniques [Irfan et al., 2013]. However, even if this is an interesting perspective for an alternative of our approach, it has not been developed nor tested. One of the major concerns about this possible approach is the time frame needed to detect any potential untrustworthy interaction. Moreover, even if the system has guaranteed correct functioning, the interaction with other peers might provide untrustworthy data on real-time execution. Also, as stated before, verifying the correctness of the specification model can be hard, even if this was not an entirely manual process. Therefore, in this thesis, we discuss how a number of properties can be still verified for a system under test when the formal system specification is absent.

To the best of our knowledge, none of the systems or frameworks distributed network monitoring with a flexible and expressive-enough language in order to provide trust information. All the previous works have a single point of observation, which implies that they only take into account the direct interactions between them and their peers or the feedback (reputation/recommendation) regarding the interactions between other peer entities. Nevertheless, the evaluation of interactions in order to construct the "experience" are based on the assessment of a single point of observation at the time. More than that, none of the works found in the literature tries to provide a generic approach to describe behaviors to provide feedback to any soft trust management engine, which is what our approach provides.

2.2 Trust management engines related work

Many trust management systems are based on authentication and security policies. The basic notion behind the concept of security policies consist on expressing a set of rules, based on three factors: subjects, resources, and actions. A subject is nothing more than an identified entity, i.e., the entity requesting the access; therefore, a way to uniquely identify the entities is needed. Resources are considered the object to which the access is given. For instance, data, service or system component. Finally, an action defines which type of access the subject requests on the resource. Verifying a security rule is considered to be guaranteeing that the subject requesting to perform the specified action on the requested resource is entitled to do it. If the request if found in the security rules the access if granted. After the security rule is verified, some concepts like *obligations* that the subject needs to fulfill can be stated. Additionally, there are many languages to express security policies. Nevertheless, the concepts of obligations and the different languages are out of the scope of this thesis.

For example, pioneering systems like PolicyMaker [Blaze et al., 1996], KeyNote [Blaze et al., 1999], REFEREE [Chu et al., 1997] and SD3 [Jim, 2001]; have presented trust management systems based on security policies. These type of systems work by exchanging credentials and applying the security policies to the authenticated entities. These systems have several advantages. However, they are not generic and force users to adopt certain authentication mechanisms and policy languages.

A more generic work is presented by TrustBuilder2 [Lee. et al., 2009]. In this work the authors presented a generic and extensible framework for authentication and security policy exchanges. The extensible features of the tool should allow interaction between their tool and information provided by an external module. For example, the trust information provided by analyzing the behaviors using distributed network monitoring. We agree philosophically on a generic and reconfigurable framework is the best path to make existing software to incorporate trust notions.

Other frameworks based on security policies have been presented. The framework called XeNA was presented in [Haidar et al., 2009]. In this work the authors propose to use eXtensible Access Control Markup Language (XACML) for access control management. Trust with security policies is considered "hard trust"; because of the rigid concept of only accepting certain entities and under certain situations.

In the literature, there are other works that use less rigid measures of trust. First of all, these measures are dynamic, this means that the trust level of an entity can increase or decrease along the time. Such approached are considered "soft trust". The most common parameter that systems that incorporate to achieve these are notions of

trust as an experience/reputation. Both of the concepts are related. On the one hand, experience is considered to be a cumulative measure of the evaluation of the previous interactions. On the other hand, reputation is considered to be the experience regarding a specific trustee estimated by an external entity and distributed. Many constraints can be encountered in regards who to trust with recommendations; in general terms this is also a problem of finding trustworthy recommendations.

For example, the work realized by [Movahedi et al., 2012], the authors applied these concepts to mobile ad-hoc networks (MANet). They monitor the behaviors of the neighbor nodes and keep local information of those interactions. Also they share this information to other nodes on the network. Using both local and remote information, they can calculate the trustworthiness of a node in the network.

Moreover, in [Toumi et al., 2012], the authors created TRUST-OrBAC. In this work the authors propose to add an experience module on top of the security policy engine. We also share the idea that trust systems should incorporate different approaches of trust. Another interesting work that combine different approaches to provide trust information is the work presented in [Grandison and Sloman, 2003]. The authors build a tool called SULTAN. This tool incorporates also notions of risk into its design, quantifying risk and actually applying these notions into their trust evaluation algorithms.

There are other works which use less rigid parameters of trust; less rigid parameters of trust are similar to the human cognitive concept of it [Falcone and Castelfranchi, 2001]. Researchers have naturally proposed solutions that "mimic" the human responses to trust. Recently, there is related work in the domain of supervised machine learning [Hauke et al., 2013]. One interesting work that applies machine learning techniques is found in [Song and Phoha, 2004]; in this work, they propose to use neural networks in order to provide a global reputation model using the distributed reputation evaluations. The global reputation is determined by the neural network's output unit, a two class classification in this case. Interesting results were shown by the global reputation model using neural networks. A trust model with a broader scope, not only considering reputation was introduced in [Wang and Vassileva, 2003]; in this work, the authors propose the use of a Bayesian-Network trust model to properly interact with trustworthy peers. In the latest, the trust was qualified as "satisfying" and "unsatisfying", respectively denoted as $T = 1$ and $T = 0$. Finally, in [Zhao and Pan, 2014] the authors presented a more flexible trust model based on Support Vector Machines. In the previous, the experiments were applied to social networks and the classification for this approach was "trust", and "distrust". Our trust model approach is also based on solving a classification problem, and we also apply SVM to solve it. Nevertheless, we propose a multi-class classification to have a fine distinction when interacting with the entities. In addition to

that, our approach aims to provide a generic framework to include any parameters and an approach to describe the behaviors observed through network monitoring.

To the best of our knowledge, none of the systems or frameworks in their particular methodologies and trust models target a generic framework. Furthermore, no trust models have considered a multi-class classification; likewise to achieve a higher prediction accuracy in an automated manner, we propose to solve the problem with SVM and provide an algorithm to determine the best suited parameters to achieve higher accuracy. All the exposed works provide little to no consideration of how the data gets collected by the frameworks, they all assume the data is present at the trust management engines somehow. In this thesis, we present our solutions for the previously stated open challenges.

2.3 Remaining Open Challenges

As exposed before, some open challenges are not precisely tackled with the existing solutions. Tackling some of these open issues is the contribution of this thesis. Therefore, the remaining open issues and main contributions to highlight in this thesis are the following:

- In the literature, there is little consideration on how to universally provide behavioral feedback for trust management. Under the assumption that systems that need to guarantee trustworthy interactions are systems connected to the network, we define a universal, granular, and on-line approach. This work is presented in the Chapter 3 of this thesis.

- Many approaches which rely on passive testing independently from the input sources, only test those properties which are explicitly declared. More often than not, finding which properties are sufficient to test can be somewhat challenging. Furthermore, to be sure that the properties are relevant (useful to test). In order to tackle this, we generate trust properties from observed *faults*, with the help of a model, using Process Mining techniques. This work is exposed in Chapter 4.

- Our approach to provide behavioral evaluation heavily relies on an on-line network monitoring system. We understood that the on-line monitoring systems have different constraints that are overlooked by many. First, we enhance the algorithms for the trust properties' evaluations given the on-line resources consumption constraints. This work is shown in Section 5.3.

- Many languages proposed by the on-line monitoring system do not consider the ease of usage for the end-users, and moreover, are not capable of expressing all what it is required. In order to ease the use of our formalism, we propose a language extension to incorporate inherent constraints of on-line network monitoring. This is shown in this thesis also in Chapter 5 in Section 5.4.

- Given the fact that no solutions in the state of the art consider the different devices' capabilities, we have proposed a framework, which can offload the trust assessment to specialized trust management engines. Moreover, by offloading the services, we consider allowing other sources to provide the feedback, trustor delegates. This work can be seen in Chapter 6, specifically in Section 6.2.

- Yet another remaining open issue tackled in Chapter 6 is the proposal of a trust model which is more flexible and has closer-to-human inference than the ones proposed by the existing solutions. This work is shown in Section 6.3.

Chapter 3

Behavioral Evaluation Feedback for Trust Management using Distributed Network Monitoring

"L'essentiel est invisible pour les yeux." — Renard / Antoine de Saint-Exupéry, Le Petit Prince.

At the beginning of the work of this thesis, we realized that for soft trust management engines which incorporated measures as experience, do not consider how to obtain the evaluation of the interactions. The behavioral evaluation is an issue that is either not addressed, assumed, omitted, or a particular strategy for a particular case is adopted as exposed in the related work. Given this motivation, we propose to provide behavioral feedback in a comprehensive manner, such that, any application can use it. Under the assumption that systems which need to trust other systems are entities connected through a computer network, we aim to guarantee the desired behaviors occur by monitoring the network protocol messages. To determine what it needs to be guaranteed, we express in a formal language the requirement, this formalism is what we describe as a **trust property**.

This Chapter is divided into four Sections. First, in Section 3.1 we present the basic notions of our approach, including the need to potentially consider different points of observation. Later, in Section 3.2 we propose the language used to describe the trust properties. In Section 3.4.4 we present the architecture, design, and algorithms for our prototype tool set used to test the trust properties on on-line network captures. In the last Section, we describe the experiments performed by using our formalism and tool set

in an industrial partner architecture, including the performance evaluation for our tool set as shown in Section 3.4.6.

In order to clarify the scope of the approach presented in this chapter, as mentioned before, a trust property can be considered an invariant, and methods for guaranteeing the fulfillment of these invariants is what this chapter proposes. There might be other methods to guarantee the correctness of the entities' interactions. For example, expressing "*test objectives*" with a partial specification model. This model could potentially be some Finite State Machine (FSM) model. Given the fact that the desired passive testing needs to be distributed, and therefore any input (message type) can appear at any state, most probably the FSM model would also be complete. Nonetheless, this interesting perspective is left out of the scope of this thesis. Furthermore, such form of testing behaviors is left for future works.

In this Chapter, we expose the work which can be partially found in the following publication list: [Che et al., 2015; López et al., 2014a, 2013, 2015a].

3.1 Untrustworthy behaviors detected through the use of distributed network monitoring

3.1.1 General Definitions of the Proposed Approach

Our main objectives are: i) to be able to detect untrustworthy behaviors of entities in a flexible manner, with a different perspective than other approaches, providing feedback as fast as possible; ii) to provide a **generic** method to describe these untrustworthy behaviors and finally, iii) to automatically test the described behaviors providing verdicts regarding the trustworthiness of the interactions.

To tackle the first point, the distributed network monitoring approach was proposed. With the use of distributed network monitoring, we can see behaviors, that when using a single point of observation will not be possible to see. Let us present a possible case scenario, a client computer is sending request to perform operations to a server. Both, the client and the server have a trust management engine and they have allowed actions and replies from each other. The client computer (at the left) sends a message of the type "A" to the server (illustrated at the right side), and at the server side, the message received is a message of the type "B". The message type "B" is an allowed message type and the server performs the action. If the network traffic for both points of observation could be obtained and compared, an untrustworthy behavior can be detected. This example is illustrated in the Figure 3.1. Without correlating both points

of observation, the untrustworthy behavior cannot be detected, even if having trust management engines, the systems will consider the interaction trustworthy.
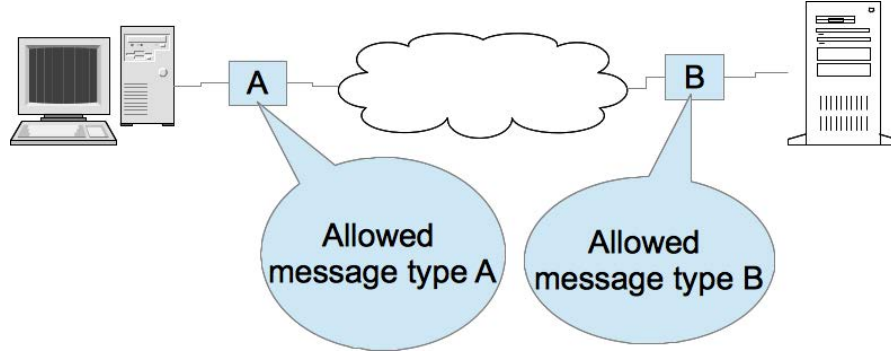
FIGURE 3.1: Data changed/tampered in the communication channel

Many other behaviors can be constructed from distributed network monitoring. The complete scenario is derived when relationships are created from different network traces.

We do not consider this simply a security issue. In fact, due to the trust definition we do not focus if the untrustworthy answer was due to an attack, a software failure (bug), system failure, etc. The relevant fact is that the interaction was not proper and reporting the untrustworthy interaction as soon as possible is our goal.

Our approach can solve problems with many other applications and scenarios. In the Section 3.4, we present a detailed case scenario that when using our approach, untrustworthy behaviors are detected.

In order to accurately understand how our proposed approach solves the stated issues, first we need to introduce some preliminary concepts. A **network packet** (packet for short) is the abstraction of the transmitted bit-streams in a computer network; this abstraction allows us to interpret a packet as a formatted data unit. A packet is interpreted as a "Message" from a telecommunications protocol, for example a DNS query, a DNS response, etc. **Analyzing a packet** is to access the data inside that packet to search for particular values; these values have a defined meaning depending on the network protocol. Finally, **network monitoring** is the technique of analyzing the packets transmitted over a computer network.

In this thesis, we assume that the network packets are being forwarded from the different sources of interest to a monitoring server. Each of these sources are network entities monitored through interfaces called points of observation (P.O). We also assume that if the network entity has many interfaces, all the forwarded packets from the same network entity will be considered the same point of observation.

The sequence of packets from a point of observation is called a *network trace.* A network trace (trace for short) is potentially infinite. When we have different traces from the points of observation, we can analyze the packets from one trace and create a relationship to another trace, defining the concept of **distributed network monitoring**. The complete scenario is derived when relationships are created from different network traces.

In order for us to provide evaluation about behaviors, relationships between packets from different P.Os are created. The relationships are created with the packets fields and conditions that hold over those fields in regards of other packets. The relationships are made by doing comparisons. We can compare the values of these observations with constant values or variable values. The variable values are extracted from other packets. These comparisons are defined formally later in this work by the definition of **atoms**. The atoms formalization is found on Definition 3.3. Please note that for the time relationships, we assume the network traces are synchronized using the NTP protocol [Mills, 1991]. Since there are multiple network traces from multiple P.Os, the comparisons can be done from: i) a specific network trace, that is through a specific point of observation, ii) any network trace, except a specific one or iii) any network trace, that is, any point of observation, i.e., not specifying a point of observation.

By using the packet relationships and comparing the values will result in a composition. This composition is formally defined also at Section 3.2 by the definition of **formulas** in Definition 3.4. One formula is a formal representation of what we will call a **trust property**. Many trust properties can be described and formalized in order to describe trust on a environment or context. Once the desired trust properties are checked on the network traces, we can give a testing verdict regarding the checked trust property. The possible verdicts are **pass**, **fail** if the statement is present. If the trust property does not reach a verdict, the result will be temporarily assigned to an **inconclusive** verdict. If many trust properties are described, then, different trust verdicts can be obtained.

3.2 Formal Description of Trust Properties

The motivation and method of our approach were presented in the previous Section. Now, in order to test the proposed trust properties using distributed network monitoring we need to be able to express those properties. It is not sufficient to express the trust properties, in fact we need to accurately express them, not leaving room for any ambiguity; considering that, we need to employ a formalism. A formalism is not only useful to unambiguously express the properties, but, also for the software tools to be able to provide accurate verdicts. Without a doubt, our approach is more valuable when verdicts can be automatized with a program. Further, when providing a formalism, more

researchers related to the field can generate trust properties to test. Because of those reasons we have created the necessary formal approach.

We decided to use our own approach rather than using other existing ones, the reason is that with the use of our formalization, we can describe the packets in a finely parameterized and at a granular level, as shown in Section 2.1. Thus, we can make more complex and detailed relationships between packets. New application protocols rely heavily in the data parts and the semantics of the protocols require a more data-oriented checking that other approaches which are on-line are not able to provide. Even old protocols have semantics that if the packets are treated as bit-streams some values in the data will be unavailable to obtain. For example, in the DNS protocol, the DNS notation and data compression method allows to specify a pointer to previously used data in the packet to avoid duplication of data (see [Mockapetris, 1987b]).

The formalism basic and most important principles are: the representation of a protocol message (packet) and the formal language lexical, syntactical and semantical properties; we discuss these aspects next.

3.2.1 Preliminaries, Basic Definitions, and Language Specification Formalisms

A communication protocol message is a collection of data fields of multiple domains. Data domains are defined either as *atomic* or *compound* [Che et al., 2012b]. An *atomic* domain is defined as a set of numeric or string values. A *compound* domain is defined as follows.

Definition 3.1. A *compound* value v of length $n > 0$, is defined by the set of pairs $\{(l_i, v_i) \mid l_i \in L \land v_i \in D_i \cup \{\epsilon\}, i = 1...n\}$, where $L = \{l_1, ..., l_n\}$ is a predefined set of labels and D_i are data domains. A *compound domain* is then the set of all values with the same set of labels and domains defined as $\langle L, D_1, ..., D_k \rangle$.

Once given a network protocol P, a *compound* domain M_p can generally be defined by the set of labels and data domains derived from the message format defined in the protocol specification/requirements. A *message* of a protocol P is any element $m \in M_p$.

For each $m \in M_p$, we add two fields: a real number $t_m \in \mathbb{R}^+$ which represents the time when the message m is received or sent by the monitored entity and $P.O$ a string label which represents the point of observation from which the message m is collected.

Example 1. A possible message for the DNS protocol [Mockapetris, 1987b], specified using the previous definition could be

$$m = \{(time, \text{`863.596183000'}), (P.O, \text{Auth\_DNS\_Srv})$$
$$(query\_id, 6912), (flags, \{(response, 0), (opcode, \text{std\_query}), (truncated, 0),$$
$$(recursion\_desired, 1), (reserved, 0), (non\_auth\_data\_acceptable, 0)\}),$$
$$(questions, 1), (answers, 0), (authority\_RRs, 0), (additional\_RRs, 0),$$
$$(queries, \{(name, \text{telecom-sudparis.eu}), (type, \text{A}), (class, \text{IN})\})\}$$

representing a DNS query for the domain telecom-sudparis.eu. The value of *time* '863.596183000' ($t_0 + 863.596183000$) is a relative value since the P.O started its timer (initial value t_0) when capturing traces.

A *trace* is a sequence of messages of the same domain containing the interactions of a monitored entity in a network, through an interface, i.e., the P.O, with one or more peers during an arbitrary period of time. The P.O also provides the relative time set $T \subset \mathbb{R}^+$ for all messages m in each *trace*.

As described in the Section 3.1, our approach focuses on applying distributed network monitoring to the trust management domain. In order to achieve that, we extended the language taken from the work realized in [Lalanne and Maag, 2013] which has previously been used for off-line passive testing of telecommunication protocols. Our current approach is for on-line and moreover, distributed systems, nevertheless, language exposed in that work can be used as a base to our formalism. In this work the syntax and semantics have been extended to include several P.Os. The syntax, based on Horn clauses is defined to express properties that are checked on extracted traces. We describe it in the following. Formulas in this logic can be defined with the introduction of terms and atoms, as it follows.

Definition 3.2. A *term* is defined in the Backus Normal Form (BNF) as $term ::= c \mid x \mid x.l.l...l$ where c is a constant in some domain, x is a variable, l represents a label, and $x.l.l...l$ is called a *selector variable*.

Example 2. Let us consider the message in example 1, the value of *recursion_desired* inside *flags* can be represented by **m.flags.recursion_desired** by using the *selector variable*.

Definition 3.3. An *atom* is defined as

$$A ::= \overbrace{p(term, ..., term)}^{k} \mid term = term \mid term \neq term$$
$$\mid term < term \mid term > term \mid term + term = term$$

where $p(term, ..., term)$ is a predicate of label p and arity k. The *timed atom* is a particular atom defined as $p(\overbrace{term_t, ..., term_t}^{k})$, where $term_t \in T$.

Example 3. Let us consider the message m of the previous example. A point of observation constraint on m can be defined as 'm.PO = Auth_DNS_Srv'. These atoms help at defining timing aspects as mentioned in Section 3.2.1.

The relations between *terms* and *atoms* are stated by the definition of clauses. A *clause* is an expression of the form

$$A_0 \leftarrow A_1 \wedge ... \wedge A_n$$

where A_0 is the head of the clause and $A_1 \wedge ... \wedge A_n$ its body, A_i being *atoms*.

Definition 3.4. A formula is defined by the following BNF:

$$\phi ::= A_1 \wedge ... \wedge A_n \mid \phi \rightarrow \phi \mid \forall_x \phi \mid \forall_{y>x} \phi$$
$$\mid \forall_{y<x} \phi \mid \exists_x \phi \mid \exists_{y>x} \phi \mid \exists_{y<x} \phi$$

where $A_1, ..., A_n$ are *atoms*, $n \geq 1$ and x, y are variables.

In our approach, while the variables x and y are used to formally specify the messages of a trace, the quantifiers commonly define "it exists" ($\exists$) and "for all" ($\forall$). Therefore, the formula $\forall_x \phi$ means "for all messages x in the trace, ϕ holds".

The semantics used previously presented resembles the traditional Apt–Van Emdem–Kowalsky semantics for logic programs [van Emden and Kowalski, 1976], from which an extended version has been provided in order to deal with messages and trace temporal quantifiers. Based on the above described operators and quantifiers, we provide an interpretation of the formulas to evaluate them to $\top$ ('*Pass*'), $\bot$ ('*Fail*') or '?' ('*Inconclusive*'). Here, we do not detail the evaluation and interpretation of the formal properties on traces. However, in the Section 3.4.4 all the algorithms are defined.

We formalize trust by using the syntax above described and the truth values $\{\top, \bot, ?\}$ are provided to the interpretation of the obtained formulas on real protocol execution traces. These formulas represent and allow to model **trust properties**.

3.3 Algorithms for Distributed On-line Network Property Evaluation

After having the properties formalized, we propose a set of algorithms in order to evaluate how each incoming packet from the live capture is evaluated to see if matches the disred
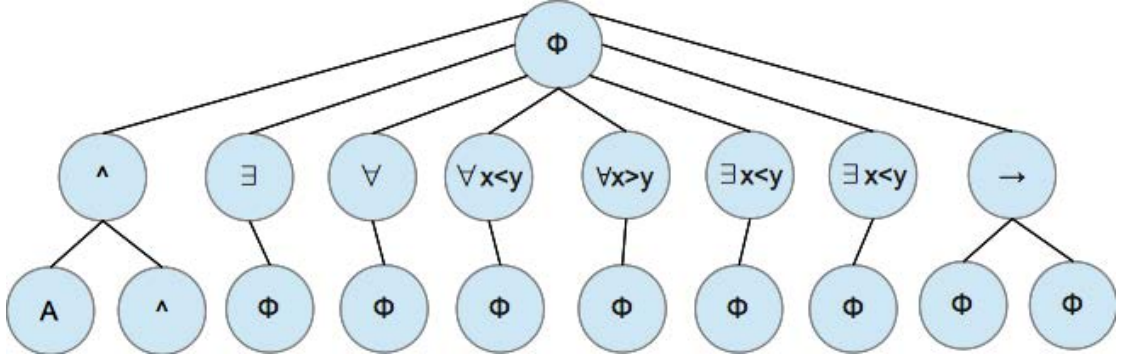
FIGURE 3.2: Syntax Tree Formula Representation

properties to test. The evaluation process is described in the following. The evaluation process, the algorithm to evaluate if a packet is a member of a set that satisfies a formula starts by understanding some properties about the formulas to check. If the production rule describing the formulas in the language grammar presented in Definition 3.4 is represented as a syntax tree, as shown in the Figure 3.2; it will produce a recursive subtree, in exception of the conjunction of atoms node ($\wedge$). Therefore, this conjunction of atoms will always be at the leafs of any formula. This conjunction of atoms will model what we call a *packet prototype*. This is such a crucial concept for the testing algorithms that a formal definition is necessary.

Definition 3.5. Let x be a packet prototype. x contains a set of atoms, $a_1, a_2, ..., a_n$. With the restriction on the cardinality of the set of atoms $n > 0$. The packet prototype contains all the necessary conditions for a packet to satisfy a given formula or part of it. This will also include dependencies to other prototypes and relationships between them, i.e., comparisons. When a packet matches all the atoms from a packet prototype, it will become tagged as the packet prototype prototype.

As an example of the prototypes, consider for a simple property:

$$\Phi = \forall_x(x.flags.request = 0 \rightarrow \exists_{y>x}(y.flags.request = 1 \wedge y.id = x.id))$$

The prototype y that belongs to the property Φ describes that, in order for a packet to be tagged as a "y", it needs to have the request flag set("on"). Also, it has a dependency of type x and with that packet of type x, both their "ids" should match.

Finally, when a packet is tagged with a prototype, it gets stored on a list of that prototype. Also, if the packet has dependencies it gets associated to those dependencies within the property that the prototype belongs. When all the prototypes of a property are completed or a packet of those associations gets a timeout, verdicts regarding that property can be provided. The associated packets get removed from the respective lists.

Typically each packet gets tested for each prototype, unless a prototype has dependencies and any of the dependency lists is empty. In this case, the packet does not even get tested for that prototype. This is a performance enhancement. All the design motivation for our testing algorithm is to have a fast matching so that a packet can be kept, tagged or discarded as fast as possible. The pseudo-code of the described evaluation algorithm for the **runmon** tool is shown on the Algorithm 1.

Algorithm 1: Algorithm of online tester

Input: packet data-structure //Sent by the different threads which processing the client connections.
Output: property verdicts report
for *each packet on live_capture* **do**

    packet_time ← get_time(packet);

    if *last_observed_packet_time < packet_time* **then**

        last_observed_packet_time ← packet_time;

    end

    for *each prototype on prototype_packets* **do**

        property ← get_prototype_property(prototype);

        if *match_properties_of(prototype, packet)* **then**

            prototype_list ← get_prototype_list(prototype);

            for *each prototype_dependency on dependencies(prototype)* **do**

                matched_dependency ← FALSE;

                for *each stored_packet on get_dependency_prototype_list(prototype_dependency)* **do**

                    if *match_properties_dependency(prototype_dependency, packet, stored_packet)* **then**

                        associate(packet, stored_packet, property), matched_dependency ← TRUE;

                        goto next_dependency;

                    end

                end

                if *!matched_dependency* **then**

                    goto next_prototype

                end

            end

            if *prototype_determines_property(prototype)* **then**

                associations_list ← get_associations(packet)

                report_property_pass(property, packet, associations_list)

                delete_from_prototype_lists(associations_list)

            end

            else

                push(prototype_list, packet)

            end

        end

        next_prototype:

    end

end

The Algorithm 1 is pretty straightforward, even if it looks somewhat large. It takes as an input a packet data structure, as the hierarchical attribute-value pairs as shown in the compound value definitions in Section 3.2. The output of the algorithm are the verdicts for the stated formulas for that packet. Each packet goes to the same testing process. The testing process starts by synchronizing the last observed time for the packets obtained from the current one. Then, for each of the prototypes of the formulas, it tests the prototype atoms which do not include dependencies to other packets; if all the tests are successful, the testing process continues, otherwise, the next prototype is tested. If the testing process continues, the packet tests each atom which contains a dependency against all the stored packets for that dependency; if all tests with the dependencies are successful, the packet gets tagged as the prototype being tested. Once the packet gets tagged, if the prototype is the last to be tested within a formula, the **Pass** verdict is given. On the contrary, if the prototype does not the last to be tested within the formula, it gets stored in the queue along with other packets tagged as the current tested prototype. Finally, the evaluation process continues for renaming prototypes.

As mentioned before, an important point to consider with our algorithm design is the scalability. This was taken into consideration in the evaluation algorithm. In here we present the time and space complexity of our algorithm. We start with the time complexity evaluation, by looking at the work our algorithm does in the worst-case scenario.

$$
\begin{aligned}
T(eval\_all\_prots) &= \sum_{i=1}^{N_p} (\Theta(1) + T(eval\_prop_i)) \\
&= N_p + \sum_{i=1}^{N_p} T(eval\_prop_i)
\end{aligned}
\tag{3.1}
$$

N_p is the number of prototypes.

$$
\begin{aligned}
T(eval\_prot_i) &= T(eval\_independent\_atoms_i) + \Theta(1) + \sum_{j=1}^{NPD_i} T(eval\_dep_j) + \Theta(1) \\
&= 2\Theta(1) + T(eval\_independent\_atoms_i) + \sum_{j=1}^{NPD_i} T(eval\_dep_j)
\end{aligned}
\tag{3.2}
$$

NPD_i is the *ith* prototype number of dependencies.

$$T(eval\_independent\_atoms_i) = \sum_{j=1}^{NPA_i} \Theta(1)$$
$$= NPA_i$$

(3.3)

NPA_i is *ith* prototype number of atoms that require no dependencies.

$$T(eval\_dep_j) = \Theta(1) + \sum_{k=1}^{QL_j} (T(eval\_dependency\_atoms_k) + \Theta(1)) + \Theta(1)$$
$$= 2\Theta(1) + QL_j + \sum_{k=1}^{QL_j} T(eval\_dependency\_atoms_k)$$
$$= 2\Theta(1) + QL_j + QL_j * T(eval\_dependency\_atoms_k)$$

(3.4)

QL_j is the length of the queue of stored packets for dependency j.

$$T(eval\_dependency\_atoms_k) = \sum_{l=1}^{NDA_j} \Theta(1)$$
$$= NDA_j$$

(3.5)

NDA_j is the number of atoms of the dependency j.

Substituting the equation 3.4, 3.5 and 3.3 into the equation 3.2 we obtain the following equation:

$$T(eval\_prot_i) = NPA_i + 2\Theta(1) + \sum_{j=1}^{NPD_i} (2\Theta(1) + QL_j + QL_j * NDA_j)$$
$$= NPA_i + 2\Theta(1) + 2NPD_i + \sum_{j=1}^{NPD_i} (QL_j + QL_j * NDA_j)$$

(3.6)

Substituting the equation 3.6 into the equation 3.1 we obtain the following equation:

$$T(eval\_all\_prots) = N_p + \sum_{i=1}^{N_p} \left(NPA_i + 2\Theta(1) + 2NPD_i + \sum_{j=1}^{NPD_i} (QL_j + QL_j * NDA_j) \right)$$

$$= N_p + 2N_p + \sum_{i=1}^{N_p} NPA_i + \sum_{i=1}^{N_p} 2NPD_i + \sum_{i=1}^{N_p} \sum_{j=1}^{NPD_i} QL_j + \sum_{i=1}^{N_p} \sum_{j=1}^{NPD_i} QL_j * NDA_j$$

$$= 3N_p + \sum_{i=1}^{N_p} NPA_i + \sum_{i=1}^{N_p} 2NPD_i + \sum_{i=1}^{N_p} \sum_{j=1}^{NPD_i} QL_j + \sum_{i=1}^{N_p} \sum_{j=1}^{NPD_i} QL_j * NDA_j$$

$$(3.7)$$

The meaning behind the $\sum_{i=1}^{N_p} \sum_{j=1}^{NPD_i} QL_j * NDA_j$ term, is that, for each prototype dependency, all the atoms related to that dependency need to be tested against the stored packets of that dependency. Considering the worst-case analysis, that will be when each prototype takes all possible dependencies into the evaluation. According to the semantics of the language, the largest amount of dependencies a prototype can take is all the previously observed dependencies, i.e., when, $NPD_i = i - 1$. Under that assumption, the expression $\sum_{i=1}^{N_p} \sum_{j=1}^{NPD_i} QL_j * NDA_j$ can be written as: $\sum_{i=1}^{N_p} \sum_{j=1}^{i-1} QL_j * NDA_j$. This expression can be reduced to $\sum_{i=1}^{N_p}((N_p - i)QL_i * NDA_i)$(and respectively $\sum_{i=1}^{N_p} \sum_{j=1}^{NPD_i} QL_j$ to $\sum_{i=1}^{N_p}(N_p - i)QL_i)$. Finally, the work performed by our algorithm in the worst-case can be expressed in the following equation:

$$T(eval\_all\_prots) = 3N_p + \sum_{i=1}^{N_p} NPA_i + \sum_{i=1}^{N_p} 2NPD_i + \sum_{i=1}^{N_p}(N_p - i)QL_i + \sum_{i=1}^{N_p}((N_p - i)QL_i * NDA_i)$$

$$(3.8)$$

Where N_p is the number of prototypes in the formulae, NPA_i is number of atoms that require no dependencies of the *ith* prototype, NPD_i is the number of dependencies of the *ith* prototype, QL_i is the length of the queue of stored packets of the *ith* prototype, and NDA_i is the number of atoms that require dependencies of the *ith* prototype.

As shown in the equation 3.8, the algorithm heavily depends on two different parameters. The first parameter is the number of atoms in the formulae (individual comparisons of the packets), hereinafter denoted as n, and the second one is the length of the stored packet queues. Depending on the number of stored packets and on the number of atoms in the formulae, different ways to express the time complexity of the algorithm can be obtained. We do the complexity algorithm for the both parameters. If we consider that the asymptotic upper bound depending on the number of atoms in the formulae, we

can say that the running time of the algorithm can be expressed by a function which is $f(n) = O(n) = \sum_{i=1}^{N_p} NPA_i + \sum_{i=1}^{N_p}((N_p - i)QL_i * NDA_i)$, a linear combination of all the atoms in all prototypes. On the other case, if we consider the asymptotic upper bound depending on the length of the queues of stored packets, we get that the running time of our algorithm can be expressed by $f(n) = O(n) = \sum_{i=1}^{N_p}(N_p - i)QL_i + \sum_{i=1}^{N_p}((N_p - i)QL_i * NDA_i)$, yet again a linear combination of elements. Given the experiments performed so far, the running time of the algorithm is good which the is backed up with the complexity analysis which provides us the linear time $(O(n))$ complexity. Finally, if denoting the complexity as a function of all involved variables it is said to be $f(n, l) = O(n * l) = \sum_{i=1}^{N_p} NPA_i + \sum_{i=1}^{N_p}((N_p - i)QL_i * NDA_i)$.

The space complexity analysis for our algorithm follows, we start by looking at the space our algorithm requires in the worst-case scenario.

$$
\begin{aligned}
S(eval\_all\_prots) &= \sum_{i=1}^{N_p}(\Theta(0) + S(eval\_prot_i)) \\
&= \sum_{i=1}^{N_p} S(eval\_prot_i)
\end{aligned}
\tag{3.9}
$$

N_p is the number of prototypes.

$$
\begin{aligned}
S(eval\_prot_i) &= \Theta(0) + \Theta(0) + \sum_{j=1}^{NDP_i} S(eval\_dep_j) + \Theta(1) \\
&= \sum_{j=1}^{NDP_i} S(eval\_dep_j) + \Theta(1)
\end{aligned}
\tag{3.10}
$$

NPD_i is the *ith* prototype number of dependencies. $\Theta(1)$ is the result of storing the packet in the prototype list assuming the worst-case scenario where the formulae will store all packets get stored in all lists.

$$
\begin{aligned}
S(eval\_dep_j) &= \Theta(0) + \Theta(1)) \\
&= \Theta(1)
\end{aligned}
\tag{3.11}
$$

The $\Theta(1)$ is the result of associating always the packet to each dependency.

Now, substituting the equation 3.11 into the equation 3.10 we get:

$$S(eval\_prot_i) = \sum_{j=1}^{NDP_i} \Theta(1) + \Theta(1) \tag{3.12}$$
$$= NPD_i + 1$$

Similarly, if substituting the equation 3.12 into the equation 3.9 we get:

$$
\begin{aligned}
S(eval\_all\_prots) &= \sum_{i=1}^{N_p} NPD_i + 1 \\
&= \sum_{i=1}^{N_p} 1 + \sum_{i=1}^{N_p} NPD_i \\
&= N_p + \sum_{i=1}^{N_p}(NPD_i + 1)
\end{aligned}
\tag{3.13}
$$

Where N_p is the number of prototypes in the formulae, NPD_i is the number of dependencies of the *ith* prototype.

For the sake of consistency, in order to express our space complexity with the same variable as our time complexity, we note that the number of dependencies contained in each prototype can be described as the number of dependent atoms multiplied for a constant, that is $NPD_i = c_i * NDA_i$. The reason is that for a prototype to have a dependency at least an atom is needed to express it. Therefore, the space required by our algorithm can be expressed like: $N_p + \sum_{i=1}^{N_p}(c_i * NPD_i + 1)$ and reduced to: $2N_p + \sum_{i=1}^{N_p}(c_i * NPA_i)$. This implies that the space complexity can be expressed as a linear combination of the number of dependent atoms in each prototype, i.e., $f(n) = O(n) = \sum_{i=1}^{N_p}(c_i * NPA_i)$. This result makes sense, since, an algorithm that can be executed in linear time can only modify linearly the memory.

Besides the main process listening to client connections, two threads are created; one for periodically updating the current status to the user and the other for the global timeout of packets. The global timeout of packets refers to the maximum time a packet can be kept if this packet is not a dependency of any other packet within a property. This global timeout is used by the on-line monitoring to ensure that potentially incomplete properties do not occupy resources permanently. For instance, a very simple property which states that: "for each request there should exist a response" if no response exists, then, the request package would potentially be stored permanently and no fail verdict will be reported. When a packet is decided to be removed by the *timeout function* if the packet has not been used for completing one property, then a **Fail** verdict will be

reported for that property. The value of the timeout its determined by an expert in the protocol. Let us assume using the same property, there is no request present (or has not occurred). This is when a property actually can report the **Inconclusive** status. The basic idea behind the inconclusive verdict is that there is not enough information present to provide other verdict. In general terms it happens due to the lack of presence of expected packets where those packets are the cause of the property. As in our previous example, no requests are present. In more specific cases, it can happen when the cause of a property relies on a P.O who's network traffic is missing at the monitoring server. The inconclusive verdict in on-line monitoring differs from off-line approaches. In off-line approaches, when an expected reply is not found at the end of a collected packet capture (in a file, for instance), this causes an inconclusive verdict. In on-line network monitoring, the presence of a consequent packet is affected by the configured timeouts. The pseudo-code for the timeout function is shown in the Algorithm 2.

Algorithm 2: Algorithm of timeout function

Input: global_timeout_value
Output: property verdicts report / packet list cleanup
sleep(global_timeout_value);
for *each prototype on prototype_packets* **do**
    property ← get_prototype_property(prototype);
    for *each stored_packet on get_prototype_list(prototype)* **do**
        associations_list ← get_associations(stored_packet);
        report_property_fail(property, stored_packet, associations_list);
        delete_from_prototype_lists(associations_list);
    end
end

The Algorithm 2, takes as input the global timeout parameter, the maximum time a packet is allowed to be present in the system without completing a formula (providing a pass verdict). As an output, it provides fail verdicts for all the packets that exceeded the timeout and cleans the prototype packet queues; removing the no-longer needed packets due to the timeout. The algorithm starts by waiting the global timeout value, after the global timeout value passed, it proceeds to go into each of the prototypes. For each prototype, it goes into each of the stored packets queues and if the elapsed time of the packet exceeds the global timeout, the packet gets deleted and the function reports fail.

In this section we presented the details and design about our software. In the next section we present the experiments performed by using our tools and formalism, in order to provide the evaluation about the behavior of the described system and to provide trust verdicts.

3.4 Experiments

3.4.1 Preliminaries — The Domain Name System

The Domain Name System (DNS), standardized on RFC 1035 [Mockapetris, 1987b], is a hierarchical naming system for network entities. Its most common known use is to associate a domain names to numerical IP address. Although, it can be used to store several other types of information. It is one of the most essential services in the Internet. The main purpose is to make easy for users to remember specific hostnames or service providers. For instance, it is easy to remember the name "google.com", yet, is hard to remember "173.194.40.167" which is actually one of the IP addresses associated for that domain name. Moreover, for all the services in the Internet, DNS provides information as well, for example, information about where certain service as the mail exchanger is located for certain domain.

The functional aspect of DNS as described before is hierarchical. A domain name consists of labels that are concatenated to each other and separated by dots. For example the domain name "www.example.com" consists of three labels, "www", "example" and "com". The labels hierarchy goes from left to right, being the rightmost label the one with the highest hierarchy. The rightmost label is called top level domains (TLD). The information of the TLDs are stored on the root DNS servers. The TLD DNS servers have the information of the authoritative DNS servers for each domain. The authoritative DNS servers are the servers that have the official DNS records for a domain.

The domain name resolution mechanism starts with a DNS resolver. The DNS resolvers asks its local caching DNS server the domain in question. The caching DNS server is usually a recursive resolver. It queries the root DNS servers for the information about the TLD of the domain in question. After that, it queries the TLD server to obtain the information about the authoritative DNS server. Then it continues this recursive process until it reaches an answer. The final step is to return the DNS resolver an answer. In Figure 3.3 an example message sequence chart of a DNS resolver asking the domain "subdomain.domain.tld" and its resolution process is shown.

We shortly described the domain name system in order to have clear concepts of the problematic we decided to tackle. In the following subsection we describe the scenario.

3.4.2 Scenario Description

We choose to tackle the same scenario as chosen by [Jim, 2001] and [Fan et al., 2011] since this is still an open issue on trust and security. Also, the implications of trust and
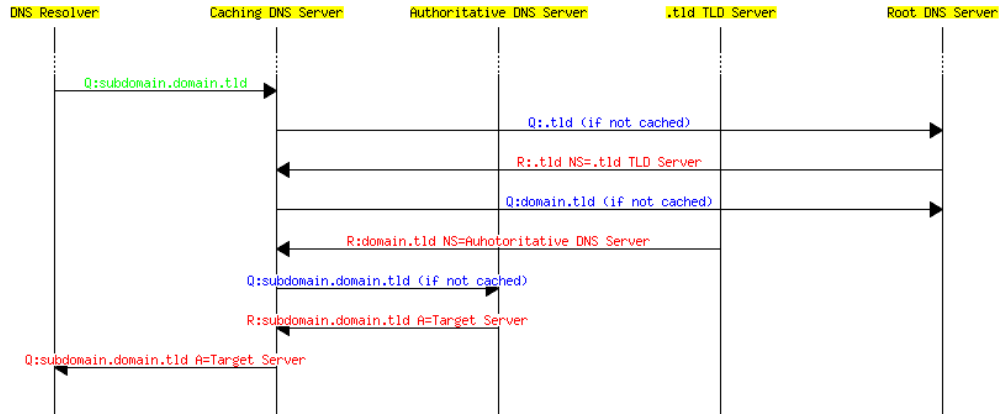
FIGURE 3.3: Distributed monitoring of DNS responses

security on this scenario can affect all types of end users and systems. The problem can be described as trusting DNS responses to queries.

The DNS original design does not take into consideration any concept of trust or security (although an extension to address these issues exist, several challenges are still present. We discuss this later). If the DNS query responses are modified, it can have severe implications. End users and systems can be deceived and send data to the wrong destination. The target is to divert the data from the original source. The information can be later delivered to the real destination; the end user or system might not notice any untrustworthy interaction. End users can be directed to phishing pages, advertising pages or any other. Moreover, the system can have a trust management engine in place and consider all the interactions as trustworthy.

To understand how a DNS response can be changed, we first analyze the structure of a DNS query and response. A DNS query has the following relevant parameters:

- Source address and **source port**.

- **Destination address** and destination port (53).

- **Query ID**.

- Query name, type and class.

The response back includes the same fields. The use of the query ID is to synchronize/correlate the response back. Knowledge about three parameters is required to successfully spoof a response: the query ID, source port and Destination address.

Many attacks against the DNS systems have been studied. The attacks basically work by winning a "race condition". The objective is to get the spoofed responses arrive

before the original one. If a specific domain is the target of the spoofing, the queries can be forced. For example, connecting to an SMTP server and providing the desired domain name in order to get the resolver to query it. Many techniques can be used in order to spoof the packet. Brute force, random chance or man in the middle attacks. Perhaps, the most effective one is the Kaminsky attack as analized in [Alexiou et al., 2010]. The possibility that the caching DNS server is compromised by any other way or that messages are being tampered in the middle exists. In any of the situations mentioned before, the caching DNS server was not acting securely.

In many occasions DNS caches change the records, willingly or not. Possibly, ISPs can do this at their caching DNS servers; motivated to save bandwidth or to hijack the non existent domains to forward users to publicity sites. We consider this case as the caching DNS server is not acting dependably. We can also consider the case the caching DNS server is returning the wrong records due to a system failure or software bug. In this case the caching DNS server lacks of competence. In [Dagon et al., 2008] the authors analyze many cases including the hijacking of non existent domains for commercial abuse and the software implementation errors.

If the caching DNS server is not competent to act dependably, securely, and reliably to return the correct DNS records, it is not acting in a trustworthy manner by definition. That is the reason why this cannot be considered only a security issue. This is a trust issue, trustworthy interactions between the DNS caching server and the resolvers are a must.

A DNS security extension, DNSSEC, is proposed at RFC 4033 [Arends et al., 2005]. In a nutshell, DNSSEC uses electronic signed data. The signatures are encrypted using asymmetric keys. The signatures are added in a hierarchical chain. All DNS resolvers have the public keys from the root DNS servers. The DNS resolvers do the chain resolution. This makes a DNS resolver also able to verify the returned records actually belong to an authoritative server.

In [Jim, 2001], to prove their framework, the authors also created a service similar to DNSSEC. This is done using their tool, SD3, by implementing policies and using the tool certificate evaluator built in the tool.

DNSSEC has backward compatibility in its design. This can be desirable, since DNS resolvers can decide not to implement the DNSSEC verification and they will be able to continue working despite the fact the authoritative records implement DNSSEC. However, a true implementation on the Internet can take many years; because, in order for it to truly work, a complete change on all internet DNS authoritative servers and

DNS resolvers is needed. A similar case happens with the IPv6 protocol [Deering and Hinden, 1998], the adoption of this protocol requiere a global change in the Internet.

Let us assume the following case, when an employee is working at a remote location, which can be a hotel with public Internet connection. The employee is using a company application that sends data to a company application server. The hotel's caching DNS server sends the wrong information to the client computer. The client computer will send the application data to a third party machine. This third party machine eavesdrops the communication and then re-routes traffic to the original destination, that is the company's application server. Both the application server and the client computer will not detect any malicious behavior.

By the use of distributed network monitoring we can improve the trustworthiness in the DNS responses. Combining information from different points of observation allow to identify situations that with a single observation will not be possible. The client computer sends the DNS traffic to the monitoring server. Also, the authoritative DNS server for the domain sends the DNS traffic to the monitoring server as well. A simulation of this scenario is shown in the Section 3.4.5

As this scenario and previous sections illustrated, we can use the formal approach and create trust properties to provide trust information which will be detailed in the following subsection.

3.4.3 Formal Specification

We have formally defined one formula in order to express trust properties. As described in the scenario, our target is to guarantee that the responses from the DNS resolvers match the responses from the authoritative DNS server. With the use of distributed network monitoring and our formal specification we can declare the necessary trust property. After evaluating it, it will provide trust verdicts applied to the previously described scenario. We describe the trust property and then formalize it.

The trust property, extracted from the concept and facilities of DNS [Mockapetris, 1987a] is: $\Psi = $ *"For all responses from an authoritative DNS server, all future responses from other points of observation are the same replies of the authoritative DNS server if the queries are the same"*. This can be expressed by the following formula:

$$\Psi = \{\forall_x(req\_f(x, ADS)) \to \exists_{y>x}(res\_f(y, x, ADS))\} \to$$
$$\{\forall_a(req\_anf(a, y, ADS)) \to \exists_{b>a}(res\_enf(b, a, y, ADS))\}$$

Where the intermediate clauses are defined such as:

$req(x) \leftarrow x.flags.response = 0$

$res(y) \leftarrow x.flags.response = 1$

$eq\_q(x, y) \leftarrow x.queries = y.queries$

$eq\_a(x, y) \leftarrow x.answers = y.answers$

$from(x, P) \leftarrow x.PO = P$

$nfrom(x, P) \leftarrow x.PO \neq P$

$after(x, y) \leftarrow x.time > y.time$

$resp(y, x) \leftarrow res(y) \wedge y.query_id = x.query\_id \wedge eq\_q(x, y)$

$req\_f(x, P) \leftarrow req(x) \wedge from(x, P)$

$req\_nf(x, P) \leftarrow req(x) \wedge nfrom(x, P)$

$res\_f(y, x, P) \leftarrow resp(y, x) \wedge from(y, P)$

$res\_nf(y, x, P) \leftarrow resp(y, x) \wedge nfrom(y, P)$

$req\_a(x, y) \leftarrow after(x, y) \wedge req(x) \wedge eq\_q(x, y)$

$req\_anf(x, y, P) \leftarrow req\_a(x, y) \wedge nfrom(x, P)$

$res\_enf(x, y, z, P) \leftarrow res\_nf(x, y, P) \wedge eq\_a(x, z)$

"ADS" is the assigned label to the authoritative DNS server P.O (see Section 3.2).

The second trust property is related to the DNS updates. An update in the DNS values will imply that caching DNS servers might have the wrong records. However, this is the expected behavior; the caching DNS servers can have the wrong values for the validity of the time to live (TTL) of each record. Even if this is the expected behavior, this means the caching DNS server has untrustworthy records. In this work we do not focus on how to measure or interpret the trust information we provide. Nevertheless, this information could be used by caching DNS servers to re-query the authoritative DNS server when a change in the records is performed. We will consider that a caching DNS server with the wrong DNS records can still be trustworthy if the following trust property holds: $\Phi = $ *"For all responses from an authoritative DNS server, if it exists a future response from other points of observation that is not the same response of the authoritative DNS server and the queries are the same, then, a previous authoritative DNS response with the same value as the conflicting response must exist; also, its TTL should be bigger than the difference between the conflicting response and the previous authoritative response"*. This can be expressed by the following formula:

$$\Phi = \{(\alpha \rightarrow \beta) \rightarrow \gamma\}$$

$\alpha = \{\forall_x(req\_f(x, ADS)) \rightarrow \exists_{y>x}(res\_f(y, x, ADS)\}$

$\beta = \{\exists_a(req\_anf(a, y, ADS)) \rightarrow \exists_{b>a}(res\_dnf(b, a, y, ADS)\}$

$$\gamma = \{\exists_m(req\_bf(m,a,ADS)) \rightarrow \exists_{n>m}(res\_eft(m,n,b,ADS))\}$$

The intermediate clauses used for the first formula are used in these intermediate clauses, but, also we define new ones, which are:

$ne\_a(x,y) \leftarrow x.answers \neq y.answers$

$bef(x,y) \leftarrow x.time < y.time$

$ttl_a(x,y) \leftarrow x.anwers.TTL + x.time > y.time$

$req\_b(x,y) \leftarrow bef(x,y) \wedge req(x) \wedge eq\_q(x,y)$

$req\_bf(x,y,P) \leftarrow req\_b(x,y) \wedge from(x,P)$

$res\_dnf(y,x,z,P) \leftarrow res\_nf(y,x,P) \wedge ne_a(y,z)$

$res\_ef(y,x,z,P) \leftarrow res\_f((y,x,P) \wedge eq\_a(y,z)$

$res\_eft(y,x,z,P) \leftarrow res\_ef(y,x,z,P) \wedge ttl_a(y,z)$

By formalizing two trust properties we demonstrate how generic our method is and the global applicability to any other protocol. Furthermore, how using our formal approach, trust properties can be formulated using the description of the behavior through the use of distributed network monitoring. The only requisite is to have the messages of the protocol represented as a hierarchical attribute-value pairs, i.e., compound values as described in Section 3.2.

3.4.4 Software Tools for Automatized Testing of Trust Properties: extmon and runmon

Automatically testing the trust properties gives a lot of added value to our contribution. In order to automatically test the trust properties using the formal approach presented in the Section 3.2. Furthermore, the approach used to implement the tools and the implemented algorithms rely on the syntax of the language and the concepts presented in the same formal approach section. We developed two different software tools. Both software pieces were developed in the C language. Those are: **extmon**: A software tool used to forward filtered network traffic observed locally to a remote monitoring server. This software runs on all different P.Os. **runmon**: A server-side program that gathers, correlates and tests the packets sent from different points of observation. This tool runs on the monitoring server. More details about the design of both software tools is given in the following.

The **extmon** tool captures the network traffic using libpcap [McCanne and Jacobson, 1993]. The live capture starts on a network interface specified by the user; an off-line evaluation is possible by supplying a filename instead of an interface. However, for
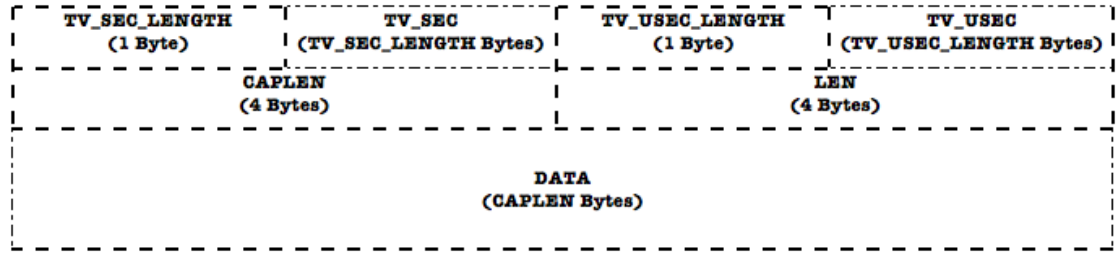
FIGURE 3.4: PCAP Over Network packet format

off-line captures, the time at which the packets were captured can affect when making time relationships with other live captures. Also, all packets are forwarded without any pause by the tool and that can produce undesired results. The traffic is filtered, so only interesting traffic gets forwarded to the monitoring server. This filtering is also done via libpcap. An example of a filter could be "port 21", for instance, in case that we are interested in the FTP control traffic. Please note that by default the traffic directed to the monitoring server (address and port) is always excluded to avoid flooding and potentially a self DoS. Hence, an explicit filtering of this is not needed.

After the interesting traffic has been filtered, a connection to the user-specified monitoring server is established. This connection is by default a secure connection; the data get encrypted using well known cryptographic protocols, SSL[Freier et al., 2011] and TLS[Dierks, 2008]. The motivation for using a secure connection by default is to avoid introducing security flaws when handling the interesting traffic. If the data were sent in clear text, the monitoring channel could introduce security holes, for instance leaving the application data vulnerable to man-in-the-middle attacks. Regarding the protocol in which our tool transmits the information, we developed a very simple protocol. The protocol transmits data using the TCP transport layer[Postel, 1981] to guarantee the reliable deliver of packets. No specific port has been chosen for the communication of the protocol; as of now, the port is a configurable option. The data sent by the **extmon** tool follow a very simple packet format. The format of the packets is based on the data structures provided by libpcap. One of the most important aspects regarding the protocol is that the time when the packet was captured is included in the packet. The captured time is important due to the synchronization issues as mentioned in Section 3.1. The format of the protocol packets is described in Figure 3.4.

The description of each field is explained on the Table 3.1.

On the receiving end, the **runmon** tool, a server socket is bound to a local IP address and port, and it listens for client connections. For each new connection a new thread is created that processes the bit-streams of network traffic. Those streams are mapped into a data structures and the specific packet properties are added. For example, the

TABLE 3.1: PCAP Over Network Packet Format Fields

Field	Description
TV_SEC_LENGTH (1 Byte)	The field with the length of the TV_SEC field. This is due to the libpcap specification, that states that data structures should use architecture dependent representation for time values. Some architectures might have different length of those time values. Moreover, for future compatibility this avoids changing the packet structure.
TV_SEC (TV_SEC_LENGTH Bytes)	The field containing the time at which the captured package occurred, in Unix-time representation.
TV_USEC_LENGTH (1 Byte)	The field with the length of the TV_USEC field. The same concept as TV_SEC_LENGTH.
TV_USEC (TV_USEC_LENGTH Bytes)	The field containing the rest of the elapsed time in microseconds at which the captured package occurred.
CAPLEN (4 Bytes)	The field with the length of the portion present of the captured packet.
LEN (4 Bytes)	The field with the actual length of the captured packet.
DATA (CAPLEN Bytes)	This is the actual data of the captured packet. This includes all network layers, which means it will contain the intact information as sniffed. This is important, since the forwarded packet can be delivered via a different network interface (or using a relay) and the point of observation will be taken from information contained in the captured packet. The design is like this, because, the interactions of interest are the ones being forwarded through the capturing tool.

P.O information depending on the package source and destination IP addresses. Also, the packet time is used for controlling the last observed time of packets which is used for timeouts. The bit-stream arrival time is not used, since, network delays in delivering the traffic could provide false results of the package arrival times. As stated in Section 3.1, we assume those systems have time synchronization.

Finally, the interaction between the **extmon** (forwarding the network traffic) and **runmon** is shown on Figure 3.5. All clients are running the **extmon** tool and the monitoring server is running the **runmon** tool. Please note that given the evaluation process described above the number of points of observation is only limited by the resources of the host running the **runmon** tool.

As shown on Figure 3.5, the interaction requires of several points of observation(P.Os). In most of the cases, the traffic forwarded using the extmon utility to the monitoring server can be enclosed by the trust properties and their requirements; if the trust properties work only on a specific protocol, there is no need to send all traffic. In our case, we were only interested in the DNS protocol, therefore, the DNS traffic is filtered and forwarded. The necessary intervention for a system to become a monitored node consists only in installing the utility, configuring what is the interesting traffic, the target server and other necessary parameters. The required access rights to install the utility are super-user rights; in order to open a live capture on an interface, super-user access rights are needed. This depends from operating system to operating system, but, in most cases these privileges are required. Nevertheless, the software is installed with execution super-user privileges (known in Linux as the sticky bit). Therefore, a regular user is able to execute the software if the administrator installs the tool with this permission or adds
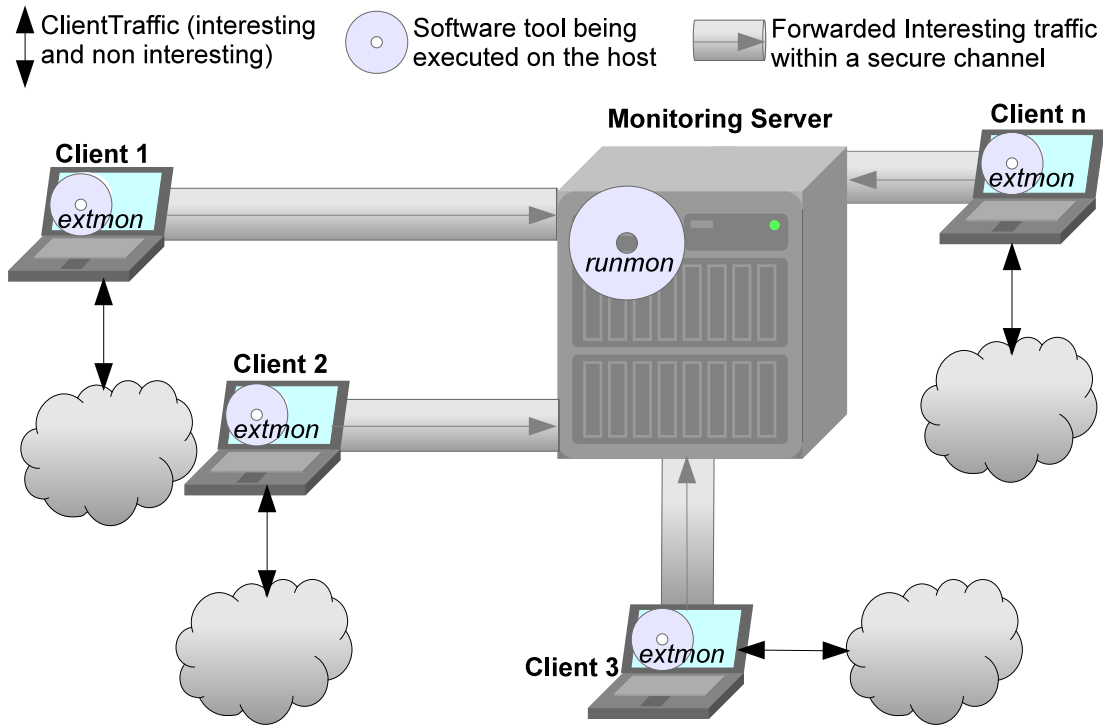
FIGURE 3.5: extmon / runmon interaction

it afterwards. This process needs to be done only once. Afterwards, executing the tool can be an automated and simple process. In fact, forwarding packets with our utility partially creates the points of observation. A point of observation is considered per interface. It does not really matter if that interface has many IP addresses configured. However, the points of observation depend on the configuration on the tester / target server. In the monitoring server, the final configuration of the P.Os can be done by configuring per combinations of IP address and ports. For instance, three different IP addresses are considered to be the same P.O or the same IP address is considered different P.Os with different ports. This was done to provide a broader degree of flexibility.

The amount of P.Os needed depends on the desired trust properties to check. For example, for the particular case where a single point of observation is required by the trust properties, only the interaction of that P.O and its peers can be checked; this will be external network monitoring, and not, distributed network monitoring. Nevertheless, the usability to external monitoring is valuable, for instance for cloud computing systems. On the other hand, when trust properties depend on several P.Os, the necessary ones need to be available. If the input of the required and necessary P.Os is not available at the monitoring server, it will not be able to provide a valuable conclusion.

3.4.5 Experimental Simulation Architecture and Results

We proposed a first approach of how, by using our approach, trust verdicts can be obtained. By manually checking combined off-line traces from different points of observation we proved the value of the formal approach [López et al., 2013]. In this work, we collect the live captures from different points of observation, on-line and automatically. The development of the necessary on-line and distributed network monitoring concepts, developing tools and performing experiments and simulations have been done to extend this work. In this work. we conducted our experiments performing a simulation of the specified scenario on a real case study. A monitoring server was deployed. The **runmon** tool was executed in that server. On desktop machines (clients), we executed our **extmon** tool, forwarding the DNS traffic to the monitoring server. Also, the **extmon** tool was executed in a live server provided by an industrial partner, Tilidom[1]. These experiments were realized at "ns2.tilidom.com", one of the authoritative DNS servers for the domain "tilidom.com". In this server, the only traffic forwarded to the monitoring server was the DNS traffic as well. A table with the machines configuration is shown in Table 3.2.

Machine designation	CPU configuration	RAM configuration	Operating system
Monitoring Server	2 X Intel(R) Pentium(R) CPU G6950 @ 2.80GHz	4GB	debian 6.0.6
Client 1 / Client 2 / Rogue DNS Server	Intel(R) Core(TM) i5-2415M CPU @ 2.3GHz	2GB	CentOS 6.4 with GUI, Virtual Machine running on VirtualBox 4.3.6 r91406 for Mac OS X 10.9.1
Authoritative DNS Server	–(Not disclosed for privacy reasons)	–(Not disclosed for privacy reasons)	–(Not disclosed for privacy reasons)

TABLE 3.2: Machines Configuration

As explained in the Section 3.4.4, the runmon utility is running at the monitoring server, testing the coded properties, and the extmon utility is running in all other points of observation. At the **runmon** tool we configured the P.O "ADS" for the IP addresses of the authoritative DNS servers for the domain "tilidom.com". The Ψ property was coded into the **runmon** tool. We also created a DNS service in other server (the rogue DNS server) using the bind[2] software. At that service we created a master zone for the domain

---

[1] http://tilidom.com/
[2] https://www.isc.org/downloads/bind/

"tilidom.com" and created different values for the DNS records. These values differed from the answers of the authoritative DNS server values. This in order to simulate a rogue DNS server. This rogue DNS server was configured as the DNS resolver for the "*Client 2*" machine. We illustrate our testing architecture in Figure 3.6.
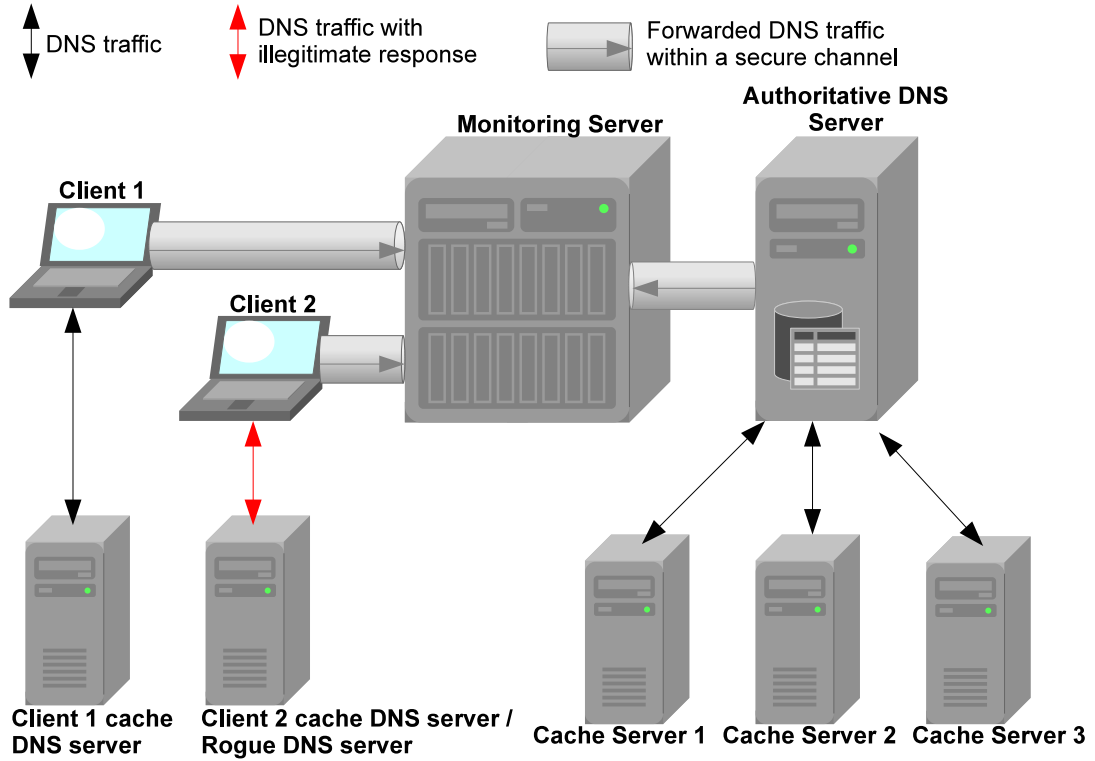
FIGURE 3.6: Simulation Architecture

The authoritative DNS server always replies to queries from different DNS caching servers, such as, Google public DNS servers, OpenDNS servers or Level3 DNS servers. Thus, this traffic was the one that was being forwarded from the authoritative DNS server to the monitoring server. At the client machines we forced the query for the particular domain, "tilidom.com" by going to a web browser and requesting that web page.

The Message Sequence Chart (MSC) for the relevant messages received by the monitoring server is shown on Figure 3.7.

PASS verdicts were obtained for the Ψ property for all hosts in exception of the traffic sent by "*Client 2*". For the "*Client 2*" **FAIL** verdicts were obtained. This was in fact the expected result of the simulation. Results from the Φ trust property were $\top$ ('*Pass*'). This means that on that network trace no DNS updates or untrustworthy responses were provided.
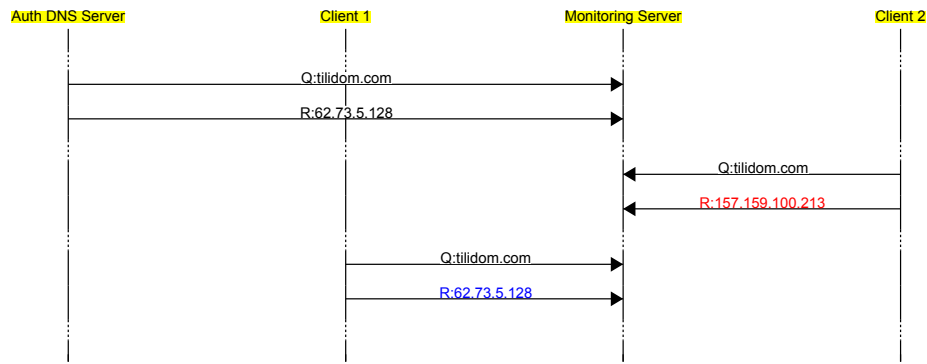
FIGURE 3.7: MSC for relevant messages at the monitoring server

3.4.6 Performance Evaluation

Some notions regarding the performance evaluation of our tools are presented in this subsection. When executing our tools, the load of the different resources has been observed. The observed performance so far is good; the runmon tool took few resources. When testing, the runmon tool usually consumes around 40MB of RAM memory and the CPU usage increases around 5% of. Nonetheless, this depends highly on the tested properties and amount of received packets, we discuss some tests performed to better outline the result. In order to perform a stress test, an off-line capture of three million packets processed without any delay, the memory increased up to around 80MB. The reason why the RAM memory does not get a high impact is because of the timeouts; the timeouts avoid the RAM memory resource starvation. When testing the off-line capture with three million packets, the CPU usage reached a 100% since, there exists no limit to the amount of resources the tool uses and all packets were processed at once. The network traffic was increased linearly by the receiving messages. On the sending hosts, the forwarding of packets uses no perceivable CPU consumption. When testing the tool to send up to 100 packets per second, the CPU usage increased in a 10% for the lasting period of the test. The same rise was experienced when the host sent 100 packets per second without the tool running. Finally, regarding the network traffic for the extmon tool, it increased in the amount of duplicated data to the monitoring server. No noticeable change in the memory was observed.

Chapter 4

Using Process Mining to Automatically Generate Trust Properties

"What makes Superman a hero is not that he has power, but that he has the wisdom and the maturity to use the power wisely..." — *Christopher Reeve*

After providing trust verdicts regarding the interactions of the entities, we realized that, generally speaking, in passive testing techniques, there is little consideration of how to extract relevant properties to test. It can be a difficult task to choose what behavior to assure. Moreover, when passive testing techniques are at runtime, the allocation of the resources has a critical role. Therefore, correctly choosing the properties to test becomes a crucial task.

As stated in the introduction, in general terms, the work performed in this chapter corresponds to enhancing current state of the art proposals, specifically for passive testing methods, in order to automatically generate the "test properties". This addresses the remaining open issue as stated in the Section 2.3.

This Chapter is divided as follows: in the Section 4.1, we give some concepts of what Process Mining is and how it can be applied to telecommunication protocols. Later, in Section 4.2 we present our approach to automatically extract trust properties from the obtained *Process Mining (PM) model* and a given specification.

In this chapter, we detail the contribution which is partially published in the work realized in [López et al., 2015c].

4.1   Preliminaries — Process Mining

Process mining (PM) is a research discipline that seeks to discover, monitoring and improve real processes by extracting knowledge from event logs available. This discipline has proven to be a valuable tool for discovery and understanding processes in different contexts, such as business processes in productive industries, clinical processes in health centers and software development processes [van der Aalst, 2011]. With PM, different perspectives of the process can be analyzed: control flow (how the activities are performed), organizational (who run the process), temporal (when and how long the execution takes) and data (what specific characteristics of each execution of the process).

PM algorithms use event logs with varied types of information; while the modeled process can be any of the ones described above, some fields are mandatories: `case id`, `activity name`, `timestamp`, `resource`. Using this information, algorithms can model the sequences of activities in a global model. PM algorithms can handle this data to a greater or lesser extent, with noise, incomplete data or several different manners to run the processes. These algorithms detect sequentiality and parallelism found in the actual execution of the cases, which may differ from planned or expected behavior.

The first algorithm developed is the Alpha Miner (AM) [Van der Aalst et al., 2004], which creates a Petri net from the event log. AM analyzes the traces of every case performed, counting how many times the activities are performed in a particular order (e.g.: activity `B` follows activity `A`). With this analysis the sequentiality and parallelism in the process are determined, and the process model can be created. Additionally, AM identifies the average time between activities, who performed every activity and the times of each one, that allow us to differentiate the performance of each actor in the process. Even though the AM algorithm is simple and intuitive, it cannot handle real logs. Nevertheless, starting from Alpha Miner, several new algorithms have been developed addressing the weakness of the AM, such as A+ (an improvement of AM), Fuzzy Miner (FM), Flexible Heuristic Miner (FHM), etc.

Fuzzy Miner (FM), was the first algorithm to directly address the problems of large numbers of activities and highly unstructured behaviors. FM uses significance/correlation metrics to interactively simplify the process model at desired level of abstraction. When FM takes as inputs the logs and it generates the PM weighted graphs [Günther and Van Der Aalst, 2007].
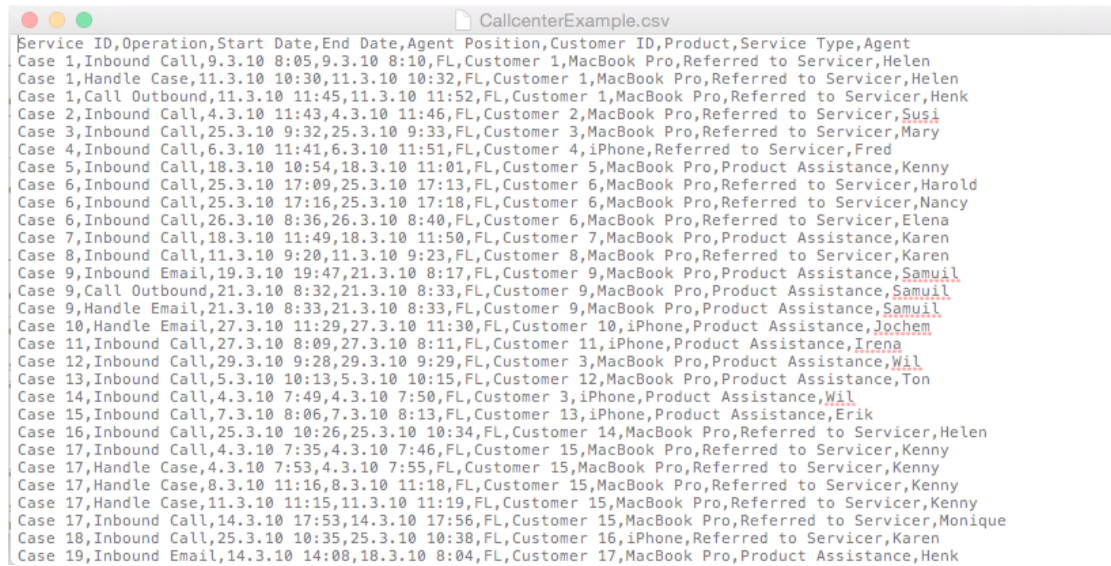
In order to analyze other perspective of the processes, social network analysis algorithms were developed to create the social net behind the event log. Also algorithms to understand the roles (who are performing a set of activities), work teams (who are working together), an how is flowing the activities between actors (handover of work).

Several programs implement these algorithms, the most complete —in the sense of algorithms implemented— is the ProM Framework, an open source software developed by the Process Mining Group, Eindhoven Technical University. Its commercial PM software version is the well known Disco tool [1], developed by Fluxicon. Furthermore, the Disco tool is based on the FM algorithm [Rozinat, 2013].

4.1.1 Process Mining example

For a better understanding of what PM techniques can achieve, in this subsection, we use one of the Disco data examples, to briefly demonstrate the PM capabilities.

The example is a log taken from a real call-center which has been simplified and "anonymized". The file is formatted as a Comma Separated Values (CVS) file. The first row of the file contains the field names. Part of the CVS file is shown in the Figure 4.1.


```
CallcenterExample.csv
Service ID,Operation,Start Date,End Date,Agent Position,Customer ID,Product,Service Type,Agent
Case 1,Inbound Call,9.3.10 8:05,9.3.10 8:10,FL,Customer 1,MacBook Pro,Referred to Servicer,Helen
Case 1,Handle Case,11.3.10 10:30,11.3.10 10:32,FL,Customer 1,MacBook Pro,Referred to Servicer,Helen
Case 1,Call Outbound,11.3.10 11:45,11.3.10 11:52,FL,Customer 1,MacBook Pro,Referred to Servicer,Henk
Case 2,Inbound Call,4.3.10 11:43,4.3.10 11:46,FL,Customer 2,MacBook Pro,Referred to Servicer,Susi
Case 3,Inbound Call,25.3.10 9:32,25.3.10 9:33,FL,Customer 3,MacBook Pro,Referred to Servicer,Mary
Case 4,Inbound Call,6.3.10 11:41,6.3.10 11:51,FL,Customer 4,iPhone,Referred to Servicer,Fred
Case 5,Inbound Call,18.3.10 10:54,18.3.10 11:01,FL,Customer 5,MacBook Pro,Product Assistance,Kenny
Case 6,Inbound Call,25.3.10 17:09,25.3.10 17:13,FL,Customer 6,MacBook Pro,Referred to Servicer,Harold
Case 6,Inbound Call,25.3.10 17:16,25.3.10 17:18,FL,Customer 6,MacBook Pro,Referred to Servicer,Nancy
Case 6,Inbound Call,26.3.10 8:36,26.3.10 8:40,FL,Customer 6,MacBook Pro,Referred to Servicer,Elena
Case 7,Inbound Call,18.3.10 11:49,18.3.10 11:50,FL,Customer 7,MacBook Pro,Product Assistance,Karen
Case 8,Inbound Call,11.3.10 9:20,11.3.10 9:23,FL,Customer 8,MacBook Pro,Referred to Servicer,Karen
Case 9,Inbound Email,19.3.10 19:47,21.3.10 8:17,FL,Customer 9,MacBook Pro,Product Assistance,Samuil
Case 9,Call Outbound,21.3.10 8:32,21.3.10 8:33,FL,Customer 9,MacBook Pro,Product Assistance,Samuil
Case 9,Handle Email,21.3.10 8:33,21.3.10 8:33,FL,Customer 9,MacBook Pro,Product Assistance,Samuil
Case 10,Handle Email,27.3.10 11:29,27.3.10 11:30,FL,Customer 10,iPhone,Product Assistance,Jochem
Case 11,Inbound Call,27.3.10 8:09,27.3.10 8:11,FL,Customer 11,iPhone,Product Assistance,Irena
Case 12,Inbound Call,29.3.10 9:28,29.3.10 9:29,FL,Customer 3,MacBook Pro,Product Assistance,Wil
Case 13,Inbound Call,5.3.10 10:13,5.3.10 10:15,FL,Customer 12,MacBook Pro,Product Assistance,Ton
Case 14,Inbound Call,4.3.10 7:49,4.3.10 7:50,FL,Customer 3,iPhone,Product Assistance,Wil
Case 15,Inbound Call,7.3.10 8:06,7.3.10 8:13,FL,Customer 13,iPhone,Product Assistance,Erik
Case 16,Inbound Call,25.3.10 10:26,25.3.10 10:34,FL,Customer 14,MacBook Pro,Referred to Servicer,Helen
Case 17,Inbound Call,4.3.10 7:35,4.3.10 7:46,FL,Customer 15,MacBook Pro,Referred to Servicer,Kenny
Case 17,Handle Case,4.3.10 7:53,4.3.10 7:55,FL,Customer 15,MacBook Pro,Referred to Servicer,Kenny
Case 17,Handle Case,8.3.10 11:16,8.3.10 11:18,FL,Customer 15,MacBook Pro,Referred to Servicer,Kenny
Case 17,Handle Case,11.3.10 11:15,11.3.10 11:19,FL,Customer 15,MacBook Pro,Referred to Servicer,Kenny
Case 17,Inbound Call,14.3.10 17:53,14.3.10 17:56,FL,Customer 15,MacBook Pro,Referred to Servicer,Monique
Case 18,Inbound Call,25.3.10 10:35,25.3.10 10:38,FL,Customer 16,iPhone,Referred to Servicer,Karen
Case 19,Inbound Email,14.3.10 14:08,18.3.10 8:04,FL,Customer 17,MacBook Pro,Product Assistance,Henk
```

FIGURE 4.1: PM Model Weighted Graph

As you can see, the mapping between the file columns and the necessary fields (case id, activity, timestamp, and resource) from the PM required fields is necessary, given the fact that the column names are not forcibly required to be the same. Using the disco tool, a mapping can be achieved. For this particular example, the mapping is a follows: *Service ID=Case ID; Operation=Activity; Start Date,End Date=Timestamp; Agent=Resource.*

To easily exemplify how process mining can help we show in Figure 4.2 the graph of activities extracted from the example log. In this graph, the process flow starts with the

---

[1] http://fluxicon.com/disco/

green vertex (with the "play" sign), and finishes with the red vertex (with the "stop" sign). The transitions between each activity (vertex) show the number of times those were found in the logs. From this graph, a great quantity of information can be obtained or inferred. In the next section, we discuss some usages of the PM techniques.
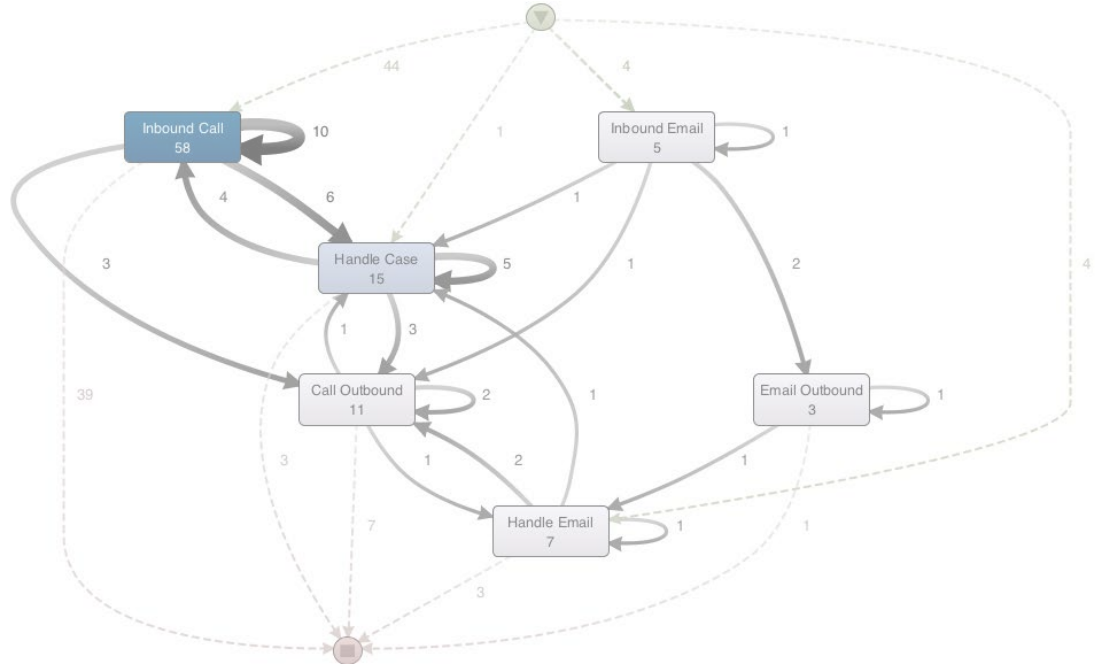
FIGURE 4.2: Generated PM graph from example input log

4.1.2   Process Mining Beyond Process Discovery Related Work

Process mining considers the activity sequences and times, it is capable of control-flow discovery, conformance checking and enhancement of the processes. PM algorithms are also capable to show organizational, case and time perspectives that also play an important role. Another distinctive feature of PM is that it considers parallelism and concurrence for the analysis. From the point of view of using process mining for model and analyze Internet traces, and given the newly nature of this techniques and tools, we can cite the work published in 2013 by Poggi et al [Poggi et al., 2013]. In there, the authors analyze a set of e-commerce Website logs, in order to extract business models from real-life Web data, obtaining a relationship between pages, exit points and critical path taken by costumers. Notice that the main objective of this work is to analyze the process from the e-commerce business point of view and not for analyzing the behavior of the web protocol.

Some theoretical work was made by Houi et al. [Houy et al., 2011] where they study how PM techniques and tools should be applied in large sets of interdependent inter-organizational processes instead the limited set used in the precedent studies.

We also may cite the works like, [Azzini and Ceravolo, 2013] that uses the process mining techniques to discover the interactions between social media, to improve recommender systems. This interesting work inspired our approach in the sense that process mining techniques and algorithms can be used for purposes other than just process discovery.

However, though these works are interesting, to the best of our knowledge, there are no systems or frameworks that use process mining to extract trust properties to be applied within monitoring techniques.

4.2 Automatic Extraction of Trust Properties

As described in Section 3.2, the trust properties are descriptions of expected behaviors. In many occasions, the trust properties are meant to be designed by experts in each particular protocol. However, manually going through an RFC and trying to recognize which requirements might be interesting to enforce as trust properties can be somewhat difficult. Furthermore, when manually extracting all possible trust properties derived from the protocol standard, one can model trust properties that will not be relevant to check; for example trust properties that will always produce pass (or fail) verdicts, and such properties will be just occupying resources unnecessarily. One basic constraint regarding on-line monitoring solutions is to provide the verdicts as soon as possible, this is exactly the value behind them. For that reason, having trust properties that will always have the same result, wasting resources can have a negative impact in the monitoring system. Therefore, the main question that arises is — how can trust properties be extracted automatically? We propose the use of PM techniques in order to automatically extract trust properties from observed *faults*, i.e., untrustworthy behaviors in the communication protocols' interactions.

As explained in the Section 4.1, the PM techniques take as input the logs from the systems and the output of the process mining is a directed and weighted graph, an ordered triplet, $G = (V, A, w)$, where V is a set of elements called vertices, A is a set of ordered pairs of vertices called arcs, and w is the weight function such that $w : A \mapsto \mathbb{Z}^+$, the positive weight function. We refer to this graph as the PM model. The vertices, called activities in the PM model, represent network packets, inputs or outputs for a given communication protocol. The arcs represent the transition between one network packet to the other, the process flow in PM. Finally, the weights represent the number of times the associated process flow (arc) is observed in the input logs.

Our approach relies on *comparing* the PM model with a specification model. The specification model can be obtained in two different manners; on the one hand it can be

obtained by manually deriving this model from a protocol RFC, and on the other hand it can be obtained via a previously generated PM model. Before analyzing how the *comparison* between the specification model and the PM model is done, important constraint, regarding the PM model and the specification model are described:

1. Both models have the same vertex names, due to the fact that in the PM the activities are explicitly enumerated, and therefore no unknown vertices can be obtained in the PM model. As a consequence, the sets of vertices and arcs are labeled equally and they can be compared without the need of an explicit mapping

2. Both models have a vertex with in-degree equals to zero (no incoming arcs), source vertex (start of a process). In other words, both models are "rooted" graphs

3. Both models have a vertex with out-degree equals to zero (no outgoing arcs), sink vertex (end of a process)

4. The arcs' associated weights in the PM model denote the number of occurrences from one activity to the other and thus, these weights depend on the observed processes in the input logs of the PM. For this reason, weights cannot influence how both models can be *compared*

5. Vertices in the models are activities; the meaning behind a vertex which is not present in the PM model is that the activity was not observed during the execution time of the logs. Therefore, the missing vertices and associated arcs in the PM model are not considered as a *fault*

6. An arc which is not found in the specification model can have two meanings: if the specification model was generated via a previous PM model, the specification might not be complete and the missing arc can be incorporated into the specification; the second meaning is that a *fault* in the process flow occurred

To *compare* the specification model and the PM model, we define the comparison as a conformance relation. Intuitively, that conformance relation between the two models can be done using the definition of graph isomorphism. It is highly important to note that, taking into account the previously mentioned constraints, we note that knowing if both models are isomorphic or not, does not yield useful information for our model and purposes. Then, we define our conformance relation based on those constraints as the partial order relation: "the PM model should be a subgraph of the specification model, ignoring the associated weights of the arcs". This is formally expressed as:

Definition 4.1. Let $\mathcal{S} = (V_S, A_S, w_S)$ be the specification model, where V_S is the set of vertices, A_S the set of arcs, and w_S the positive weight function of $\mathcal{S}$. Let

$\mathcal{P} = (V_P, A_P, w_P)$ be the PM model, where V_P is the set of vertices, A_P the set of arcs, and w_P the positive weight function of $\mathcal{P}$. $\mathcal{P}$ is said to conform to $\mathcal{S}$ if and only if $\mathcal{P}$ is a subgraph of $\mathcal{S}$, ignoring their weights. In a formal notation: $\mathcal{P} \simeq \mathcal{S} \Leftrightarrow \forall (a, b) \in A_P \implies (a, b) \in A_S$

Given the constraint 1, we note that $V_P \subseteq V_S$ is guaranteed. If $\mathcal{P} \not\simeq \mathcal{S}$ (the PM model does not conform to the specification), it implies at least a *fault* is present in the PM model, and therefore an untrustworthy behavior observed. The other possibility would be that the specification model is not correct. Nevertheless, throughout this study, we assume the correctness of the specification model; it is out of the scope of this work the validation of it. As consequence of the Definition 4.1, we note the statement in the following corollary:

Corollary 4.2. *If $\mathcal{P} \not\simeq \mathcal{S}$, there exists elements in the set of arcs of the PM model that are not present in the set of arcs of the specification model. Each element in the set of arcs not present in the specification are considered faults. In a formal notation: $\mathcal{P} \not\simeq \mathcal{S} \Leftrightarrow A_P \neq A_P \cap A_S$. Hence, the set of faults $F = \{(a, b) : (a, b) \in A_P \setminus A_P \cap A_S\}$.*

Proof. Let us derive the very simple proof of our previously stated corollary. Let us begin by proving that $A_P \neq A_P \cap A_S \implies \mathcal{P} \not\simeq \mathcal{S}$.
if $A_P \neq A_P \cap A_S$, then $A_P \setminus A_P \cap A_S \neq \emptyset$
Let $(c_1, c_2) \in A_P \setminus A_P \cap A_S$
$(c_1, c_2) \in A_P$ // Given the set difference definition, it must be present in this set.
$(c_1, c_2) \notin A_S$ // Otherwise it will not be present in the difference.
if $(c_1, c_2) \in A_P \implies (c_1, c_2) \in A_S$ does not hold. Therefore, $\mathcal{P} \not\simeq \mathcal{S}$ $\square$
Let us continue by proving that $\mathcal{P} \not\simeq \mathcal{S} \implies A_P \neq A_P \cap A_S$
if $\mathcal{P} \not\simeq \mathcal{S} \implies \exists (c_1, c_2) \in A_P \wedge (c_1, c_2) \notin A_S$
$A_P \neq A_P \cap A_S$ // Since (c_1, c_2) exists in A_P and not in A_S $\square$
Having proved both implications, we can conclude that:
$\mathcal{P} \not\simeq \mathcal{S} \Leftrightarrow A_P \neq A_P \cap A_S$ $\square$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

In order to extract the trust properties automatically, we will use the known PM model faults. Knowing which process faults can occur, we can generate a trust property for each of those untrustworthy behaviors in order to monitor them. At a first glimpse, composing paths from the *start* vertex to the *end* vertex which include the largest amount of faults can be the solution. Nevertheless, this approach will be incorrect due to the fact that all possible combinations need to be enumerated in order to guarantee that all the untrustworthy behaviors are detected independently from the process flow. By individually taking the faults, no untrustworthy behaviors can go undetected. We

also aid the performance of the checking tool, since the combinations in the paths can easily exceed the number of individual faults. In short, to get verdicts as fast as possible, we use on-line monitoring systems, and the less resources invested to guarantee the trustworthy behavior of entities, the better.

Having established on which elements we will generate the trust properties, it is important to note that the concept of these elements are faults and trust properties are expressed as expected behaviors. As a result, both concepts have opposite logical reasoning, one looks for expected behaviors and the other knows how a fault must be like. Note that the concept of fault in trust properties is not considered in our approach and the feasibility to include this into our concepts is outside of the scope of this thesis. Then, how do the trust properties can be generated automatically? The strategy is to generate what we want to enforce; for each fault, we generate a set of trust properties for each of the allowed behaviors in the specification model (arcs in the specification model), which have the same source vertex as the fault. By generating the trust properties as stated before, not only the particular fault will be enforced, all other behaviors which are not allowed after the enforced activity (source vertex) will rise fail verdicts, not only the one observed. One might conclude that only the specification model is necessary to derive such trust properties. Nevertheless, as stated before, not all specified behaviors are necessary to enforce (is useless to have an always-pass verdict, for instance) and the contribution of the PM model is to discover which of those faults can actually take place based on the system's logs.

After having the clear strategy how the trust properties can be generated from the conformance relation between the models, we need to introduce another concept in order to generate the trust properties in our proposed language. It is important to note that the PM activities and packet prototypes have a one-to-one relationship, they both describe the packet required fields. Therefore, we define:

Definition 4.3. The mapping function $\mathcal{T}$, is a function such that each activity in the PM, i.e., vertex, is transformed into a packet prototype. $\mathcal{T}$ is formally defined as: $\mathcal{T} : V_S \mapsto Prototype$

Where V_S is the set of vertices in the specification model, that is, the set of activities defined in the PM configuration, and *Prototype* is a formatted text string according to our language's grammar, as shown in Section 3.2. For example, the transformation of the activity, query type "MX" to our language will include two atoms, testing that the flag response is not set and that the query type is MX, that is $\mathcal{T}(query \setminus MX) = p.flags.response = 0 \wedge p.queries.type = MX$

Using our definitions we can now proceed to present the trust property generation from the specification and PM models. The algorithm is shown in the Algorithm 3. The algorithm is pretty straight forward, first it creates the set of trust properties as an empty set, the property count (i) is initialized and the set of faults F is computed (as the arcs present in the PM model and not in the specification). Then, for each fault (or arc not present in the specification model), a set of properties is generated. The generated set of properties puts for all of them the causes as the transformation of the source vertex. The conclusions are derived from all the sink vertices on the specification model. Finally, note that since the trust properties have the inverse logic of the arcs which are considered faults, repeated sets of trust properties can appear for arcs with a common source. That is the reason why the algorithm considers deleting arcs with common source after the trust property generation.

Algorithm 3: Trust property generation algorithm

Input: $S = (V_S, A_S, w_S), \mathcal{P} = (V_P, A_P, w_P)$
Output: trustProperties
trustProperties $\leftarrow \varnothing$;
i $\leftarrow 0$;
$F \leftarrow A_P \setminus A_P \cap A_S$;
foreach *varc* $\in F$ **do**
    cause $\leftarrow$ "$\forall_x (`\mathcal{T}(source(varc))`) \rightarrow \exists_{y>x}$";
    foreach *arc* $\in arcsWSrc(source(varc), A_S)$ **do**
        consequence $\leftarrow$ "($`\mathcal{T}(sink(arc), A_S)`$)";
        i $\leftarrow$ i $+ 1$;
        pName $=$ "$\Phi`i`$ $=$";
        trustP $\leftarrow$ concat(pName, cause, consequence);
        add(trustP, trustProperties);
    end
    foreach *arc* $\in arcsWSrc(source(varc), F)$ **do**
        deleteFrom(F, arc);
    end
end

4.2.1 Automatic Trust Property Extraction Algorithm Usage

In this subsection, we use a running example, to better demonstrate use the previously presented approach and algorithm.

Let us consider the partial specification model of the DNS protocol (RFC 1035 [Mockapetris, 1987b]), $\mathcal{S}$, represented by Figure 4.3, and an assumed PM model $\mathcal{P}$ for a DNS resolver, represented by Figure 4.4. Also, let us consider the following *transformation function*, manually constructed by a DNS protocol expert based on the PM activities

and our language syntax:

$$\mathcal{T}(query \setminus A) = x.flags.response = 0 \wedge x.queries.type = A$$

$$\mathcal{T}(query \setminus AAAA) = x.flags.response = 0 \wedge x.queries.type = AAAA$$

$$\mathcal{T}(response \setminus A) = y.flags.response = 1 \wedge y.queries.type = A$$

$$\mathcal{T}(response \setminus AAAA) = y.flags.response = 1 \wedge y.queries.type = AAAA$$
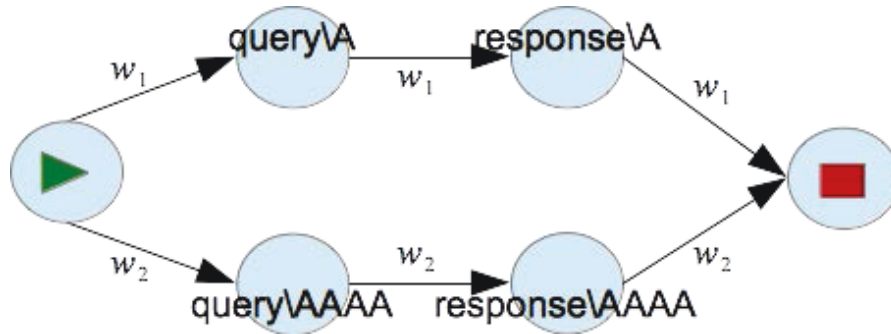
FIGURE 4.3: Specification Model Weighted Graph
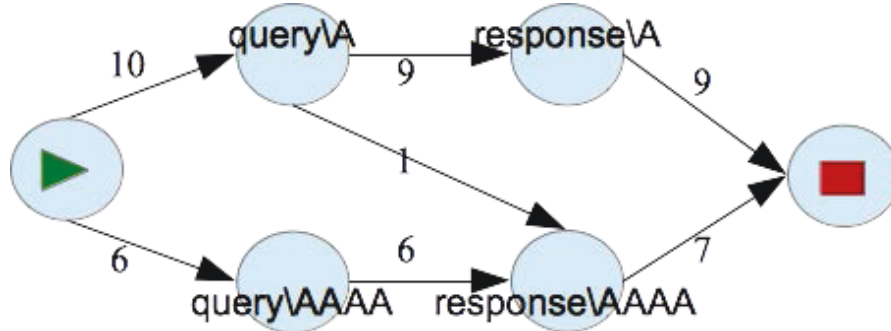
FIGURE 4.4: PM Model Weighted Graph

We know that $\mathcal{P} \not\simeq \mathcal{S}$, since $A_P \setminus A_P \cap A_S \neq \varnothing$. Furthermore, $A_P \setminus A_P \cap A_S = \{(query \setminus A, response \setminus AAAA)\}$. This information can be useful to add to $\mathcal{S}$ if it was derived using a previous PM model. However, this is out of the scope of the example and moreover, we are interested in the automatic trust property generation. If we input the two models to the Algorithm 3, the trust properties generated by our algorithm will be the following:

$$\Phi 1 = \forall_x (x.flags.response = 0 \wedge x.queries.type = A)$$

$$\rightarrow$$

$$\exists_{y>x} (y.flags.response = 1 \wedge y.queries.type = A)$$

Φ1 is the single trust property generated by our algorithm. This trust property will be enforcing the allowed transitions where the untrustworthy behavior was observed in the PM model as explained before.

As shown in the previous example, our proposed method is able to automatically derive meaningful trust properties. As demonstrated, our approach does not depend on a specific communication protocol. Our approach is generic and extensible. The only requirements necessary are: i) having a specification model which can be obtained from previous PM observations; ii) having a generated a PM model using well-known techniques as presented in Section 4.1; iii) having a mapping function for the PM activities to network packet prototypes; since both PM activities and network packet prototypes have a one-to-one relationship, deriving this function does not represent a challenge. All these requirements are highly feasible to obtain and the benefits of automatically generating the trust properties in order to enforce trustworthy interactions has a greater value.

Chapter 5

Enhancements for On-line Network Monitoring Systems

"Perfection is not attainable, but if we chase perfection we can catch excellence." —
Vince Lombardi

In Chapter 3, we presented our approach to provide behavioral feedback by using distributed on-line network monitoring. The approach deeply relies on the distributed on-line network monitoring tools to test the trust properties.

After doing the experimental results, we noticed that our approach had some elements that could be improved. In fact, we noticed that the concept of *timeout* to avoid resource starvation was mandatory. Furthermore, given the nature of real time feedback, great performance of the evaluation needs to be guaranteed. In this chapter, we propose enhancements to the current On-line Network Monitoring Systems.

In the introductory chapter we exposed that, in general terms, the work performed in this chapter corresponds to enhancing current state of the art proposals, specifically for passive testing methods, in order to provide ease of usage of our approach and to have a scalable evaluation of the properties. This addresses the remaining open issue as stated in the Section 2.3.

This chapter is divided as follows: In Section 5.1, we give basic concepts and notions about Extended Finite State Machines and Extended Finite State Automata,(EFSA) employed in future sections of this chapter. Later, in Section 5.2 we detail the elements that to be enhanced. In Section 5.3, we expose the approach for enhancing the property evaluation process. Finally, in Section 5.4, we propose a new language to adopt all the new concepts and to ease the usage. At a first glimpse one might be tempted to avoid

having a language representation to ease the usage if having an EFSA to describe and test the behaviors. Nevertheless, the language increases the usability. Furthermore, since the basic notions of atoms, prototypes, etc., are kept in both the language and the EFSA both concepts are coherent and throughout this chapter we present this.

In this chapter, we expose the work which was partially published in the following publications: [López et al., 2014b, 2015b].

5.1 Preliminaries — Extended Finite State Machines and Extended Finite State Automata

In this section, we introduce the necessary concepts of an Extended Finite State Automata (EFSA). First, we use the definition of Extended Finite State Machines (EFSM) to help define the EFSA model. The EFSA model has been used for different applications in the testing domain, such as, [El-Fakih et al., 2008; El-Fakih et al., 2003]. Furthermore, to test telecommunication protocols, the same model has been used as shown in [Kushik et al., 2014]; additionally, the EFSM model definition is extracted from the previously exposed work. Let us start by introducing the EFSM model in the following subsection.

5.1.1 EFSM Model

A *finite state machine* (*FSM*), or simply a *machine* as it often appears in the literature, is a 5-tuple $\mathbf{S} = \langle S, I, O, h_S, S' \rangle$, where S is a finite nonempty set of states with a nonempty subset S' of initial states; I and O are finite input and output alphabets; and $h_S \subseteq S \times I \times O \times S$ is a *behavior* (*transition*) *relation*. An output function is defined as $out : S \times I \mapsto O$, the output symbol obtained when applying a state-input pair. If $|S'| = 1$ then the machine is *initialized*, otherwise it is *non-initialized* (*weakly initialized*). An FSM is *nondeterministic* (NFSM) if for some pair $(s, i) \in S \times I$ there exist several pairs $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_S$, otherwise $\mathbf{S}$ is *deterministic*. If for each pair $(s, i) \in S \times I$ there exists $(o, s') \in O \times S$ such that $(s, i, o, s') \in h_S$ then the FSM is *complete*, otherwise it is *partial*. If for each triplet $(s, i, o) \in S \times I \times O$ there exists at most one state $s' \in S$ such that $(s, i, o, s') \in h_S$ then the FSM is *observable*, otherwise it is *nonobsevable*. The FSM behavior is extended to input sequences (by sequentially applying the inputs and following the transitions). For example, the output function defined over the input sequence of length n, $\alpha \in I^* = I_1 I_2 ... I_n$, and the state s as an output sequence $\omega \in O^*$ by sequentially applying I_j to the output function at all intermediate states (following the transitions).

Given an FSM $\mathbf{S} = \langle S, I, O, h_S, S' \rangle$, two states $s_1, s_2 \in S$ are *compatible* if for each input sequence $\alpha \in I^*$ the sets of output responses at these states to α coincide, i.e. $out(s_1, \alpha) = out(s_2, \alpha)$. Two states are *distinguishable* if there exists an input sequence α such that α is a defined input sequence at both states s_1 and s_2 and $out(s_1, \alpha) \neq out(s_2, \alpha)$. The FSM $\mathbf{S}$ is *reduced* if its states are pair-wise distinguishable. States s_1, s_2 of $\mathbf{S}$ are *separable* if there exists an input sequence $\alpha \in I^*$ such that $out(s_1, \alpha) \cap out(s_2, \alpha) = \varnothing$; in this case, α is a *separating* sequence of states s_1 and s_2. If there exists an input sequence α that separates every two distinct states of the set S', then α is a *separating* sequence for the set S'.

An extended finite state machine (EFSM) [Petrenko et al., 2004] A is a pair (S, T) of a set S of states and a set T of transitions between states, such that each transition $t \in T$ is a tuple $(s, i, o, P, v_p, o_p, s')$, where $s, s' \in S$ are the starting and final states of a transition; $i \in I$ is an input with the set D_{inp-i} of possible vectors of corresponding input parameter values, $o \in O$ is an output with the set D_{out-o} of possible vectors of output parameter values; P, v_p, and o_p are functions, defined over input parameters and context variables V. The context variables are variables which can hold different valuations v, and the associated set of valuations is denoted as D_V also called a context of A. A configuration of A is a tuple (s, v) of state and context. A designated configuration that the machine usually starts from is called the *initial* configuration. By definition, $P : D_{inp-i} \times D_V \longrightarrow \{True, False\}$ is a predicate where D_V is the set of context vectors; $o_p : D_{inp-i} \times D_V \longrightarrow D_{out-o}$ is an *output parameter update* function; $v_p : D_{inp-i} \times D_V \longrightarrow D_V$ is a *context update* function.

According to [Petrenko et al., 2004], we use the following definitions. Given an input i and a vector $\mathbf{p} \in D_{inp-i}$, the pair $(i, \mathbf{p})$ is called a *parameterized input*; if there are no parameters for the input i then i is a *non-parameterized* input. A sequence of parameterized inputs (possibly some of them are non-parameterized) is called a *parameterized input sequence*. A context vector $\mathbf{v} \in D_V$ is called a *context* of A. A $configuration$ of A is a pair $(s, \mathbf{v})$. Usually, the initial state and the initial configuration of the EFSM are given; thus, given a parameterized input sequence of the EFSM, we can calculate a corresponding parameterized output sequence by simulating the behavior of the EFSM under the input sequence starting from the initial configuration.

An EFSM is *consistent* if for each transition at state s with input i, every element in $D_{inp-i} \times D_V$ evaluates exactly one predicate to true among all predicates guarding the different transitions with the starting state s and input i; in other words, the predicates are mutually exclusive and their disjunction evaluates to true. An EFSM A is *completely specified* if for each pair $(s, i) \in S \times I$, there exists at least one transition at state s with the input i. The authors of most papers develop test derivation strategies for consistent

and completely specified EFSMs. However, such EFSMs are rarely met when building protocol specifications at high abstraction levels.

The equivalence and distinguishability relations for EFSM configurations are defined similar to those over FSM states. Two initialized EFSMs are *compatible* if their initial configurations are compatible. Differently from FSMs, we still lack necessary and sufficient conditions for establishing whether even two complete and consistent EFSMs are equivalent. Two states of an EFSM are *separable* if there exists a (parameterized) input sequence such that at these states the sets of parameterized output responses of the EFSM to this input sequence do not intersect (for any values of context variables). In other words, if two states s and s' of the EFSM are separable then each two configurations at these states are separable.

When the values of each context variable and of each input parameter are finite, an EFSM A can be unfolded to an equivalent FSM, written $FSM_{sim}(A)$, by simulating its behavior with respect to all possible values of context variables and input vectors. The equivalence means that the set of traces of the FSM coincides with the set of parameterized traces of the EFSM. Given a state s of EFSM A, a context vector $\mathbf{v}$, an input i and the vector $\mathbf{p}$ of input parameters, we derive the transition from configuration $(s, \mathbf{v})$ under input $(i, \mathbf{p})$ in the corresponding FSM. We first determine the outgoing transition $(s, i, o, P, v_p, o_p, s')$ from state s where the predicate P is true for the input vector $\mathbf{p}$ and the context vector $\mathbf{v}$, update the context vector to the vector $\mathbf{v}'$ according to the assignment v_p of this transition, determine the parameterized output $(o, \mathbf{w})$ and add the transition $((s, \mathbf{v}), (i, \mathbf{p}), (o, \mathbf{w}), (s', \mathbf{v}'))$ to the set of transitions of the FSM $FSM_{sim}(A)$. The number of states of the obtained FSM equals the number of different configurations $(s, \mathbf{v})$ of the EFSM that are reachable from the initial configuration. If an EFSM is consistent and completely specified, the corresponding FSM is complete and deterministic. Two EFSMs are equivalent if and only if their corresponding FSMs are equivalent [Faro and Petrenko, 1990]. When the specification domain of some context variable and/or some input parameter is infinite or the number of generated transitions becomes huge, the EFSM behavior is simulated up to the given number of transitions or up to the given length of input sequences.

5.1.2 EFSA Model

As stated before, we used the EFSM model to help defining the EFSA model. Departing from the EFSM definition, we define an EFSA also as a pair (S, T, S_i, S_f) of a set S of states, S_i a non-empty subset of S of initial states, S_f a non-empty subset of S of "final" or "accepting states", and a set T of transitions between states, such that each

transition $t \in T$ is a tuple $(s, i, o, P, v_p, o_p, s')$ as defined in an EFSM. A restriction on the transitions is defined for an EFSA, only an input or output can be found in the transitions and not both at the same time.

While there are other definitions of the "*same*" object in the literature, as the one shown in [Smith et al., 2008], we prefer the definition previously shown due to its clarity regarding the predicates and updating functions needed for our proposed auxiliary data structure.

In this section, we have presented the Extended Finite State Automata model. Later, in the Section 5.3 we use an auxiliary EFSA data structure to enhance the evaluation of trust properties.

5.2 Distributed On-line Network Monitoring — Identifying Areas of Improvement

In this Section we identify the areas of improvement of state of the art solutions of on-line network monitoring. This was motivated after we extensively worked on our approach presented in Section 3. Since our approach heavily relies on on-line network monitoring systems and its correct and scalable functioning, the need to enhance those features as much as possible is a must. Therefore, we seek to improve our proposed solutions in these areas.

5.2.1 Language lack of expressiveness

As stated before, a distributed on-line network monitoring approach requires the correlation between different packets. There are some important constraints in the system's behavior. Those constraints are the following:

- A property has a set of conditions that must be met. Conditions over the packet itself or previously stored packets (dependencies)

- In order to avoid resource starvation (resource leaks, memory and processor), our proposed strategy is to incorporate stored packet timeouts. A timeout might produce a verdict in the system, a timeout fail

- Some exceptions to the timeouts are needed. For instance, some packets might be kept until a condition is met, e.g., when a new sample of a packet prototype is found

- For each packet, all conditions must be tested, since, in on-line network monitoring each packet could arrive to the testing system at any given state

Our current language does not take all these constraints into account. This motivates us to improve our current language in order to be able to express timeouts and hold conditions. Changing the semantics of some quantifiers can also be useful to correctly express in the language what we need. Another aspect that could be improved is to have a better property evaluation. The motivation behind this is due to the fact that on-line monitoring can be a highly costly process.

5.2.2 Effective language evaluation

Considering the critical constraints about time processing in on-line network monitoring, we note that the formula can be evaluated using an auxiliary data structure. We have chosen to use an Extended Finite State Automata (EFSA). Works like [Smith et al., 2008], clearly expose the urge to have great performance on the properties' evaluation. These properties use a regular expression language. More than that, they also use a state model to efficiently test these properties.

Please note that, this does not imply that we need a model to describe the system. By using a state model to evaluate the properties, a better performance and expressiveness is obtained as a result. The previously described model fits the requirements of on-line monitoring systems. We explain how this model fits in the following.

State models have been widely used for scalable evaluation of other properties / languages. For example, a regular expression parser, without using a state machine evaluation, for evaluating disjunctions, it will have to exhaust all the OR (options) possibilities until it matches the proper one, for instance:

$$\text{Allowed} = \wedge(a|e|i|o|u) + (b|c|d|f|g|h|j|k|l|m|n|p|q|r|s|t|v|w|x|y|z) + \$$$

Assuming a generic evaluation algorithm, without using state machines for evaluation, the disjunctions (or options) would have to be evaluated one by one, and using the extreme cases where the input string can be "uuuzzz", 72 (4*3 + 20*3) wrong comparisons will be performed. On the other hand, with a Finite State Automata used to evaluate the input string, at the initial state, all vowels are added as inputs and the transitions are to the same initial state. In the same state, all consonants are added as inputs and they all have their transitions at the final / accepting state. While evaluating the same string, when observing the letter "u" as input in the initial state, the transition is done

to the same initial state. The algorithm for evaluating with the transition functions is quite simple.

For our particular case, let us assume that we have this property to check in a distributed on-line monitoring system:

$$\Phi = \forall x (x.dns.flags.QR = 0 \land x.PO = \text{``ADS''}) \rightarrow$$
$$\exists y > x (y.dns.flags.QR = 1 \land y.PO = \text{``ADS''} \land y.dns.ID = x.dns.ID) \rightarrow$$
$$\exists a > y (a.dns.flags.QR = 0 \land a.PO \neq \text{``ADS''}) \rightarrow$$
$$\exists b > a ((b.dns.flags.QR = 1 \land b.PO \neq \text{``ADS''} \land b.dns.ID = a.dns.ID \land$$
$$y.dns.answers = b.dns.answers)$$

Note that our generic algorithm for evaluating a formula presented in Chapter 3, can be enhanced performance-wise oriented. One might think than when evaluating the last packet prototype of a given formula has the worst performance. However, since a packet can match several prototypes, all prototypes need to be checked against each packet. On the other hand, if one could keep track of the already matched predicates (or atoms), then, this would improve the testing procedure. For instance, in the property described above (Φ), for a packet that matches the characteristics described by the "prototype b" (to be tagged as a "b" type). Using the generic algorithm presented in Section 3.4.4, eight comparisons / atomic tests (or atoms) will be made to conclude that the packet matches the "prototype b"; out of which four of those evaluations will be repeated.

If a state model derived from the formula existed, the evaluations can be "tracked" and the problems previously described do not apply. The condition for using an associated state model is that the state model can accept the same inputs as the described formula. If this condition is satisfied, then, using a state model as an auxiliary data structure is better suited for properties' evaluation.

A complexity analysis of both algorithms should reveal which of those is the better (we will discuss this later). More than that, when different sets of properties are analyzed with the use of a combined EFSA, all properties that have common predicates (or atoms), for instance, several properties might start with "for each request". In fact, the algorithm for generating the EFSA will strongly influence the performance of the evaluation algorithm. The "naive" algorithm for generating the EFSA can improve the performance. However, exactly the composition can lead to an EFSA, such that, each state will have fewer and only necessary predicates to be evaluated. The goal

is to evaluate at the beginning the most common conjunction of predicates of all the properties.

Taking into account the proposed way to generate the EFSA, we note some relevant facts:

- The more repeated transitions are added first to the predicates to evaluate, thus, they are evaluated first by our state evaluation algorithm.

- If no repeated predicates (atoms) are found in the formulae, the transitions number will be the number of atoms.

In the subsequent sections, we describe our proposed solutions for the stated challenges exposed in this section.

5.3 Enhancing On-line network monitoring system's expressiveness and scalability with EFSA

The evaluation process in an on-line monitoring system consists in evaluating if each packet complies the desired trust properties we need to check. Therefore, a scalable way for the evaluation algorithm is perhaps the biggest requirement. The trust properties have a set of conditions (atoms) that a packet's data needs to match against constant values or against the values of previously stored packets, as explained before. After matching the packet's conditions, checking if the matched packet completed a trust property is necessary. In order to provide verdicts regarding the trust properties we developed a first approach using the algorithm presented in our work at Section 3.4.4. The worst-case analysis of the time work performed by the previously mentioned algorithm is expressed by the following equation:

$$
T(eval\_all\_prots) = 3N_p + \sum_{i=1}^{N_p} NPA_i + \sum_{i=1}^{N_p} 2NPD_i + \sum_{i=1}^{N_p} (N_p - i)QL_i
$$
$$
+ \sum_{i=1}^{N_p} ((N_p - i)QL_i * NDA_i)
$$

(5.1)

Where N_p is the number of prototypes in the formulae, NPA_i is number of atoms that require no dependencies of the *ith* prototype, NPD_i is the number of dependencies of the *ith* prototype, QL_i is the length of the queue of stored packets of the *ith* prototype,

and NDA_i is the number of atoms that require dependencies of the *ith* prototype. The experimental results achieved with the first algorithm are good. However, due to the on-line monitoring constraints, we are required to create the most scalable algorithm for the evaluation of trust properties. Based on the time complexity analysis of our algorithm, we note that the term that dominates the equation is the term $\sum_{i=1}^{N_p}((N_p - i)QL_i * NDA_i)$; from this term we can observe that atoms (conditions/comparisons) need to be checked against the stored packet queues and this is being repeated up to $N_p - i$ times. In order to create a scalable algorithm, we need to avoid re-doing any atomic tests on a packet since the result of the atomic test will not change for each packet.

In order to improve the algorithm, known techniques are applied. First, we propose to make use of a data structure that will aid avoiding repeated checks. In addition to that, we propose to keep track of previously visited packets in the stored queue to avoid re-visiting packets, which did not match previous tests.

We have chosen to use a tree-structured (single rooted) extended finite state automata (EFSA) as the structure for the scalable evaluation. The reason is that we propose evaluating the packets by doing the atomic test once and to keep track of the already tested atoms (a transition model based on predicates) and then, when a packet is found to match a prototype (at some accept state), execute some actions (updating functions), for instance, storing the packet on a queue or reporting a property verdict. These types of models have become popular to achieve scalable algorithms, for example, several works like [Becchi et al., 2009; Smith et al., 2008] use different types of automata, finite, non-deterministic, hybrid and extended to evaluate a regular expression language to achieve a scalable deep packet inspection.

Our target is to generate the EFSA from the necessary prototypes. The strategy in order to avoid repeating atomic tests is to generate transitions from the root state, adding predicates of the atomic tests which are more common at the beginning and creating related atomic tests (atomic tests which are part of the same prototype) along the same path; for the next prototypes, uncommon atoms will be branches added at the current state after following the common transitions. Therefore, our algorithm to generate the EFSA relies on three principal actions: i) comparing each atom and add a count of how many times it appears in the formulae; ii) sorting the prototypes putting first the ones containing the most common atoms, then, do a nested-sorting according the second most common atom, and so on; iii) finally, going along the path of the EFSA creating new nodes branching with its respective transitions based on the atoms or just following the already existing ones (starting from the root) and adding the proper updating functions.

The algorithm to generate our EFSA based on the formulae prototypes can be found in the Algorithm 4.

Algorithm 4: EFSA generation

Input: Prototypes $\mathcal{P}$, State root
Output: EFSA
foreach *atom* $\in \mathcal{P}$ **do**
   | repeated[atom] $\leftarrow$ count(atom, $\mathcal{P}$);
end
maxRepeated $\leftarrow$ max(repeated);
sortedPrototypes $\leftarrow \mathcal{P}$;
repeat
   | atom $\leftarrow$ atomRepeated(maxRepeated, $\mathcal{P}$);
   | sortedPrototypes $\leftarrow$ sortWRT(sortedPrototypes, atom)   `/* nested-sorting */`
   | repeated[atom] $\leftarrow$ -repeated[atom];
   | maxRepeated $\leftarrow$ max(repeated);
until *maxRepeated >1*;
current $\leftarrow$ root;
foreach $P \in$ *sortedPrototypes* **do**
   | **while** *(atom $\leftarrow$ higerCountAtom(P)) > 0* **do**
   | | **if** *atom $\notin$ transitions(current)* **then**
   | | | addTransition(current, atom);
   | | **end**
   | | delete(atom,P);
   | | doTransition(current, atom);
   | **end**
   | addUpdatingFuncs(current, selectFuncs(P))   `/* appropriate for the prototype */`
end

Example of a EFSA generation: let us consider the trust property "For all responses from an authoritative DNS server, all future responses from other points of observation are the same replies from the authoritative DNS server if the queries are the same". The trust property as shown before is:

$$\Phi = \forall x (x.dns.flags.QR = 0 \wedge x.PO = \text{"ADS"}) \rightarrow$$
$$\exists y > x (y.dns.flags.QR = 1 \wedge y.PO = \text{"ADS"} \wedge y.dns.ID = x.dns.ID) \rightarrow$$
$$\exists a > y (a.dns.flags.QR = 0 \wedge a.PO \neq \text{"ADS"}) \rightarrow$$
$$\exists b > a ((b.dns.flags.QR = 1 \wedge b.PO \neq \text{"ADS"} \wedge b.dns.ID = a.dns.ID \wedge$$
$$y.dns.answers = b.dns.answers)$$

Using our approach, the prototypes found in the previous formalized trust property are:

$$p1 \leftarrow p.dns.flags.QR = 0 \wedge p.PO = \text{``ADS''}$$

$$p2 \leftarrow p.dns.flags.QR = 1 \wedge p.PO = \text{``ADS''} \wedge p.dns.ID = p1.dns.ID$$

$$p3 \leftarrow p.dns.flags.QR = 0 \wedge p.PO \neq \text{``ADS''} \wedge p.dns.queries = p1.dns.queries$$

$$p4 \leftarrow p.dns.flags.QR = 1 \wedge p.PO \neq \text{``ADS''} \wedge p.dns.ID = p3.dns.ID$$
$$\wedge\, p.dns.answers = p2.dns.answers$$

As an example the list of compared atoms will be:

$$p.dns.flags.QR = 0 \leftarrow 2$$
$$p.PO = \text{``ADS''} \leftarrow 2$$
$$p.dns.flags.QR = 1 \leftarrow 2$$
$$p.dns.ID = p1.dns.ID \leftarrow 1$$
$$p.PO \neq \text{``ADS''} \leftarrow 2$$
$$p.dns.queries = p1.dns.queries \leftarrow 1$$
$$p.dns.ID = p3.dns.ID \leftarrow 1$$
$$p.dns.answers = p2.dns.answers \leftarrow 1$$

Given the sorting performed by the algorithm, the order of the sorted atoms is: $p1p2p3p4$. For this particular example, the order of the prototypes is not altered when sorting them. Nonetheless, the algorithm in the general case will sort with respect of the nested most common atoms. Finally, the generated EFSA by our algorithm is represented in the Figure 5.1.

Once having the generated EFSA, we can introduce the proposed algorithm that will be used to evaluate the packets using the auxiliary data structure we generated. The algorithm is shown in Algorithm 5.

The main idea behind the Algorithm 5 is to go through the transitions of the EFSA. If the evaluated predicate (or atom equivalent test) contains dependencies, filter the stored packets that do match the predicate along with the packet; if the packet matches with at least one stored packet, the predicate is considered to be successfully passed the test. If the predicate does not include dependencies, then, the predicate is directly tested. If the predicate successfully passed the test, it triggers a transition. If the transition is executed, all updating functions are performed. The updating functions related to store the packet or report success on the property. If the transition is executed, the current state is replaced by the final state of the transition and the evaluation process starts
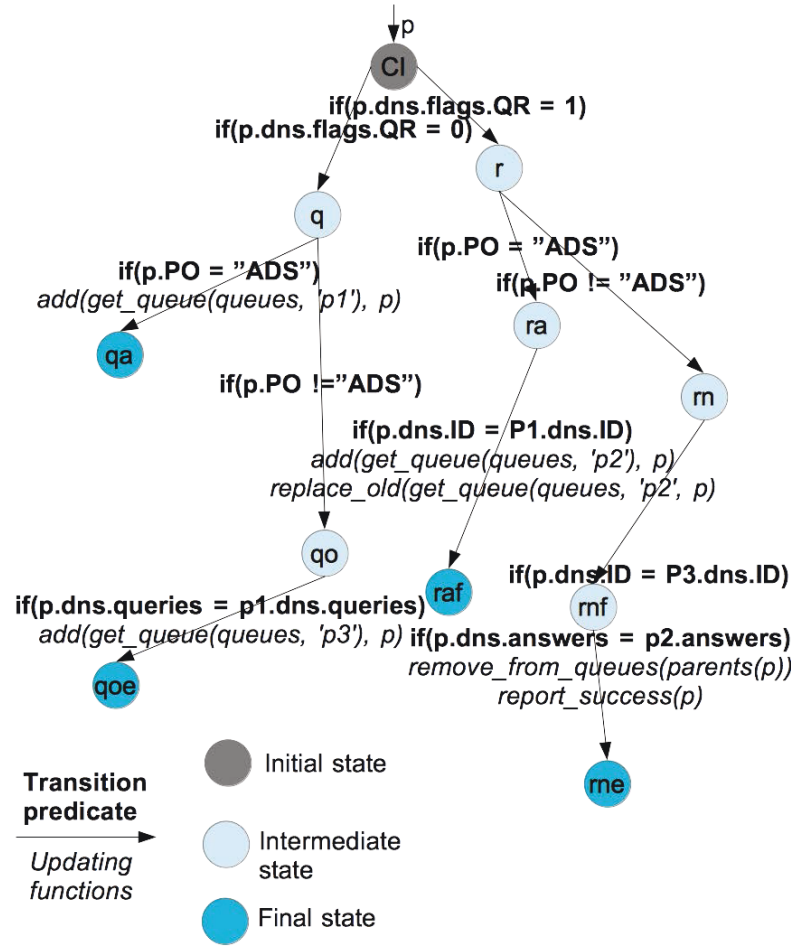
FIGURE 5.1: Generated EFSA

again from the predicates of that state. If the predicate does not pass successfully, then, the next predicate of the current state is evaluated. Now, we can proceed to calculate the complexity of the evaluation algorithm using the auxiliary EFSA (as shown in Algorithm 5):

$$
\begin{aligned}
T(eval\_states) &= \sum_{i=1}^{|S|}(\Theta(1) + T(eval\_trans_i)) \\
&= |S| + \sum_{i=1}^{|S|} T(eval\_trans_i)
\end{aligned}
\tag{5.2}
$$

Where $|S|$ is the cardinality of the set of states in the EFSA.

Algorithm 5: Evaluation algorithm using EFSA

Input: EFSA A, StoredPackets sps, Packet p_i
Output: Property verdicts
repeat

    transitioned ← false;
    foreach *predicate in getTransitions(getCurrentState(A))* **do**
        if *hasDependencies(predicate)* **then**
            partialResult ← false;
            tempStoredPackets ← createStoredPackets();
            foreach *storedPacket in getStoredPackets(sps, predicate)* **do**
                if *evalDependency(predicate, p_i, storedPacket)* **then**
                    partialResult ← true;
                    add(tempStoredPackets, storedPacket) /* avoid unnecessary checks */
                end
            end
            setStoredPackets(sps, predicate, tempStoredPackets);
            transitioned ← partialResult;
        else
            transitioned ← evalIndependent(predicate, p_i);
        end
        if *transitioned* **then**
            foreach *updatingFunc in getUpdatingFuncs(A, predicate)* **do**
                execute(updatingFunc, p_i);
            end
            setCurrentState(A, getStateTransition(A, predicate));
            goto *end_loop*;
        end
    end
    end_loop :
until *transitioned*;

$$T(eval\_trans_i) = \sum_{j=1}^{|TA_i|} (\Theta(1) + \Theta(1) + \Theta(1) + T(eval\_sp_j) + \Theta(1) + \Theta(1)) + \Theta(1)$$

$$+ \sum_{j=1}^{|U_i|} \Theta(1) + \Theta(1)$$

$$= 5\,|TA_i| + |U_i| + \sum_{j=1}^{|TA_i|} T(eval\_sp_j) + 2\Theta(1)$$

$$(5.3)$$

Where $|TA_i|$ is the cardinality of the set of transitions of the *ith* element of the state set, $|U_i|$ is the cardinality of the set of updating functions for the *ith* state executed transition.

$$T(eval\_sp_j) = \sum_{k=1}^{|Q_j|} (\Theta(1) + \Theta(1) + \Theta(1)) \tag{5.4}$$
$$= 3\,|Q_j|$$

Where $|Q_j|$ is the length of the queue of the *jth* prototype stored packets queue.

Substituting the equation 5.4 into the equation 5.3 we get:

$$T(eval\_trans_i) = 5\,|TA_i| + |U_i| + 2\Theta(1) + \sum_{j=1}^{|TA_i|} 3\,|Q_j| \tag{5.5}$$

Subsequently, substituting equation 5.5 in the equation 5.2 we get:

$$T(eval\_states) = |S| + \sum_{i=1}^{|S|} (5\,|TA_i| + |U_i| + 2\Theta(1) + \sum_{j=1}^{|TA_i|} 3\,|Q_j|)$$
$$= |S| + \sum_{i=1}^{|S|} (5\,|TA_i|) + \sum_{i=1}^{|S|} (|U_i|) + \sum_{i=1}^{|S|} (2\Theta(1)) + \sum_{i=1}^{|S|} (\sum_{j=1}^{|TA_i|} 3\,|Q_j|) \tag{5.6}$$
$$= 3\,|S| + \sum_{i=1}^{|S|} (5\,|TA_i|) + \sum_{i=1}^{|S|} (|U_i|) + \sum_{i=1}^{|S|} (\sum_{j=1}^{|TA_i|} 3\,|Q_j|)$$

We note that counting from all states each transition is the equivalent to count all transitions, i.e., $|T|$, the cardinality of all transitions. Similarly, counting from all states each updating function is the equivalent to count all updating functions, i.e., $|U|$, the cardinality of all updating functions. Doing this substitutions in the 5.6 equation, we get:

$$T(eval\_states) = 3\,|S| + 5\,|T| + |U| + 3\sum_{i=1}^{|T|} |Q_i| \tag{5.7}$$

The complexity of our algorithm results in an improved linear complexity, $O(|T|) = \sum_{i=1}^{|T|} |Q_i|$. We also note that any algorithm that runs in linear time can only modify a linear amount of memory cells and therefore the space complexity of the algorithm yields a linear space complexity. It is also important to remark that the complexity of the algorithm (both in time and space) highly depends on the length of the stored queues of packets. In our previous chapter, Chapter 3, we have proposed having a continuous parallel process that given a timeout threshold will remove from the packet queues

unused packets. We do this in order to avoid resource starvation in the monitoring system. This timeout parallel process is kept for this algorithm as well.

5.4 Effective language modifications adapted to the enhanced system's model

In the previous section we presented an approach that is capable of testing properties in a scalable manner. As it can be appreciated, the algorithms do not depend on the input language, but, on the basic definitions proposed by our approach, particularly in the Definition 3.5, the packet prototype definition. Therefore, any input language can be used as long as it expresses correctly the concept of packet prototypes. One might think that as a result no language is necessary to be defined, nonetheless, there exist many advantages in having a carefully designed language that will express all the constraints needed. Perhaps the main reason is that a common language can motivate ease the use of the approach for different users and researchers without in-depth knowledge of the internal concepts and functioning of our proposed approach. Furthermore, with a very well specified language the ambiguity in while describing the desired trust properties can be avoided and this fact while testing is highly important, in order to clearly know the expected results of the properties. Given this motivation, we propose an extension and modification of the language proposed by [Lalanne and Maag, 2013]. All the aspects are further discussed in a holistic manner through different considerations of the language. These considerations are exposed in the following subsections.

5.4.1 Lexical considerations of TeAR

In order to define the structure of the language (grammar) we first need to define which characters are allowed in the language. More importantly, how the sequences of characters form meaningful character strings (we call these sequences: tokens, and a single sequence: token) are formed. In order to precisely define how the language meaningful sequences are formed, the lexical analysis is performed. This will be the equivalent to human language as how to identify correctly an article, a subject, etc.; then, with the syntactic analysis we say that a sentence can be formed using them in the appropriate order.

We assume that all reserved keywords or tokens are case sensitive for those characters with case distinction they are considered differently. For example a variable named x1 will differ from the variable named X1 or a property named π will be different from a property named Π.

We have few reserved keywords in our language. Those are the following: "HN", "exists", "foreach", and "null". The tokens, "->", "<-", "(", ")", "∧", "=", "!=", ">", ">=", "+", ":", and "-" are self-represented tokens and they have a special meaning depending on where they are found in the language (grammatical sense). Further details about the meaning of each token, and reserved word are expressed in subsection 5.4.3 of the language.

In order to provide readability and cleanness we have also considered a mechanism to add comments. We decided to use one of the most standard notations for comments in computer science. The beginning of a comment starts with the character sequence "//" and finishes with the end of the line; a multi-line comment starts with the sequence "/*" and finishes with the sequence "*/". Tokens are separated from each other with the use of a blank space. Continuing with the aspect of readability, a blank space is defined as one or more white spaces, tabulators, breaking-line characters and also comments. This was motivated due to past experiences when we struggled to keep our work easily readable for us and for other reviewers of our work.

NUMBER is an integer number or a floating-point number. An integer number is a sequence of numeric characters (from 0 to 9) or a 0x sequence followed by a sequence of numeric characters or characters from a to f, and its respective upper case equivalent. An integer can also be a minus sign (-) followed by a sequence of numeric characters described before. Note that integers can be described as in its hexadecimal representation with the use of the sequence 0x. A floating-point number is a sequence of numeric, followed by a dot and another sequence of numeric characters; it can also be a minus sign (-) followed by a sequence of numeric characters followed by a dot and another sequence of numeric characters. Please note that for the lexical analysis we do not consider the length of the numbers, the range of those numbers is expressed in subsection 5.4.3.

STRING is used to describe a string constant. String constants are formed by a sequence of CHARS enclosed by double quotes. CHARS are considered to be any UTF-8 character in exception of double quotes (") or backslash (\), plus the two-character sequences \", \\, \t, \r and \n to denote double special characters inside the string constant.

PROP_NAME is used to describe a property name. Typically in our approach, property's names are considered to use Greek characters, just to make a clear distinction of what a property name from any other identifier. Then, PROP_NAME is a sequence of alphanumeric characters ([0-9a-zA-ZA-Ωα-ω]), underscore (_) and not starting with a number. It is not a requirement to name properties with Greek characters.

VAR_NAME is used to describe variable identifiers. VAR_NAME can be a variable name or a variable name with a timeout. A variable name is formed by a sequence

of alphanumeric characters or the underscore character, but not starting with a number. VAR_NAME_W_TO describes a variable name with a timeout, it is a sequence of alphanumeric characters (not starting with a number), or the underscore character followed by a colon (:) and then followed by an integer number (see above). VAR_NAME_W_HC describes a variable name with a hold condition timeout, it is a sequence of alphanumeric characters (not starting with a number) or the underscore character followed by a colon (:) and then followed by the keyword HN. Details regarding the timeouts are given in subsection 5.4.3.

SELECTOR_VARIABLE is used to describe data within a hierarchical structure. The pattern used to describe this is formed by any variable name (see above) followed by any amount of repetitions formed by: a dot (.) followed by a variable name sequence. An example would be "req_sip1.sip.cseq.method". The meaning of this is detailed as well in subsection 5.4.3.

Finally, it is important to consider that all tokens must be separated by a blank space (see above) or a token that is not a keyword or an identifier. For example the sequence HNHNHNHN will not be considered four different keywords, it will be considered a variable name due to the fact that the longest matching sequence corresponds to a variable name. Nevertheless, the sequence "req_sip1.sip.cseq.num-2" will be considered three different tokens, since the "-" token, which is not a keyword nor an identifier separates the SELECTOR_VARIABLE and the NUMBER.

Note: Any other input will be considered unacceptable, and it will make that the input source not to be taken into account.

5.4.2 TeAR grammar

In this subsection, we describe the TeAR grammar. We will use the very popular and well-adopted Backus Naur Form (BNF) [Backus, 1959] notation to represent the grammar. Briefly described, the non-terminal symbols are denoted in lower case characters and enclosed between the pair of symbols "<" and " >". A terminal symbol will be represented in upper case or between quotation marks. The vertical bar (or pipe symbol) "—" represents a choice between one symbol substitution and the other. Finally, the "::=" symbol is used to separate the non-terminal symbol on the left hand side from the right hand terminal and non-terminal symbols in the production rules. For our particular case, our terminal symbols do not include any of the meta-symbols (such as repetitions or optional) used to describe the grammar; hence, we will not use any special notation other than the previously described tokens in the lexical analysis of the language. The first is the term production rule:

$$< term >::= < term > \text{``} + \text{''} < term > \, | < term > \text{``} - \text{''} < term > \, |\text{``(''} < term > \text{``)''}$$
$$|NUMBER|STRING|SELECTOR\_VARIABLE|VAR\_NAME|\text{``null''}$$

The last production rule specifies which data can be considered. Please note that the basic data types and basic arithmetic operators are considered so far. For more information regarding the terms, please see our subsection 5.4.3. A term is an element which can be used while doing atomic comparisons. The following production rule shows the atoms (atomic comparisons) production rule:

$$< atom >::= < term > \text{ `` } = \text{ '' } < term > \, | < term > \text{``!} = \text{ '' } < term >$$
$$| < term > \text{ `` } > \text{ '' } < term > \, | < term > \text{ `` } >= \text{ '' } < term >$$

The production rule presented above, is one of the most important production rules of the language. It describes the atomic comparisons upon which the language is based on. We add a useful range of comparisons; by combining different terms, more complex comparisons can be achieved. The meaning behind each comparison is very clear, nevertheless, each interpretation is covered in subsection 5.4.3. The following production rule is simply an auxiliary production rule, it is nothing more than a comma separated term list used in other production rules.

$$< term\_list >::=< term > \, | < term > \text{``,''} < term\_list >$$

One of the production rules that uses the term list is the macro production rule. The primary use of it is to provide a ease of use and readability when using the language, in order not to repeat common conjunctions of atoms. We give further detail of the macros in the subsection 5.4.3.

$$< macro >::= VAR\_NAME\text{``(''} < term\_list > \text{``)''}$$

Please note that the macro production rule is a way to invoke a previously defined macro. How to define macros is presented later in this subsection. The following production rule denotes the atoms conjunction:

$$< atoms\_conjunction >::= < atom > \mid < macro > \mid < atom > \text{ "} \wedge \text{"}$$
$$< atoms\_conjunction >$$

As it can be seen from the previously stated production rule, the atoms can also be a macro. Therefore, a conjunction of atoms is a simple conjunction between macro calls and atoms. The following production rule is a production rule which encapsulates all the possible manners to express a variable and its associated timeout:

$$< var\_timeout >::= VAR\_NAME \mid VAR\_NAME\_W\_TO \mid VAR\_NAME\_W\_HC$$

As mentioned in the subsection 5.4.1, there are three ways to express a variable and its respective timeout, this production rule collects them all. The next production rule is the formula production rule, it is among the most important production rules of the language, it expresses how a formula could be constructed in order to test it against the network packets.

$$< formula >::= \text{ "foreach"} < var\_timeout > \text{ "("} < atoms\_conjunction > \text{ ")"}$$
$$\mid \text{ "foreach"} < var\_timeout > \text{ " > "} < var\_timeout > \text{ "("} < atoms\_conjunction > \text{ ")"}$$
$$\mid \text{ "foreach"} < var\_timeout > \text{ " < "} < var\_timeout > \text{ "("} < atoms\_conjunction > \text{ ")"}$$
$$\mid \text{ "exists"} < var\_timeout > \text{ "("} < atoms\_conjunction > \text{ ")"}$$
$$\mid \text{ "exists"} < var\_timeout > \text{ " > "} < var\_timeout > \text{ "("} < atoms\_conjunction > \text{ ")"}$$
$$\mid \text{ "exists"} < var\_timeout > \text{ " < "} < var\_timeout > \text{ "("} < atoms\_conjunction > \text{ ")"}$$
$$\mid < formula > \text{ " } \rightarrow \text{ "} < formula > \mid \text{"("} < formula > \text{")"} \mid PROP\_NAME$$

From the previous production rule, one important aspect is the possibility to include previously declared formula via macro formula assignation (later explained) by the use of PROP_NAME. The usage and meaning of the production rules are stated in subsection 5.4.3. Besides that, the only non-recursive production rules of formulas, i.e., the conjunction of atoms is previously stated. The goal of the language is exactly to test a set of formulae against the network packets as stated before, however, further auxiliary rules of the language are needed in order to help its definition. Following we state the formula assignation:

$$< formula\_assignation >::= PROP\_NAME \text{ “} = \text{”} < formula >$$

The previous production rule is to emphasize that each property must have its own name. This is done as a mechanism to refer to a certain formula in the formulae. The following production rule is the list of variable names, an auxiliary production rule of a simple comma separated variable list:

$$< var\_name\_list >::= VAR\_NAME | VAR\_NAME \text{ “,” } < var\_name\_list >$$

The usage of the previously stated production rule is found in the macro assignation production rule, which is stated in the following:

$$< macro\_assignation >::= VAR\_NAME \text{“(”} < var\_name\_list > \text{ “)” “<-”}$$
$$< atoms\_conjunction >$$

This provides flexibility and readability when writing the formulae. A basic replacement concept of the right hand assignation from the left hand. A clear definition of how to use the macro assignation is stated in subsection 5.4.3. Next we present another macro assignation, but, a wider level, the formula macro assignation:

$$< formula\_macro\_assignation >::= PROP\_NAME \text{ “<-” } < formula >$$

The previous production rule is just to avoid repeating common parts of a formula. Further information regarding its interpretation and usage can be found in subsection 5.4.3. Finally, we present the formulae production rule, it joins all the previously stated production rules:

$$< formulae >::= < formula\_assignation > \; | < formula\_assignation >< formulae >$$
$$| < formula\_macro\_assignation >< formulae >$$
$$| < macro\_assignation >< formulae >$$

The previous production rule, this denotes that the grammar can accept several formula assignations, formula macro assignations or macro assignations. Please notice that at least one formula assignation is necessary.

Note: The input that describes the properties must comply with the production rules defined above. If the input source does not follow the grammar rules, then, it will be considered a faulty input and not taken into account.

Also, variables express associated timeouts. We added a new production rule, a very simple production rule for the variables and timeouts described below:

Note: The input that describes the properties, must comply with the production rules defined above. If the input source does not follow the grammar rules, then, it will be considered a faulty input and not taken into account.

5.4.3 Semantics of TeAR

Hereafter, we focus on the semantic of the new language, i.e., the meaning behind the language composition. We will provide how each component and construction of the language should be interpreted; this is highly important, since it will leave no room for ambiguities. First, it is important to understand that the language is designed to test network packets and make correlations between them in real time (on-line).

As stated before (as it can be seen in our language's grammar), our language consists of a list of property assignations, macro assignations and formula macro assignations. The list should be treated as: individual and unrelated set of formulae; the macro assignations and formula macro assignations will not be tested, they are only used to avoid text duplication and to provide more readability in the formulae, therefore, at least a formula assignation is required, this is also enforced by the language's grammar. The formula assignations provide properties to check in the trace or live capture; those properties must be checked upon each new packet arrival. A sequence of packets that do not comply with one or many properties is considered to be a failure for each property that it does not comply. Therefore, it is possible to have several failures of each of the properties that a packet does not comply.

Unused macros or formula macros are acceptable, however, not recommended. The use of macros should also be taken into consideration just to provide readily and to avoid repeating common parts of the formula. Nevertheless, one should not rely on this mechanism as both of them are just pre-processing features. For both, the substitution will be done replacing the text at the left hand of the "<-" reserved word with the text at the left side. As an example, consider the following formulae:

Π <- $foreach\ x : 10(x.IP.source =$ "192.168.5.100″)

$withinTime(x, y, T)$ <- $x.time + T >= y.time$

$\Gamma 1 = \Pi$ -> $exists\ y : 10 > x(y.IP.source! =$ "192.168.5.100‴ $\wedge withinTime(x, y, 9))$

Direct substitutions will be applied taking the macro assignations into the formulae. This will have the same result as writing the single property like this:

$\Gamma 1 = foreach\ x : 10(x.IP.source =$ "192.168.5.100″) ->

$exists\ y : 10 > x(y.IP.source! =$ "192.168.5.100‴ $\wedge x.time + 9 >= y.time)$

As stated before, the only advantage of macros is readability and avoiding text duplication, and both formulae achieve the same set of tests.

Given the fact that two formulas can express the same with different order of elements, we define *equivalence between formulas*. For the sake of simplicity and readability, we explain the equivalence between them when we explain each of the language components. Nevertheless, we will make use of this concept to further explain some elements of the language with higher hierarchy.

Moving on to the formula assignations, they are nothing more than a mechanism for referring to each formula. Also note that we consider possible the possibility of two or more different formula assignations with same or *equivalent formula*. This is discouraged and considered to be a fault. We do not remove one of the formulae, since, the tool is unable to determine which one the user wants to keep. The reason for this to be a fault is that, on-line monitoring systems cannot waste any resources. In order to express possibilities after a common part of the formula, we do not provide a mechanism to express options. The reason is that given the inherited meaning of formulae being independent clauses, the logical or value is implicit between them. Therefore, writing two formulae with different parts expresses the optionality.

Logical or construction for TeAR proof. Let α, β, and γ be a formule. Let Φ the desired construction of choices between β and γ after α, i.e., $\Phi = \alpha \rightarrow (\beta \vee \gamma)$.
$\Phi = \alpha \rightarrow (\beta \vee \gamma)$, $\Phi = \neg\alpha \vee \neg\alpha \vee (\beta \vee \gamma)$, $\Phi = \neg\alpha \vee \beta \vee \neg\alpha \vee \gamma$, $\Phi = \alpha \rightarrow \beta \vee \alpha \rightarrow \gamma$ $\square$

For example, to express that after a request from an IP address a response from one of two different IP address can come it can be done with two different formulae. First let us assume the source iP as:,"10.0.10.10", and the possible responses as, "10.1.10.1" or

"10.1.10.2". This will be expressed as:

$$\Phi1 = foreach\ x : 10(x.IP.source = \text{``}10.0.10.10\text{''}) \rightarrow$$
$$exists\ y : 10 > x(y.IP.source = \text{``}10.1.10.1\text{''})$$
$$\Phi2 = foreach\ x : 10(x.IP.source = \text{``}10.0.10.10\text{''}) \rightarrow$$
$$exists\ y : 10 > x(y.IP.source = \text{``}10.1.10.2\text{''})$$

It is up to the entity processing the verdicts to determine if the failure of one of this properties is considered a fail or the composition of them (if $\Phi1$ or $\Phi2$, then, success or fail).

A formula expresses a sequence of packets as they will appear, the tests required for them to pass, and the relationships between those packets. Given the construction of the formula grammar, the formula can only express the tests applied to a packet. If a packet matches the tests (the atomic tests conjunction, discussed after), then, this packet will be considered to belong to the set of the stated variable, and added to its corresponding queue. Therefore, we say that a variable in the language declares a packet *prototype*, containing all the specifications of what a packet should have to be considered to belong to the set of that prototype.

As well, as stated in the grammar, the only way to *link* two or more packets is through the reserved keyword "->", which should be interpreted as *implies*; an if-then semantic construction. The time relationship between the packets is expressed via the ">" and "<" reserved keywords, they should be interpreted as after and before, respectively. Two important concepts need to be explicitly stated. First, the **cause** and **consequence** packets, the cause is the packet described by the formula at the left part of the "->" keyword, and the consequence, the one at the right. If the cause formula is met, then, the consequence of the formula must be met; otherwise, a **FAIL** verdict will be obtained (so far, fail verdicts can be obtained via the timeouts, more of this explained further). The second concept is about the time positioning of the packets. The packet which is called **predecessor** is the one captured by the monitoring tool, before the **successor**, and viceversa.

Please note that a packet which is stated to be the predecessor to any other successor of packets cannot include any references (in the terms) to the packets depending on it. The reason is that information of the future packet is not available in when testing the packet stated to be a predecessor in the past. For example, the following formulae

contains errors:

$$\Phi1 = foreach\ x : 10(x.IP.source = \text{``}10.0.10.10''\text{''}) \rightarrow$$
$$exists\ y : 10 < x(y.IP.source = \text{``}10.1.10.1''\text{''} \wedge y.tcp.sport = x.tcp.dport)$$

The reason is that when testing a packet, to determine if that packet belongs to the set "y", no information regarding "x" is available. Therefore, this formula is invalid. For instance, to test if the corresponding communication ports match, the atomic tests could go in the successor packet, "x", like this:

$$\Phi1 = foreach\ x : 10(x.IP.source = \text{``}10.0.10.10''\text{''} \wedge y.tcp.sport = x.tcp.dport) \rightarrow$$
$$exists\ y : 10 < x(y.IP.source = \text{``}10.1.10.1''\text{''})$$

A packet will be kept in the system during a certain period of time, depending on the specified configuration. Nevertheless, there are two main cases, if the associated timeout (expressed via <var_timeout>) has not been reached, or if the packet is a dependency of a successor packet. In the second case, if the timeout is reached for the second packet, the dependency will be removed and if the packet has no other dependencies it will be released. If the timeout is reached for a packet, the error will be reported for this packet and all its associated predecessors only once. The default time units for the **timeouts are expressed in microseconds**. The timeout **must** be detailed in the language in a formula when the new variable (prototype) is declared. A timeout of a previously declared variable will be ignored. For example:

$$\Phi1 = foreach\ x : 10(x.IP.source = \text{``}10.0.10.10''\text{''} \wedge y.tcp.sport = x.tcp.dport) \rightarrow$$
$$exists\ y < x : 20(y.IP.source = \text{``}10.1.10.1''\text{''})$$

The timeout 20us in the variable x is ignored in the previous formula, and the prototype y is not considered to be complete, since, no timeout was declared, thus yielding an error.

The quantifiers, "foreach", and "exists" have a meaning in regards of the cause (left part of the "implication"). "exists" implies that the cause is taken away from the potential packets in its set of the cause packet(s), since, at least one occurrence matching "exists" has been found. On the contrary, "foreach", will keep the cause packet(s) in its(their) queue. At the *chronological* beginning (the first predecessor) of a formula, only "foreach" quantifiers should be used, since on-line monitoring is a continuous process, and looking for a single packet is not its target. It is considered an error to have an "exists" as

the first chronological packet. Also, please note that the performance might be affected when using the "foreach" quantifier in the consequence packets (after the "implication"), and *most* of the times each packet is not associated to many. Also, the quantifiers differ in respect of the timeout of the cause of the packet. For an "exists" quantifier in the consequence, the packets at the cause get released from the memory if they have no dependencies. For a "foreach" quantifier in the consequence, the last packet at the cause remains in the queue for the configured timeout amount of time, if the timeout is then reached, no error is produced for that packet. Consider the following TeAR formulae:

$$\Phi 1 = exists\ y : 10 > x(y.IP.source = \text{``10.0.10.10''} \land y.tcp.sport = x.tcp.dport)\ \text{->}$$
$$foreach\ x : 10 < y(y.IP.source = \text{``10.1.10.1''})$$

Please note that in the previous formulae, the first chronological packet prototype is x. Therefore, no violation regarding the quantifier chronological position occurs. An interesting aspect to note is how the semantic behind the construction of the reply packet in the cause differs if the reply packet was in the consequence. If the reply packet is described in the consequence, it implies that all requests *must* have those replies; if the reply packet is in the cause, it implies that a reply formed like that *must* have a request as described.

A special way to keep packets is defined with the VAR_NAME_W_HC type. It keeps the packet until a new packet matching the atoms arrives, then, the packet is released when no more dependencies are present.

Atoms conjunctions are nothing more than a logical conjunction of atomic tests — a list of tests. Atoms on the other hand, as stated before, are atomic tests, comparisons between one data element (term) and another. Note that relationships such as < and <= can be expressed interchanging the order of the terms, this is the reason why they are not included. A term can have four different data types: "null", number, string, and associative array. The associative array type is a collection of key value pairs; the values can be either 'null", number, string, and associative array, the keys are string labels. The comparisons between "null" and any data type is allowed. String-number comparisons are not allowed. The comparison between associative arrays can be done only with associative arrays (and null). The comparison between associative arrays is done recursively, so that all the set of keys in one should be the same in the other, and the values associated of both should be the same.

Terms belong to the stated data types, they can either be constant values or variable terms extracted from the network packets through the SELECTOR_VARIABLE. The SELECTOR_VARIABLE can give also dictionary types as explained before.

Numbers can only be represented in the range $[-2147483647, +2147483647]$, larger integers or floating point numbers are not allowed. This is very important for the language users as well as for the implementation; the messages hierarchical data structures **must** take this into consideration, if a number has a bigger representation in the protocol, this number should be split into parts to process it correctly.

Finally, we can explain the concept of formula equivalence starting by the definition of term equivalence. A term is said to be equivalent to another if they are both constants and if they are equivalent, or if they are the same selector variable. String constants are equivalent if they are equal strings, case sensitive. Number constants are equivalent if the number value is exactly the same; for integer-floating-point number comparison, no approximation is performed. An atom is said to be equivalent to another if the operator of both atoms is the same and they contain the two matching terms (in no particular order). An exception occurs with the ">", and ">=" operators. the mathematical equivalence should be preserved, and therefore the terms are not exchangeable. For example, consider the following formulae:

$$\beta 1 = exists\ y : 10 > x(y.IP.source = \text{``}10.0.10.10'' \wedge y.tcp.sport = x.tcp.dport) \text{ ->}$$
$$foreach\ x : 10 < y(y.IP.source = \text{``}10.1.10.1'')$$
$$\beta 2 = exists\ y : 10 > x(\text{``}10.0.10.10'' = y.IP.source \wedge y.tcp.sport = x.tcp.dport) \text{ ->}$$
$$foreach\ x : 10 < y(y.IP.source = \text{``}10.1.10.1'')$$

In the previous example, both, $\beta 1$ and $\beta 2$ are equivalent formulae. As stated before, equivalent formulae is considered an error in the formulae input.

One last important fact that needs to be taken into account is the uniqueness of the identifiers. The identifiers must be unique for the formulae names as well as for the variable prototypes inside a formula. This must be true after the macros have been replaced. Circular macro referencing will produce an error, i.e., one macro cannot reference another macro which references the first one.

5.4.4 TeAR examples

In this subsection, for a better understanding of the language, an example is provided. We express properties extracted from the SIP [Rosenberg et al., 2002] protocol conformance and performance requirements in the proposed language. Also, we express in comments technical details of those formulae, such as, timeouts and others.

Let us assume the following conformance requirements: "Every INVITE request must be responded with a final non-provisional code", "Every successful INVITE request must be responded with a success response". Also consider the following performance requirement: "The response time for each request should not exceed 8s". Based on this requirements, we create the following formulae:

```
/************************************************************************

MACROS

*************************************************************************/


/************************************************************************

*** A request has the sip method.

*** For a response the SIP response code is found instead.

*************************************************************************/
```
$$request(x) <\text{-} x.sip.method \mathrel{!=} null$$
$$response(y) <\text{-} y.sip.method = null$$

```
/************************************************************************
```
To see if a SIP response corresponds to a request, one should check the equality

of the cseq of the headers.
```
*************************************************************************/
```
$$responds(y, x) <\text{-} request(x) \land response(y) \land x.sip.headers.cseq = y.sip.headers.cseq$$

```
/************************************************************************
```
A provisional request code in SIP is defined as 1xx. Non provisional are the

rest of codes. Also, codes start at 100.

```
*************************************************************************/

nonProvisional(y) <- y.status.code >= 200


/*************************************************************************

A successful request code in SIP is defined as 2xx.

*************************************************************************/

successful(y) <- y.status.code >= 200 ∧ 300 > y.status.code


/*************************************************************************

A way to calculate if a packet was observed before a desired time T,

relative to another packet.

*************************************************************************/

inTime(p1, p2, T) <- p2.time + T >= p1.time


/*************************************************************************

FORMULA MACROS

*************************************************************************/


/*************************************************************************

Formula macro for invite requests with a timeout of 10s

*************************************************************************/

μInvite <- foreach x : 10000000(request(x) ∧ x.sip.method = "INVITE")


/*************************************************************************

FORMULAE

*************************************************************************/


/*************************************************************************

First conformance requirement

Every INVITE request must be responded with a final non-provisional code

*************************************************************************/

Θ1 = μInvite -> exists y : 10000000 > x(nonProvisional(y) ∧ responds(y, x))
```

```
/*************************************************************************
```

Second conformance requirement

Every successful INVITE request must be responded with a success response

```
*************************************************************************/
```

$\Theta 2 = \mu Invite \rightarrow exists\ y : 10000000 > x(successful(y) \wedge responds(y, x))$

```
/*************************************************************************
```

First performance requirement

The response time for each request should not exceed 8s

ACK responses are just acknowledgments, therefore, we do not care

```
*************************************************************************/
```

$\Phi 1 = foreach\ x : 10000000(request(x) \wedge x.sip.method\ != \ ``ACK'') \rightarrow$

$exists\ y : 10000000 > x(successful(y) \wedge responds(y, x) \wedge inTime(x, y, 8000000))$

```
/*************************************************************************
```

Example: previous formula without macro usage:

```
*************************************************************************/
```

$\Phi 1 = foreach\ x : 10000000(x.sip.method\ != null\ \wedge\ != \ ``ACK'') \rightarrow$

$exists\ y : 10000000 > x(y.status.code >= 200 \wedge 300 > y.status.code) \wedge y.sip.method = null \wedge$

$x.sip.headers.cseq = y.sip.headers.cseq \wedge y.time + 8000000 >= x.time)$

To demonstrate the flexibility of our approach, we include a property that will test the TCP protocol [Postel, 1981]. The conformance requirement to monitor is "For each syn-ack there should exist an ack corresponding to it". In addition to it, the performance requirement: "Every TCP packet to the host 217.160.43.140, an acknowledgment of the sent data should be received and it should not exceed 1s". The properties expressed in the TeAR language are as follows:

```
/*************************************************************************
```

—Conformance Requirement—

For this example, we do not use macros.

Please note:

A. The source and destination ports are matched.

B. The the ack number in the ack package corresponds to the

 previous seq number plus one (reply to syn-ack).

***/

$syncorrect = foreach\ x : 10000000(x.tcp.flags.syn = 1 \wedge x.tcp.flags.ack = 1)\ ->$

$exists\ y : 1000000 > x(y.tcp.flags.ack = 1 \wedge y.tcp.ack = x.tcp.seq + 1 \wedge$

$y.tcp.sport = y.tcp.sport \wedge y.tcp.dport \wedge y.tcp.dport = x.tcp.sport)$

/***

—Performance Requirement—

For this example, we do not use macros.

Please note:

A. The source and destination ports are matched.

B. The the ack number in the ack package corresponds to the

 previous seq number (reply not to syn-ack).

***/

$\Phi1 = foreach\ x : 5000000(x.tcp.flags\ != 0x10 \wedge x.ip.ip\_dest = "217.160.43.140'')\ ->$

$exists\ y : 5000000 > x(y..ip.ip\_source = "217.160.43.140'' \wedge y.tcp.sport = x.tcp.dport \wedge$

$y.tcp.dport = x.tcp.sport \wedge y.tcp.flags.ack = 1 \wedge y.tcp.ack = x.tcp.seq$

$\wedge y.time + 1000000 >= x.time)$

Chapter 6

A generic trust framework

"To be trusted is a greater compliment than being loved" — *George MacDonald, The Marquis of Lossie (1877).*

While performing the first work on this thesis, proposing another trust management engine was not what we intended. We assumed that, by modifying one of the existing trust management engines to incorporate our approach presented in Chapter 3, we could make trust management engines generic in terms of the possible trust features they could evaluate and take into consideration.

As stated in Chapter 1, the work performed in this Chapter is based on identifying the open problems in the trust domain. Furthermore, solutions to these problems were proposed. In addition to that, as shown in the Chapter 2, the existing solutions do not tackle these precise issues.

After our proposed approach to generically provide feedback regarding the entities' interactions, we noticed that no trust management engines or frameworks are capable of being easily re-configurable in the parameters they use, when adding new, and extensible trust properties to monitor a set of behaviors. Furthermore, the devices which can use the trust concepts might have various capabilities, and therefore, a framework that might allow offloading the services is proposed. In this Chapter, we propose a generic framework with a RESTful web-service based architecture to exchange the trust feedback and to get the trust information from the trust management engine. We also discuss the architecture, design, and proposed input features for the associated trust model, namely in Section 6.2.

Further, when having all the inputs (*trust features*), an appropriate module (*trust model*) which is capable of processing them and to accurately assess the entity's trust level is

a delicate task. An important aspect of the trust model is that it should be flexible enough and to be able to respond to the trust evaluations as close as possible to the human cognitive notion of trust. We propose a new trust model based on the solution of a multi-class classification problem. We solve the problem using a well known machine learning technique, Support Vector Machines (SVM), optionally using a Gaussian (or Radial Basis) kernel function. The trust model is shown in Section 6.3. Nevertheless, in order to fully comprehend the trust model, important concepts of machine learning and SVM are needed to be introduced. Therefore, we present a preliminary section with those concepts in Section 6.1.

In this chapter, we describe the results which are partially published in [López and Maag, 2015b] and [López and Maag, 2015a].

6.1 Preliminaries — Support Vector Machines (SVM)

To understand what Support Vector Machines (SVM) are, first, we will give a small overview of what machine learning algorithms can achieve, and the basic notions of them.

Machine learning algorithms are designed to learn from available data in order to make predictions / estimations. Typically, the machine learning algorithms operate by building a *model* from the data input to learn how to predict, without being explicitly instructed. The main distinctions in machine learning algorithms are:

- **Supervised machine learning**: the algorithm takes as inputs the *examples* alongside their *expected outputs*. Given the inputs, the final goal is to learn how to map to the training example outputs.

- **Unsupervised machine learning**: The algorithm takes as inputs examples without expected outputs. The goal is to find a structure in the pattern, discovering some common features in the data.

Many other tasks can be accomplished with machine learning, for instance, reinforcement learning is considered to differ from standard supervised machine learning, since, the goal is to take the proper actions given some interactions and to maximize some measure of cumulative reward. For example, a model that learns to play a video-game given "random" attempts and trying to maximize how much it progresses in the level. In this thesis, the discussion of such applications in machine learning is out of the scope. Nevertheless, we focus on supervised machine learning. The reason is that, our main task to solve is to correctly classify the trustworthiness of an entity based on its inputs.

Formally, the inputs are called **features**. A **feature vector**, denoted as $\overrightarrow{X}$, is an n-tuple of the different inputs, $x_1, x_2, ..., x_n$ for an n dimensional feature vector (i.e., $\left|\overrightarrow{X}\right| = n$). The expected output for a given feature vector is called a **label**, denoted simply as y, and the possible set of outputs respectively, Y. The set of examples, called a **training set**, consists of pairs of a feature vector and a label; each pair called a **training example**, denoted as $(\overrightarrow{X}^{(i)}, y^{(i)})$, for the i-th training example. The complete training set can be denoted as $\mathcal{TS} = (\overrightarrow{X}, Y)$. The cardinality of the training set is usually denoted by the letter $m = |\mathcal{TS}|$, a training set with m training examples.

In general terms, in supervised machine learning, the objective is to find a function, $h_\Theta(X)$ called the hypothesis, such that, $h_\Theta : X \mapsto Y$. Please note that, the hypothesis has a subindex Θ, this is because Θ is the array of parameters of the hypothesis function. A common hypothesis is based on the linear combination between the parameters and the features. Therefore, the objective is to find the values of Θ that minimizes the error (commonly the squared error) between the predictions and the output of h_Θ. The function to minimize is often called the **cost function**, denoted as $J(\Theta)$. To optimize the cost functions, known algorithms as *gradient descent*, *Newton's method*, or more specialized techniques as shown in [Fan et al., 2005].

A common problem with supervised machine learning is that the training set could not have enough training examples or that the training set has many examples that will train the machine with poor generalization (also called high variance); when this problem occurs, it is called **over-fitting**. To avoid over-fitting, one popular strategy (perhaps the most popular) relies on using a regularization term in the optimization objective, to obtain "simpler functions". Commonly, the regularization term introduced is the squared Euclidean norm of the weights, i.e., $\sum_{i=1}^{n} \Theta_i^2$. A constant, multiplying either the regularization term or the cost term is introduced to control finely the penalization. This constant is known as the **cost or regularization parameter**, denoted by λ or C.

In order to see how well the prediction of the model generalizes to any given data set, a common technique is employed, i.e., **cross-validation**. Given a data set, the whole data set could be considered the training set. Nevertheless, in order to see how well the trained model generalizes (avoiding over-fitting), a sub-set of the data set is chosen as the training set and another sub-set is chosen to test how accurate the model can predict; the other sub-set is known as the **validation set**. Cross-validation is essentially to perform the validation with different parts of the data set (and therefore the training set with the remainder). The different results of the cross-validation are averaged to output a final estimation of the precision obtained. Furthermore, among cross-validation, one common type which is used is n-fold cross validation. In n-fold cross validation, the set is divided into n equal parts. Then, one part is chosen as the validation set, and the

reminder $n - 1$ parts are used as the training set. The process is repeated n times for the n folds. It is important to note that, depending on the regularization parameter (and others from the particular machine learning algorithm), the same cross-validation accuracy can change.

Typically, when talking about the classification problems that supervised machine learning can solve, multi-class classification refers to when the classification of the feature vector can be mapped into more than two different classes. Depending on the algorithm, they can have native support for multi-class classification or not. In general, if an algorithm can only distinguish between "a positive" and "a negative" class, there are some well known strategies for reducing a multi-class classification problem into a binary classification problem. The most popular strategies to reduce a multi-class classification problem are the ones known as "**one-versus-all**" and "**one-versus-one**". In one-versus-all, the strategy consists in training a single classifier per class, with that class as the positive class and the rest of the classes marked as the negative class. In addition to that, the classifier needs to output a value with the confidence score of the prediction. By choosing the higher confidence score, the final prediction in regards to which class the inputs belong is obtained. In one-versus-one strategy, the model trains $k(k - 1)/2$ binary classifiers, where k is the total number of classes. For each binary classifier, it receives a pair of classes from the original training set and the classifier learns to distinguish between the two classes. The final prediction is obtained where the positive class is obtained the most among all binary classifiers. No strategy has been proven to be better for all cases, for some particular practical cases and algorithms some interesting studies support one strategy over the other, as seen in [Hsu and Lin, 2002], for example.

When talking about the hypersurface that separates the classes of the feature space, called **decision boundary**, there are two major cases. The first is when the data is linearly separable (therefore the hypersurface is a hyperplane), and the second case, is when the data is non-linearly separable. The supervised machine learning algorithms are capable of handling linearly separable data in a fairly easy manner. Depending on the algorithm, there are some ways to classify non-linearly separable data. For example, with *logistic regression*, a higher degree polynomial of the input features could be constructed to obtain a non-linear decision boundary; while using *artificial neural networks*, adding hidden layers will help obtaining a non-linear decision boundary; and finally when using SVM, using the kernel method will achieve the same. Later in this section we discuss what the kernel method is for SVM.

With the general knowledge about supervised machine learning, we can now present the SVM model particularities. Given a training set $\mathcal{TS}$, with training examples $(\vec{X}^{(i)}, y^{(i)})$,

with cardinality m, where each training example $\overrightarrow{X}^{(i)} \in \Re^n$ and $y^{(i)} \in \{-1, 1\}$, the Support Vector Machines proposed in [Boser et al., 1992], and kept until nowadays [Wang et al., 2014], proposes to optimize the following cost function:

$$\underset{\Theta}{\text{minimize}} \quad J(\Theta) = C \sum_{i=1}^{m} \xi_i + \frac{1}{2} \sum_{i=1}^{n} \Theta_i^2$$

$$\text{subject to} \quad y_i(\Theta^\intercal \cdot \phi(\overrightarrow{X}^{(i)}) - b) \geq 1 - \xi_i,$$

$$\xi_i \geq 0.$$

Note that, ξ_i is considered the prediction error for the i-*th* label, calculated as $\xi_i = 1 - y^{(i)}(\Theta^\intercal \cdot \phi(\overrightarrow{X}^{(i)}) + b)$. Also note that the constant C, must be greater than 0. C is exactly the regularization parameter as previously discussed; also note the typical squared Euclidean norm of the weights to avoid over-fitting. Another important characteristic to discuss is the function ϕ. To properly discuss ϕ, one must know the concept of kernels. Let us first introduce that concept.

An important statement is that the algorithm of SVM finds a hyperplane to separate data, making SVM a linear classifier algorithm. Later, to make SVM a non-linear classifier, the kernel method was proposed. Shortly, a kernel function (or similarity function) is a function that allows to find a linearly separating hyperplane, potentially mapping the features into a higher dimensional space. The kernel is a function, such that $k : X \times X \mapsto \Re$ that satisfies:

$$k(\overrightarrow{X}^{(i)}, \overrightarrow{X}^{(j)}) = \phi(\overrightarrow{X}^{(i)}) \cdot \phi(\overrightarrow{X}^{(j)})$$

Where ϕ is a feature map, such that $\phi : X \mapsto \mathcal{H}$, where $\mathcal{H}$ is a dot product space. Examples of widely used and applied kernels are:

- Linear Kernel: $k(\overrightarrow{X}^{(i)}, \overrightarrow{X}^{(j)}) = \overrightarrow{X}^{(i)} \cdot \overrightarrow{X}^{(j)}$

- Polynomial Kernel: $k(\overrightarrow{X}^{(i)}, \overrightarrow{X}^{(j)}) = (\gamma \overrightarrow{X}^{(i)} \cdot \overrightarrow{X}^{(j)} + r)^d$, where d is the degree of the polynomial, γ and r are kernel parameters. $\gamma > 0$

- Radial Basis Function Kernel(RBFK): $k(\overrightarrow{X}^{(i)}, \overrightarrow{X}^{(j)}) = e^{-\gamma \|\overrightarrow{X}^{(i)} - \overrightarrow{X}^{(j)}\|^2}$, γ is a kernel parameter. $\gamma > 0$

- Sigmoid Kernel: $k(\overrightarrow{X}^{(i)}, \overrightarrow{X}^{(j)}) = \tanh(\gamma \overrightarrow{X}^{(i)} \cdot \overrightarrow{X}^{(j)} + r)$, γ and r are kernel parameters.

Please note that the kernel parameters, such as d, γ and r, modify how the kernel could potentially map the data, and therefore how the classification is done. Similarly, as with

the regularization parameter, the best way to find the optimal kernel parameters is done using cross-validation. Certainly, when trying to discover the optimal value for more than one parameter, *all combinations between the reasonable values* for the parameters should be tested in order to determine the best suited parameters. For instance, for two parameters under two different ranges, each point of the Cartesian map of parameters (under the ranges) should be used to train the model using cross-validation sets. The values for the parameters with the best obtained accuracy are the better suited to finally train the model.

Although there exist newer and different kernels proposed by researchers [Wang et al., 2006; Zhang et al., 2013], the RBFK is perhaps the most widely used kernel with SVM; it has proven to have good performance and accurate classification [Deng et al., 2012].

With the concepts of how support vector machines work, we can further explain the idea of our trust model. Nevertheless, first we introduce the whole trust management framework architecture in the following section.

6.2 An Extensible and Dynamic Trust Management Framework

One of the biggest challenges when designing a generic trust management framework is that, trust is a concept for which each entity, user, or system have different trust features, measures, and different contexts. For instance, an entity might consider trustworthy that the responses from a peer arrive within a specified time frame, while another entity might consider trustworthy a low amount of retransmissions. Nonetheless, the same trust management engine should handle the different assessments of trust for the different trustors. This motivates the need to have a generic framework, in which the trust management engine is able to collect the trust information from the trustor, process those features, then, make the information of the trust levels available to the trustor.

The first thing we need to express is that each trustor might have different contexts to evaluate the trustees. Therefore, the trust features will vary from trustor to trustor and from context to context. Furthermore, the trust features of a context might change along the time, in order to use more meaningful trust features for the context, one might add or remove trust features from a given context.

We also note that, in order to give greater flexibility, each trustor might decide differently on how to interact with trustees. Therefore, the framework proposes to provide different trust levels to help the trustor to accurately decide how to interact with the trustee.

When the trust information regarding a trustee in a specified context is available, the trustor decides how to interact with the corresponding trustee. It is up to the trustor's criteria how to interact with the trustee depending on the trustee's *trust level*. For example, a trustor decides not to interact with a trustee with a certain trust level, which can be equal for another trustor that interacts with the trustee with a limited set of operations.

Finally, the last notion is that the information might be collected not only by the trustor; the trustor might allow other entities to provide trust information where the trustor cannot obtain the information. Possibly a trustee performs observable actions for the third party entity or the third party entity has knowledge of the trustee's trust features.

Now, let us define the previously expressed relationships and notions. First, let us define what a *trust feature* is.

Definition 6.1. Let x be one **trust feature**. x is a desired feature to be taken into consideration for a trust assessment. A trust feature has an associated numeric measure. No numeric limitation is defined for trust features, therefore, $x \in \Re$, where as usual, $\Re$ denotes the set of real numbers.

Having the trust feature definition, we proceed to define a *context*.

Definition 6.2. Let x_i be a trust feature, Let $\mathcal{C}$ be a **context**. $\mathcal{C}$ is a defined set of trust features, $\mathcal{C} = \bigcup_{i=1}^{n} x_i$. Where n is the number of trust features associated to the context $\mathcal{C}$. A single constraint is defined for the set size, that is $|\mathcal{C}| > 0$, to guarantee contexts are not empty sets.

With the help of the previous definition we proceed to define what is a *trustee*.

Definition 6.3. Let $\mathcal{CS}$ be a set of contexts. A **trustee** is an entity which interacts within an environment where the trust features of all elements of $\mathcal{CS}$ can be measured. Also, the limitation on the cardinality of $\mathcal{CS}$ is the same, $|\mathcal{CS}| > 0$, this guarantees that a trustee is an entity with meaningful interactions for our framework.

Finally, we define *trustor* in the following manner:

Definition 6.4. Let $\mathcal{TS}$ be a set of trustees for a defined set of contexts $\mathcal{CS}$, each context denoted as $\mathcal{CS}_i$, respectively, we denote the associated trust features as x_{ij}, for $i = 1, 2, ..., |\mathcal{CS}|$, and $j = 1, 2, ..., |\mathcal{CS}_i|$, depending on each context length. A **trustor** is an entity which: i) *might* collect trust features measures, by evaluating x_{ij} from the observations of $\mathcal{TS}$, and provides the numeric measure of those evaluations to the trust management engine; ii) *might* delegate other entities to provide the numeric measure to

the trust management engine about collected trust features, x_{ij} from the observations of $\mathcal{TS}$; iii) queries the trust management engine for the *trust level* of $\mathcal{TS}$ and based on that information, determines how to interact $\forall t \in \mathcal{TS}$.

Immediately, the first question that comes after the stated definitions is what a trust feature can be, and how it can be measured. One good example of what a trust feature is found in our generic approach to obtain trust verdicts as shown in the Section 3.2. One might consider that a simple way to measure is to map -1, 0, and 1 for fail, inconclusive, and pass verdicts respectively might be enough. Nevertheless, since our goal is to have a generic framework, we need to define how the possible trust features behave, and how to calculate their numeric measure for all cases.

Commonly, trust researchers define well known and established features as: *knowledge*, *reputation*, and *experience*. Starting from these well known features we will examine different measure possibilities. As for knowledge, it can be a wide range of elements, for example, a trustee has a certificate or some means to increase its trust, or it has a geolocation corresponding to the given information, etc. For all of these cases, the numeric measure of the trust features will be a number, either a boolean value, representing if the trustee has the corresponding trust feature or not, or perhaps a fuzzy logic value or any other measure of that knowledge. In the end, the numerical value will be represented by a real number. Regarding reputation, there are several options, reputation could be characterized as a real number if it was obtained via a central reputation engine. The other possibility is to consider different sources of reputation for the same trustee; in this case we consider two different possible approaches: to have n different trust features or to consider an average of them. In Section 6.3, we explain why we do not consider a weighted average to prioritize certain reputation opinions. Finally, experience is usually considered as the collection of behavior evaluations with a forgetting factor. The forgetting factor is a function that favors newer interactions and places more importance on newer interactions, and respectively lower importance in older interactions. So far, we have noticed that all other inputs will use one of the previously exposed numeric measures. Therefore, we define the following way to calculate the numeric measure.

Definition 6.5. Let x be a trust feature. The numeric measure of x can be calculated in two different manners, those are the following: a) $x = v$, where $v \in \Re$, v is a direct assignment of a value given an evaluation; and b) a cumulative value, $x = \sum_{i=1}^{n} ff(i) * e(i)$, where $ff(i)$ is a forgetting factor function applied to the observation i, while $e(i)$ is the evaluation of the observation i. Please note that, new measures in the first case are interpreted as updates and updates replace old measures, on the other hand, in the second case, new measures get incorporated into the measure. In our framework, we

propose the following forgetting factor functions:

$$ff(x) = 1$$
$$ff(x) = x/n$$
$$ff(x) = x^2/n^2$$
$$ff(x) = e^{(x-n)/n}$$
$$ff(x) = log(n/(n + 1 - x))$$

Where n is the number of observations. To give a better idea how those functions behave, we show them in Figure 6.1.
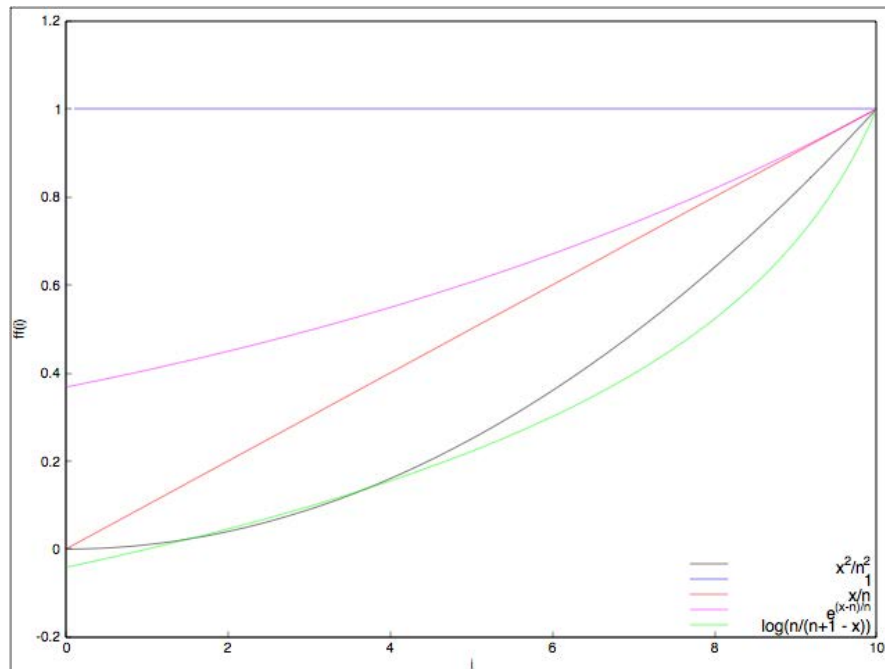

FIGURE 6.1: Forgetting factor functions

Since our goal is to entitle any device to incorporate trust notions into their interactions, we need to consider a feasible technology to exchange the trust features, contexts, trust feature measures, and trust evaluations. Nowadays, devices have very distinct capabilities, therefore we need to employ an architecture that can be used without distinction by many devices, ranging from embedded systems, wearables, and limited processing capability devices to dedicated servers and clusters. For that reason, we propose an adequate and generic architecture in which the trust management engine can be executed separately from the trustor. Nevertheless, all the functionality must be preserved. We also need to consider that, it is possible that the device can have several applications that could be interpreted as different trustors. Ideally, all those applications could use the same trust management engine.

As previously stated, we assume that entities which need to trust other entities are entities connected to a network. With this assumption, we also consider that communication in that network is possible with other systems. Therefore, we propose the use of a RESTful web-service architecture [Richardson and Ruby, 2008], in order to entitle a wide range of devices to use a trust management engine in our framework. Recently, many researchers have invested efforts to decide on the architecture that will allow on one hand specialized providers to take care of punctual tasks, and on the other hand, devices with limited capabilities to rely the processing of certain functions to more capable or more specialized entities. Many well known applications in the Internet provide web-services to interact with their applications, and moreover, low processing devices rely on such architectures [Christensen, 2009]. Also, we propose the use of JSON [Bray, 2014], as a lightweight language for data interchange. In our architecture, we assume that the handling of the traffic is following the industry standards and being sent in HTTPS when the exchange of trust data is done over a public network. Likewise, we assume that a trustors are created in the trust management system with at least one authorized key to identify its data requests; additional keys can be associated to the trustors in order to entitle third party feedback to the trust monitoring engine. The summary of the proposed RESTful API is shown in Table 6.1.

Method		Parameters	Description
GET /context/${ctx\_id}$		*ctx_id*, optional *trustor_id*, required *auth_key*, required	Gets the context information and associated trust features. If no *ctx_id* is provided, it gets all the associated contexts of the trustor.

POST /context/	*trustor_id*, required *auth_key*, required *trustP_array*, optional	Creates a new context. Adds the associated trust features to the context. The array of trust features must be in a key-value format. The key is the name of the contexts and the value should be a pair, measure and forgetting function."V" for a single value measure, and "C" for a cumulative measurement. For the forgetting function, the number associated as previously noted. Example, {"Experience_Behavior_TCP": [{"measure"·'C", "ff":1}]}, this creates a cumulative measure with $ff(x) = 1$. The result is the context object with its corresponding *ctx_id*.
POST /context/add_tp/{*ctx_id*}	*ctx_id*, required *auth_key*, required *trustP_array*, required	Adds the set of trust features, *trustP_array* to the specified context, *ctx_id*.
POST /context/del_tp/{*ctx_id*}	*ctx_id*, required *auth_key*, required *trustP_array*, required	Deletes the set of trust features, *trustP_array* to the specified context, *ctx_id*.
GET /trustee/{*trustee_id*}	*trustee_id*, required *trustor_id*, required *auth_key*, required	Gets the information related to the trustee with id, *trustee_id*.
POST /trustee/	*trustee_desc*, required *trustor_id*, required *auth_key*, required	Creates a new trustee, adds the description, *trustee_desc* to the trustee. The call returns the newly created trustee.

POST /trustee/e-val/{*trustee_id*}	*trustee_id*, required *trustor_id*, required *auth_key*, required *ctx_id*, required *trustPM_array*, required	Creates a new measurement of the trustee, *trustee_id* in the context, *ctx_id*, updating the measurements in *trustPM_array*. The measurements get replaced if the measurement is a single value, they get added into the cumulative value otherwise. An example of an element of *trustPM_array* is {"Experience_Behavior_TCP": -1}.
GET /trustee/e-val/{*trustee_id*}	*trustee_id*, required *trustor_id*, required *auth_key*, required *ctx_id*, required	Based on the trust model, the trust management engine retrieves the trust evaluation of the trustee with id, *trustee_id* in the context, *ctx_id*.

TABLE 6.1: RESTful API summary

For a better understanding of the trust data exchange in our framework, we show a simplified framework architecture in the Figure 6.2. We depict the components as blocks with the understanding that those blocks inside the dashed box can be executed in the same device or in different devices communicating through the network. First, the block depicted as the trust management engine, is in charge of the exchange of trust data from the trustors and trustor delegates (using JSON over the RESTful API). The trust data from the trustors and trustor delegates contains the evaluation of the trustees, represented by the solid arrows. Inside the trust management engine, the component which decides the trust level of the trustees based on the evaluations provided by the trustors and trustor delegates is the trust model, it is represented in a block inside the trust management engine. We consider that turstors might be different applications that can be run at the same device as the trust management engine; nonetheless, they can all be running in separate devices, they are depicted as blocks in the other end of the trust data communication. The dashed lines represent the interactions between the trustors and the trustees, from which the evaluation of the trustees behaviors can be obtained by either the trustor or its delegates. The other end of the dashed lines represent the trustees, the entities interacting with the trustors and trustor delegates for which the trustors need to assess the trust level.
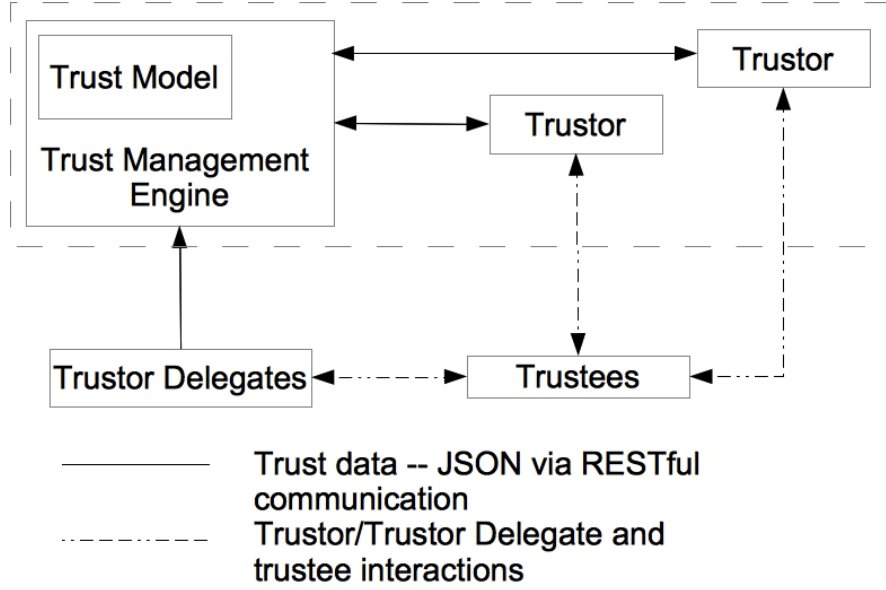
FIGURE 6.2: Generic Architecture Interaction

Finally, note that we have not specified how the trust model decides how to assess the trust data to determine the level of trustworthiness of a trustee. In our framework, we make a clear distinction of how to exchange the trust data and how it can be processed. In the Section 6.3, we present our proposed trust model.

6.3 A Machine-Learning-Based Trust Model

The trust model that a trust management engine utilizes has a fundamental role in the trust management framework. Therefore, carefully selecting an appropriate trust model for our architecture is a must. The first characteristic needed to take into account is that, our approach is context-based oriented, evaluating the trust level per context with different trust features for each context; the sets of trust features may have a different cardinality. Furthermore, the features might have different influence to the overall level of trust of the trustees. A well known approach, as used in [Toumi et al., 2012], proposes to assess the trustworthiness as a linear combination of the trust features, such that, for a context $\mathcal{C}$, the trust assessment $\mathcal{T}$, of a trustee t, is calculated as $\mathcal{T} = \sum_{i=1}^{|\mathcal{C}|} \alpha_i \cdot x_i$, where α_i is a normalized weight, an associated weight w_i is associated to the feature x_i, hence the normalized weights are commonly calculated as, $\alpha_i = w_i / \sum_{j=1}^{|\mathcal{C}|} w_j$. Often the weights are interpreted as the relevance a feature has with respect to the others, and the most common strategy of how to choose each w_i is to assign a value that can express that. For example, for a two trust feature context, if x_1 is more important than x_2, one might choose $w_1 = 5, w_2 = 10$; if both are equallly important, one might choose

$w_1 = 1, w_2 = 1$. Since the w_i values are chosen without a range the normalized weights are used to correctly express how important is w_i from the others, i.e., α_i. Additionally, adjusting the evaluations is usually desired to have the trust assessments within a known interval to be able to compare the result, and therefore decide what to do with that linear combination. There exist disadvantages to this approach, the main ones are:

- It is a hard task to choose how a given trust feature should influence globally on the trust assessment for a given context.

- The lack of flexibility of a simple weighting takes place as well. For instance, a context with n trust features in which two specific ones have little importance alone, but, high values of them combined is considered trustworthy by itself, even the rest are low. With this approach, there is no feasible way to achieve an accurate measure for this case.

- A common misconception is that with the trust level as a continuous range, a precise decision can be achieved due to the limitless numbers between the continuous range. Nevertheless, a threshold is commonly used to decide when an entity is trustworthy or not, e.g., when $\mathcal{T} > 0.2$, splitting the range between only two areas.

We would like that our trust model overcomes the previously stated drawbacks. Further, it is highly desirable, our trust model can be as flexible, and act as close as our human cognitive notion of trust. Based on this, we propose to present the assessment of trust as a multi-class classification problem and to solve it using machine learning techniques. On the one hand, the multi-class classification will provide a better refined manner to decide how to interact with the trusted entities, on the other hand, solving the problem with machine learning techniques will give the trust model a fairly close-to-human inference.

In order to achieve this, we formally define the necessary inputs for the machine learning algorithm, the set of features as the trust features in a context, i.e., $\overrightarrow{X} = \mathcal{C}$. The training sets are undoubtedly the measures stored in the trust management engine. Before applying the machine learning techniques, we need to be able to obtain the labels $y^{(i)}$, assigned for each training set, $(\overrightarrow{X}^{(i)}, y^{(i)})$, for $i = 1, 2, ..., m$, where m is the total number of trustees t, evaluated for the associated context $\mathcal{C}$, also known as the number of training sets. Then, the immediate questioning is how those labels are obtained?

At the trust management engine an administrative module is defined, first to configure the trustors and trustor delegates, specially to authorize the access through authorized keys, and second to be able to collect the training set labels. To obtain the labels, the collected measures get extracted in the trust management system and they are presented

in a view where the trustor application developers or administrators can assess to which class the measures belong. In our current approach a manual intervention is required to obtain the labels in he system. We define three trust assessment classes for our trust model, those are: "**untrustworthy**", "**neutrally trusted**", and "**trustworthy**", respectively quantified as -1, 0, and 1.

Having the training sets complete, there exist *many* machine learning algorithms to solve such multi-class classification problem [Duda et al., 2000]. Among the most popular and widely adopted algorithms to solve this, there is **logistic regression**, **artificial neural networks** and **support vector machines**.

Let us examine which machine learning algorithms will be the best suited for solving our multi-class classification problem. Based on the assumptions that the trust data can have a varied behavior, we assume the data will be seldom linearly separable. With logistic regression, to achieve non-linear classification, choosing the degree of the polynomial can be a challenging and somewhat difficult to automate task. As for artificial neural networks, designing the architecture to fit all the problems is a difficult task to automate, also, complex designs of artificial neural networks can lead to a high resource consumption. SVM can classify non-linearly separable data by applying similarity functions, also known as kernel methods. If the data is linearly separable, the linear kernel can be used. For those reasons, we have chosen to utilize SVM to solve the classification problem.

For our trust model, we show that an algorithm can be developed to find the optimal parameter configuration for support vector machines. Namely, deciding which kernel method to use and its parameters, in order to predict more accurately the trust assessment according to the particular training sets. As the authors in [wei Hsu et al., 2010] state, the **radial basis function kernel (RBFK)**[1] is suitable for most cases in exception when the number of features is *very* large, i.e., $\left| \overrightarrow{X} \right| > 10000$. We consider the number of features, i.e., trust features cannot reach a considerable high number. Nevertheless, to implement a trust model that is capable of providing accurate results for a generic training set, we consider the possibility that the data has many features or it is linearly separable, in which case the **linear kernel**[2] is better suited, and that is the reason why we consider the linear kernel as well. Also to avoid a data over-fitting, we use n-fold cross validation sets with the $\frac{1}{5}$th of the training sets to avoid this problem.

In the Algorithm 6, we propose to obtain the better suited parameters to train our SVM model. We assume $\overrightarrow{X}$, and $\overrightarrow{y}$ are inputs to our algorithm and as a solution, it obtains the best values for the parameters: $RBFK$, true if the radial basis function kernel is the

---

[1]Described in Section 6.1
[2]Described in Section 6.1

best, false if the linear kernel is; C, the cost or regularization parameter; γ, the gamma parameter of the RBFK; and *maxAccuracy*, the maximum prediction accuracy using cross validation sets. The algorithm is straightforward, it first scales the features, then, it determines the best accuracy possible, varying the value of the SVM configuration parameters. The accuracy is obtained by training the SVM and doing a n-fold cross validation on the training set. In n-fold cross validation the training set is divided into n subsets of equal size, then, one subset is tested using the classifier trained on the remaining $n - 1$ subsets. The accuracy is the percentage of correct classifications the classifier obtained from the testing set. Determining the best accuracy is done varying the parameters C, and γ over a "grid search", a combination of all pairs of the designated ranges for both parameters. In fact, a called loose grid search since the grid points are largely separated, increments of powers of two. After the optimal values are found using the loose grid search, we do a grid search with finer steps ($2^{0.25}$) in the area found by the grid search. Finally the optimal values to train the SVM are found. Note that the fixed values as 5 for the n-folds, the range $[2^{-5}, 2^{15}]$ for C, the range $[2^{-15}, 2^3]$ for γ, and the step increments of $2^{0.25}$ in the fine grid search are known and commonly recommended practical values, for more details see [wei Hsu et al., 2010].

After finding optimal parameters to train the SVM with our algorithm, we perform the training of the SVM using those parameters and obtain the trust level prediction model. This model is precisely the trust model, which predicts the trustworthiness of a trustee requested by the trustor. As an example, the trustor should obtain the response {"trustlevel": ["value": 1, "description": "trustworthy"]} for a trustee with the measures that the trust model predicts as trustworthy ($value = 1$). If the trustor requests the trust level of a trustee in a context in which the trust model is not yet trained, the following response will be obtained: {"trustlevel": ["value": 2, "description": "non trained model for context *ctx_id*"]}. When trust features are added or removed from or to the context, respectively, the previous training sets are no longer valid, then, the process of re-labeling and re-training should be conducted. Finally, If new labels are added, and based on the new labels, the prediction error goes below a configurable threshold, the trust model automatically re-trains.

6.3.1 Experimental Evaluation

We conducted the experiments by simulating trust data using two trust features, with a single context. Nowadays, different public weather service providers exist. A large amount of applications use these public service providers to present the weather for

---

Algorithm 6: SVM best parameter selection algorithm

Input: $\mathcal{TS} = (\overrightarrow{X}, Y)$
Output: optimal$(RBFK, C, \gamma, maxAccuracy)$
$RBFK \leftarrow$ true;
$C \leftarrow 0; \gamma \leftarrow 0; maxAccuracy \leftarrow 0;$
$accuracy \leftarrow 0; n = 5$;
$\overrightarrow{XS} \leftarrow scale(\overrightarrow{X})$;
for $i = -5$ **to** 15 **do**
$\quad$ **for** $j = -15$ **to** 3 **do**
$\quad\quad$ $accuracy \leftarrow xVal(n, \overrightarrow{XS}, \overrightarrow{Y}, 2^i, true, 2^j)$;
$\quad\quad$ **if** $accuracy >= maxAccuracy$ **then**
$\quad\quad\quad$ $C \leftarrow i; \gamma \leftarrow j; maxAccuracy = accuracy;$
$\quad\quad$ **end**
$\quad$ **end**
end
for $i = -5$ **to** 15 **do**
$\quad$ $accuracy \leftarrow xVal(n, \overrightarrow{XS}, \overrightarrow{Y}, 2^i, false, 0)$;
$\quad$ **if** $accuracy >= maxAccuracy$ **then**
$\quad\quad$ $C \leftarrow i; \gamma \leftarrow j; maxAccuracy = accuracy;$
$\quad\quad$ $RBFK \leftarrow$ false;
$\quad$ **end**
end
$tempC \leftarrow C, temp\gamma \leftarrow \gamma$;
for $i = tempC - 1$ **to** $tempC + 1, i \leftarrow i + 0.25$ **do**
$\quad$ **for** $j = temp\gamma - 1$ **to** $temp\gamma + 1, j \leftarrow j + 0.25$ **do**
$\quad\quad$ $accuracy \leftarrow xVal(n, \overrightarrow{XS}, \overrightarrow{Y}, 2^i, RBFK, 2^j);$
$\quad\quad$ **if** $accuracy >= maxAccuracy$ **then**
$\quad\quad\quad$ $C \leftarrow i; \gamma \leftarrow j; maxAccuracy = accuracy;$
$\quad\quad$ **end**
$\quad$ **end**
end
$C \leftarrow 2^C; \gamma \leftarrow 2^\gamma$;

---

their users. Some examples of such providers can be AccuWather[3], OpenWeatherMap[4], and Forecast.io[5]. We propose the context of trusting public forecast of weather providers, from now on referred as the context $\mathcal{C}$. In order to trust public forecast providers, we need to determine what is important in a weather forecast service. The first feature we assume to be a cumulative value of the evaluation of the TCP response time less than 1s; we want the information collected to arrive in a timely manner, this feature is addressed as $x1$. For the second trust feature, denoted as $x2$, we propose the cumulative value of the service temperature response standard deviation less than 5% of the mean. With the second trust feature, we assume the responses should not vary greatly. Randomly

---

[3] http://www.accuweather.com
[4] http://openweathermap.org
[5] http://forecast.io

we choose the simulated values for the trust features, $x1$ and $x2$, for the single context $\mathcal{C}$. We labeled each simulated measurements, $\overrightarrow{X}^{(i)}$ with our subjective perception of their trust level, respectively adding -1 for untrustworthy, 0 for neutrally trusted, and 1 for trusted, providing $y^{(i)}$. Based on the our complete training set $(\overrightarrow{X}, \overrightarrow{y})$, the plotted data looks as shown in Figure 6.3. We simulated only two trust features for the sake of simplicity, usually understanding plots is easier in two dimensions.
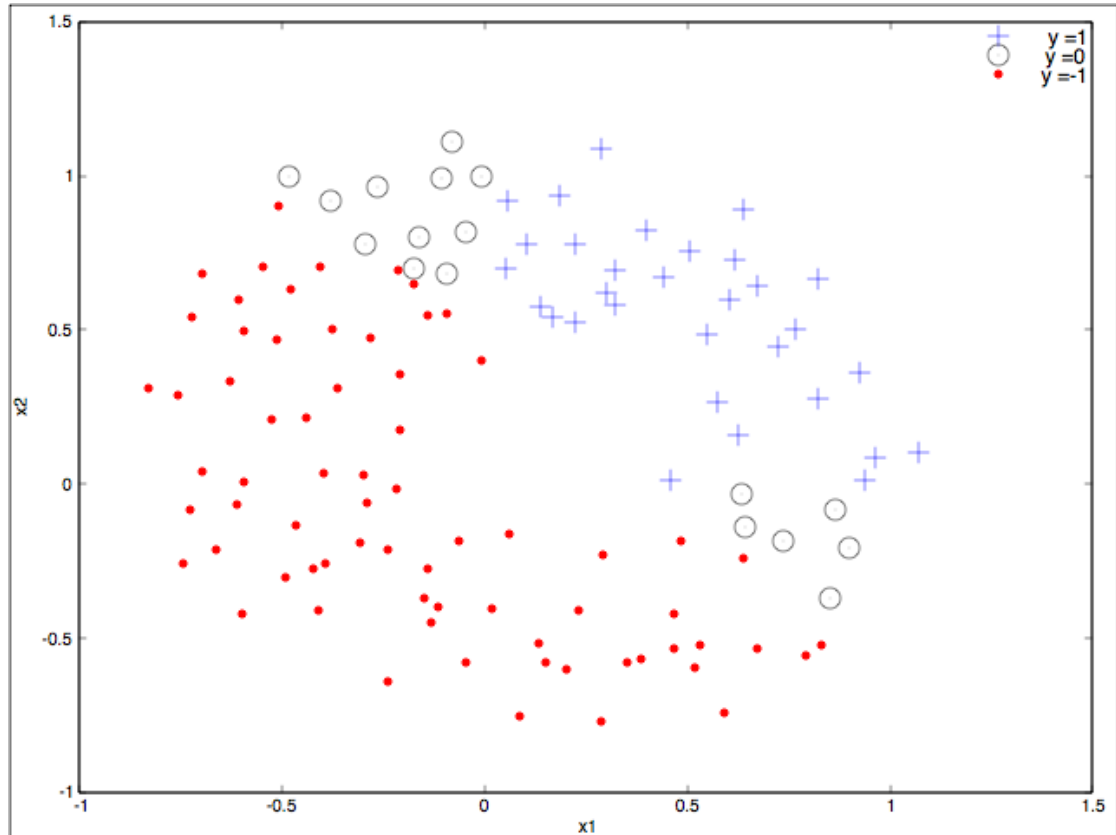

FIGURE 6.3: Simulated trust property measures

To conclude our experiments, we used the GNU Octave programming language [Eaton et al., 2008]. Then, using our algorithm we found the optimal values for C, γ, and $RBFK$. The optimal values are: $C = 2^8, \gamma = 2^{-3}, RBFK = true$. The best accuracy using the n-fold cross validation sets for the presented data obtained was 96.610169%. The worth of the presented algorithm to find the optimal values can be easily seen in here, with a $C = 2^9, \gamma = 2^{-2}, RBFK = true$, we obtain the best accuracy of 85.5932%, a very large difference compared to the slight parameter difference. The decision boundaries of a trained SVM without the optimal parameters is shown in Figure 6.4. The decision boundaries of our trained SVM with optimal parameters are shown in the Figure 6.5. It is easy to compare how the decision boundaries are more accurate to predict separate the different classes in Figure 6.5; it can be seen that the decision boundaries that

separate the classes include two different classes in some cases from the examples in the Figure 6.4(mostly seen in the neutrally trusted labels, denoted with black circles are sometimes grouped with untrustworthy labels, red dots), while they are clearly separated without mixing different classes in the Figure 6.5. To train the SVM, including the testing using the cross validations sets, we used a library, LIBSVM [Chang and Lin, 2011]. As the image shows, the accuracy of the prediction is quite good, and due to the cross set validation procedures, the over-fitting problem to this particular set of data is not a problem.
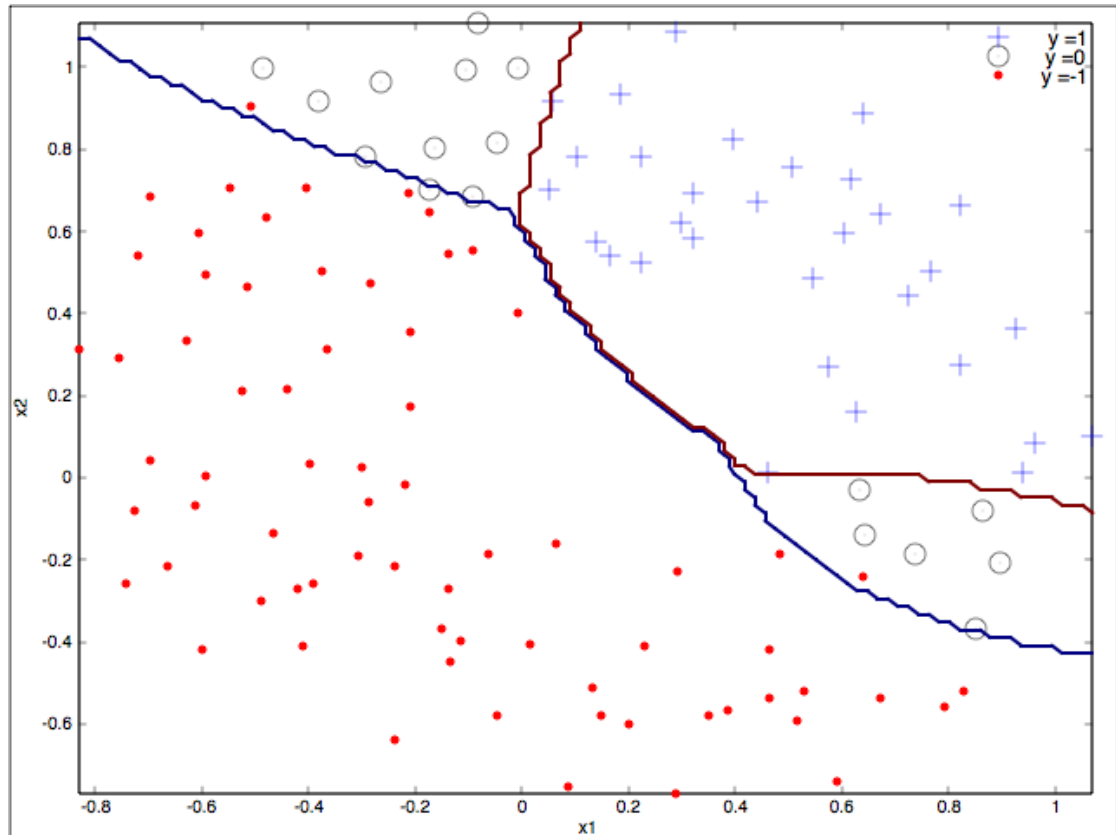

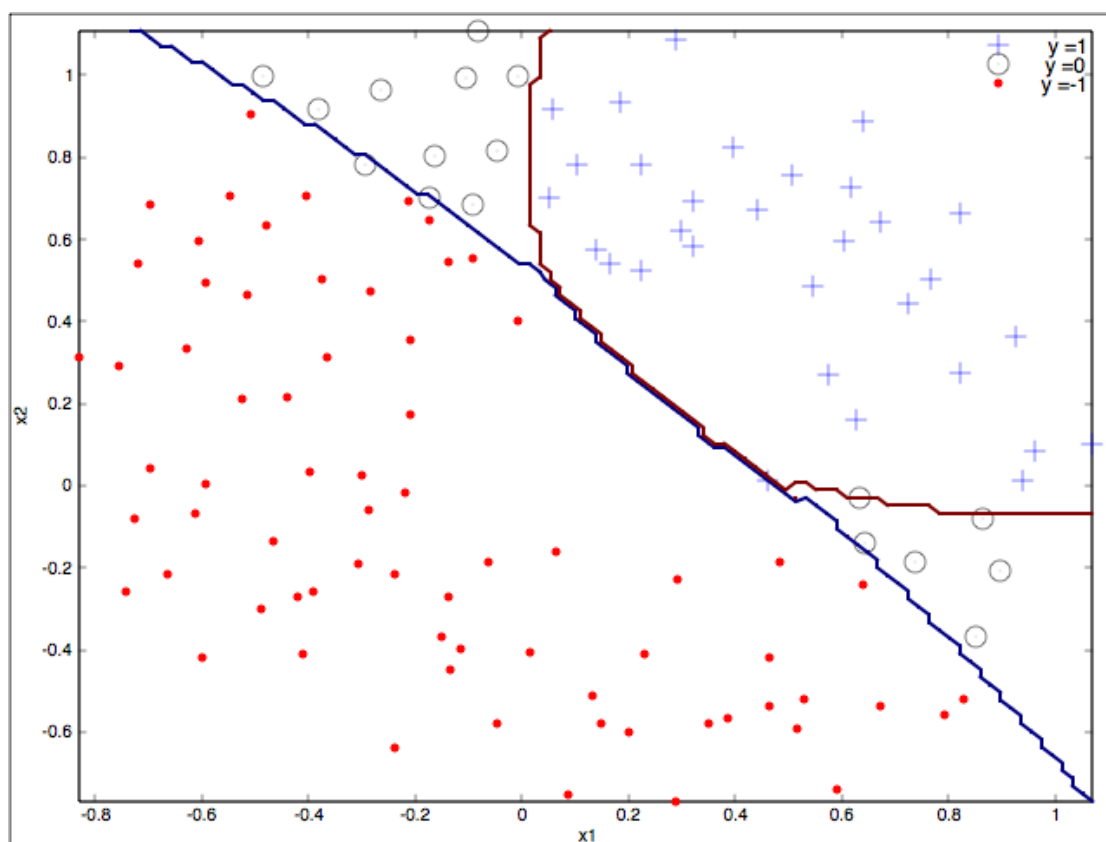FIGURE 6.4: Decision boundaries for the trained SVM without optimal parameters

FIGURE 6.5: Decision boundaries for the trained SVM using optimal parameters

Chapter 7

Conclusion and Future Work

"Do. Or do not. There is no try." — Master Yoda, The Empire Strikes Back.

As described in the Chapter 2, some remaining open issues were tackled throughout this thesis. During the reminding chapters, we have exposed our proposed solutions to this remaining open issues. The conclusion of the work performed and the perspectives for future work are described in this Chapter.

In order to present our conclusions in a better way and for greater readability, we present our conclusion separated in different sections from the perspectives. Also, for the same reason, both sections are explained in a chapter-oriented manner.

7.1 Conclusion

First, as shown in Chapter 3, we propose a novel testing methodology using a formal distributed approach, to monitor and to provide behavioral evaluation feedback regarding trust properties. We evaluate those trust properties on real distributed live captures. While most of the approaches are based on local probes (points of observations or interfaces), in this work, we provide verdicts based on the correlation of the observed live captures from distributed systems. A formal syntax and semantics are defined to express trust properties. Interesting and promising results were obtained.

Furthermore, we have implemented a suite of testing tools using our formal distributed approach to test trust properties by evaluating behaviors on on-line distributed network captures. We define a correlation of the network traffic to evaluate the behaviors in distributed systems to provide trust verdicts in a generic manner so that any trust management system can benefit from it. For the experimental results, the case scenario

is simulated within the real architecture of an industrial partner. Trust properties were verified in the industrial partner authoritative DNS server's live captures, and other DNS resolvers as points of observation.

The work done in Chapter 3 was performed by enhancing known techniques for conformance and performance black box passive testing, using network packet captures. The state of the art techniques were extended to use on-line and distributed concepts, to further fit the trust management domain requirements and open problems of that domain.

Later, in Chapter 4, we have presented an extension to our generic and extensible trust monitoring approach. In order to automatically generate trust properties, a new approach was presented aided by the use of process mining techniques. From the comparisons of two directed weighted graphs, one representing the network protocol specification, and the other model provided by as the output of a PM technique. We analyze the conformance relation between the models and conclude regarding the potentially observed faults, i.e., untrustworthy behaviors in the interactions found in the telecommunication protocols. For that purpose, we defined a mapping function from the process mining activities and our proposed language syntax, and an algorithm to compare the models to extract the trust properties. Thus, we can apply the previously stated behavioral evaluation technique based on these automatically generated properties.

The proposed approach in Chapter 4, can be described overall as proposing solutions to enhance the previously mentioned state of the art techniques to ease the usage of them and to make them more scalable. Yet another work performed in this thesis that fits this description is the one found in Chapter 5.

Precisely in Chapter 5, we have presented a scalable approach to evaluate on-line network monitoring systems. Furthermore, we have introduced an algorithm that regardless of the language used to express the monitoring properties is capable of generating an auxiliary model to evaluate them; the only requirements are the basic concepts and constraints that any on-line network monitoring system has. The proposed method after creating the data-structure uses a second algorithm that we presented in order to evaluate the packets and provide verdicts regarding them in a linear time, $O(n)$.

In addition to that, in Chapter 5, in order to express the necessary concepts of on-line distributed network monitoring, and to ease the usage of expressing the trust properties, we have presented a language modification and extension that fits any system executing testing at runtime. The language specification can be used in any domain in which on-line network monitoring is utilized. Properties of different application domains are

written in the new language. For instance, properties for conformance, performance testing, and behavioral evaluation for trust management.

Finally, in Chapter 6, we have presented a methodology in which systems, applications or devices could universally adopt trust concepts using different trust parameters for each context. Furthermore, we present the architecture, and propose an API to exchange the trust measurements and trust evaluations using RESTful web-services. In addition to it, we present a suitable trust model for the approach, by solving a multi-class classification problem with machine learning techniques, namely using support vector machines. Further, we have presented an algorithm which is capable to find the best parameters of the SVM to get better accuracy to predict the trustee's trust levels by using n-fold cross validation sets. Finally, we simulate trust measurements and provide a trust model which has a prediction accuracy of the 96.61%.

7.2   Future Work

With the work realized by this thesis, we manage to close remaining open issues in the related domains. As well, as a natural consequence, interesting aspects to work on, appeared. In this Section, we describe some of the interesting perspectives for future work.

The obtained results in Chapter 3, are very valuable, nevertheless, further testing of our method would be valuable to prove its scalability. Although the theoretical complexity and current experiments indicate that the scalability of our approach is good; having several case studies to have a better estimation about the correlation between the physical resources and our tool capabilities would be highly valuable, to show in detail the overhead introduced by our proposed tools.

An interesting perspective is, to add different and configurable weights to each part of the trust property evaluations. This will provide behavioral feedback with an accurate evaluation of the level of completion of the given trust property.

Additionally, we note that the traces may be collected by our server at different periods and then eventually with an important delay (due to the diverse queries/responses along the testing time). For the time being, as expressed in Chapter 3, we rely on the systems having time synchronized between them using the well known NTP protocol. An interesting perspective is to create a time synchronization handshake periodically between the P.Os and the monitoring server, to establish the difference in time of each P.O and using a common reference.

We consider that the comparison of the values in the trust properties can be done against constant values or values of stored packets. However, an interesting extension will be to compare against stored statistical results from these multiple points of observation.

We plan to extend our **extmon** tool in order to provide a generic interface for different systems to be able to send their interesting traffic. This will allow any external monitoring platforms to acquire messages from remote systems as soon as possible, and to perform any kind of analysis within a reduced time delay. In order to improve the exmon scalability, we plan to send not the complete packets, but, only the necessary fields used in checked trust properties.

Likewise, it is undeniable that applying our approach and tools to different domains and protocols will be interesting to prove the versatility of the tool.

Finally, proposing different methods which might be more scalable, as formalizing the "*test objectives*" as FSMs and simply checking if the input sequence belongs to the language of the machine is an interesting perspective. Nevertheless, consideration for messages (inputs) which are not included in the original protocol description (language), but that are accepted by the machine implementation is an area to pay attention.

Concerning the approach presented in Chapter 4, this method seems to provide relevant results, there are still issues to resolve and perspectives to target. Indeed, the generated properties are highly dependent to the monitored traces from which the process mining tool is applied. Therefore, it does not guarantee that these properties will be applicable on all protocol traces. This could then cause the creation of several inconclusive verdicts. Furthermore, we depend on "an external" formal protocol specification. We aim at creating this model by PM techniques through an "ideal" controlled system in which trust issues would not occur. Further, we aim at including the concept of violation in trust properties in order to consider both concepts, faults that should be avoided and trust properties that are expressed as expected behaviors.

Our contribution for the Chapter 5 of this thesis focuses on two things. The first one is to provide verdicts in a scalable manner and as stated in Section 5.3, the algorithm highly depends on the length of the queues of previously stored packets. Second, to enhance the expressiveness of the language to make it generic enough for any given protocol and on-line constrains. Therefore, incorporating the enhancements into our tool set is included into our perspectives.

Finally, since the approach described in Chapter 6 seems promising, we plan to implement the generic trust management engine to combine all the capabilities and to experiment using trust measures provided from the feedback provided by our previously

developed set of tools as shown in Chapter 3. Further, we plan to increase some features in the trust management engine, for example, adding more choices to calculate experiences or accumulated measurements; adding more options to calculate forgetting factor functions is also included. Even if the justification to use SVM to solve our problem is fair, selecting the better suited parameters to train the model requires to train *several* SVM models. For that reason, comparing the efficiency of a SVM model and an artificial neural network model, for instance is an interesting perspective. Further, many machine learning algorithms should be compared and probably many options for the model should be given. Another interesting perspective is that, due to the fact that we consider that the multi-class option gives great flexibility to the trustors we plan to further investigate expanding the presented trust classes. Yet another interesting perspective is the possibility to include different trust models to provide the capability of deciding which trust model to use. Finally, we also plan to investigate the applicability of other machine learning techniques to the trust models, especially non-supervised machine learning techniques to suggest a pre-classification without the need of explicitly providing the labels of the data set.

Bibliography

Alexiou, N., Basagiannis, S., Katsaros, P., Dashpande, T., and Smolka, S. A. (2010). Formal analysis of the kaminsky dns cache-poisoning attack using probabilistic model checking. In *Proceedings of the 12th IEEE High Assurance Systems Engineering Symposium, HASE, San Jose, CA, USA*, pages 94–103.

Arends, R., Austein, R., Larson, M., Massey, D., and Rose, S. (2005). Dns security introduction and requirements. RFC 4033 (Proposed Standard).

Artz, D. and Gil, Y. (2007). A survey of trust in computer science and the semantic web. *Elsevier Web Semantics*, 5(2):58–71.

Azzini, A. and Ceravolo, P. (2013). Consistent process mining over big data triple stores. In *Big Data (BigData Congress), 2013 IEEE International Congress on*, pages 54–61. IEEE.

Backus, J. W. (1959). The syntax and semantics of the proposed international algebraic language of the zurich acm-gamm conference. In *Proceedings of the International Conference on Information Processing, UNESCO*, pages 125–132.

Bayse, E., Cavalli, A. R., Núñez, M., and Zaïdi, F. (2005). A passive testing approach based on invariants: application to the WAP. *Computer Networks*, 48(2):235–245.

Becchi, M., Wiseman, C., and Crowley, P. (2009). Evaluating regular expression matching engines on network and general purpose processors. In *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '09, pages 30–39, New York, NY, USA. ACM.

Blaze, M., Feigenbaum, J., and Keromytis, A. D. (1999). Keynote: Trust management for public-key infrastructures. In *Proceedings of the Security Protocols, 6th International Workshop, Cambridge, UK*, pages 59–63. Springer.

Blaze, M., Feigenbaum, J., and Lacy, J. (1996). Decentralized trust management. In *Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, USA*, pages 164–173.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152.

Bray, T. (2014). Rfc 7159 – the javascript object notation (json) data interchange format.

Cavalli, A. R., Maag, S., de Oca, E. M., and Zaiadi, F. (2009). A passive conformance testing approach for a manet routing protocol. In *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC), USA, March 9-12*, pages 207–211.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27.

Che, X., Lalanne, F., and Maag, S. (2012a). A logic-based passive testing approach for the validation of communicating protocols. In *Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE, Wroclaw, Poland*, pages 53–64. SciTePress.

Che, X., Lalanne, F., and Maag, S. (2012b). A logic-based passive testing approach for the validation of communicating protocols. In *Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering, Wroclaw, Poland, 29-30 June*, pages 53–64.

Che, X., López, J., Maag, S., and Morales, G. (2015). Testing trust properties using a formal distributed network monitoring approach. *Annals of Telecommunications, DOI: 10.1007/s12243-014-0454, IF: 0.408*, 70(3-4):95–105.

Christensen, J. H. (2009). Using restful web-services and cloud computing to create next generation mobile applications. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, Orlando, Florida, USA*, pages 627–634.

Chu, Y.-H., Feigenbaum, J., Lamacchia, B., Resnick, P., and Strauss, M. (1997). Referee: Trust management for web applications. *O'Reilly World Wide Web Journal*, 2(3):127–139.

Clausen, T. and Jacquet, P. (2003). *RFC 3626 Optimized Link State Routing Protocol (OLSR)*. Internet Engineering Task Force.

Dagon, D., Provos, N., Lee, C. P., and Lee, W. (2008). Corrupted dns resolution paths: The rise of a malicious resolution authority. In *Proceedings of the Network and Distributed System Security Symposium, NDSS, San Diego, California, USA*. The Internet Society.

Deering, S. and Hinden, R. (1998). *RFC 2460 Internet Protocol, Version 6 (IPv6) Specification.* Internet Engineering Task Force.

Deng, N., Tian, Y., and Zhang, C. (2012). *Support vector machines: optimization based theory, algorithms, and extensions.* CRC Press.

Dierks, T. (2008). The transport layer security (tls) protocol version 1.2. *RFC 5246.*

Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification (2Nd Edition).* Wiley-Interscience.

Eaton, J. W., Bateman, D., and Hauberg, S. (2008). *GNU Octave Manual Version 3.* Network Theory Ltd.

El-Fakih, K., Kolomeez, A., Prokopenko, S., and Yevtushenko, N. (2008). Extended finite state machine based test derivation driven by user defined faults. In *Proceedings of the First International Conference on Software Testing, Verification, and Validation, ICST 2008, Lillehammer, Norway*, pages 308–317.

El-Fakih, K., Prokopenko, S., Yevtushenko, N., and von Bochmann, G. (2003). Fault diagnosis in extended finite state machines. In *Proceedings of the 15th IFIP International Conference, TestCom, Sophia Antipolis, France, DOI: 10.1007/3-540-44830-6_15*, pages 197–210.

Falcone, R. and Castelfranchi, C. (2001). Social trust: A cognitive approach. In *Trust and Deception in Virtual Societies*, pages 55–90.

Fan, L., Wang, Y., Cheng, X., and Li, J. (2011). Prevent dns cache poisoning using security proxy. In *Proceeding of IEEE 12th International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2011, Gwangju, Korea*, pages 387–393.

Fan, R.-E., Chen, P.-H., and Lin, C.-J. (2005). Working set selection using second order information for training support vector machines. *J. Mach. Learn. Res.*, 6:1889–1918.

Faro, A. and Petrenko, A. (1990). Sequence generation from EFSMs for protocol testing. In *Proceedings of the COMNET, Budapest*, pages 17–26.

Freier, A., Karlton, P., and Kocher, P. (2011). The secure sockets layer (ssl) protocol version 3.0. *RFC 6101.*

Grandison, T. and Sloman, M. (2000). A survey of trust in internet applications. *IEEE Communications Surveys and Tutorials*, 3(4):2–16.

Grandison, T. and Sloman, M. (2003). Trust management tools for internet applications. In *Proceedings of Trust Management, Springer First International Conference, iTrust, Heraklion, Crete, Greece*, pages 91–107.

Günther, C. W. and Van Der Aalst, W. M. (2007). Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *Business Process Management*, pages 328–343. Springer.

Haidar, D. A., Cuppens-Boulahia, N., Cuppens, F., and Debar, H. (2009). Xena: an access negotiation framework using xacml. *Annales des Télécommunications*, 64(1–2):155–169.

Hauke, S., Biedermann, S., Mühlhaüser, M., and Heider, D. (2013). On the application of supervised machine learning to trustworthiness assessment. In *Proceedings of the IEEE 12th International Conference on Trust, Security and Privacy in Computing and Communications, Melbourne, Australia*, pages 525–534.

Holzmann, G. J. (2004). *The spin model checker : primer and reference manual*.

Houy, C., Fettke, P., Loos, P., van der Aalst, W. M., and Krogstie, J. (2011). Business process management in the large. *Business & Information Systems Engineering*, 3(6):385–388.

Hsu, C.-W. and Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425.

Irfan, M.-N., Oriat, C., and Groz, R. (2013). Model inference and testing. *Advances in Computers*, 89:89–139.

Jim, T. (2001). Sd3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy, Oakland, California, USA*, pages 106–115.

Kushik, N., Kolomeez, A., Cavalli, A. R., and Yevtushenko, N. (2014). Extended finite state machine based test derivation strategies for telecommunication protocols. In *Proceedings of the SYRCoSE of the Institute for System Programming of the Russian Academy of Sciences*, pages 108–113.

Lalanne, F. and Maag, S. (2013). A formal data-centric approach for passive testing of communication protocols. *IEEE/ACM Transactions on Networking*, 21(3):788–801.

Lee., A. J., Winslett, M., and Perano, K. J. (2009). Trustbuilder2: A reconfigurable framework for trust negotiation. In *Proceedings of Trust Management III, Third IFIP WG 11.11 International Conference, IFIPTM, West Lafayette, IN, USA*, pages 176–195.

Lee, D., , Chen, D., Hao, R., Miller, R. E., Wu, J., and Yin, X. (2006). Network protocol system monitoring-a formal approach with passive testing. *IEEE/ACM Transactions on Networking*, pages 14(2):424–437.

Lo, C.-C., Huang, C.-C., and Ku, J. (2010). A cooperative intrusion detection system framework for cloud computing networks. In 280-284, editor, *Proceedings of the IEEE 39th International Conference on Parallel Processing Workshops*.

López, J., Che, X., Maag, S., and Morales, G. (2014a). A distributed monitoring approach for trust assessment based on formal testing. In *28th IEEE International Conference on Advanced Information Networking and Applications (AINA), Victoria, Canada, DOI: 10.1109/WAINA.2014.114, Rank B*, pages 702–707.

López, J. and Maag, S. (2015a). Distributed on-line network monitoring for trust management. *Russian Physics Journal (in press)*.

López, J. and Maag, S. (2015b). Towards a generic trust management framework using a machine-learning-based trust model. In *Proceedings of the 14th IEEE Internationt al Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-15), Helsinki, Finland*, pages 1343–1348.

López, J., Maag, S., and Morales, G. (2013). A formal distributed network monitoring approach for enhancing trust management systems. In *Proceedings of the 5th ACM International Conference on Management of Emergent Digital EcoSystems, MEDES, Luxembourg, Luxembourg, DOI: 10.1145/2536146.2536160*, pages 76–83.

López, J., Maag, S., and Morales, G. (2014b). Scalable evaluation of distributed on-line network monitoring for behavioral feedback in trust management. *Proceedings of the Institute for System Programming of the Russian Academy of Sciences, DOI: 10.15514/ISPRAS-2014-26(6)-11*, 26(6):125–140.

López, J., Maag, S., and Morales, G. (2015a). Behavior evaluation for trust management based on formal distributed network monitoring. *World Wide Web, DOI: 10.1007/s11280-015-0324-6, Rank A, IF: 1.623*.

López, J., Maag, S., and Morales, G. (2015b). Tear: a multi-purpose formal language specification for testing at runtime. In *ARES Conference*, pages 727–734.

López, J., Maag, S., Saint-Pierre, C., Bustos, J., and Cavalli, A. (2015c). Process mining for trust monitoring. In *29th IEEE International Conference on Advanced Information Networking and Applications (AINA) Workshops, Gwangju, Korea, Rank B*, pages 605–610.

Mallouli, W., Wehbi, B., de Oca, E. M., and Bourdellès, M. (2012). Online network traffic security inspection using mmt tool. In *Proceedings of the 24th International Conference on Software & Systems Engineering and their Applications (ICSSEA2012), 9th Workshop on System Testing and Validation (STV) Workshop, Paris, France.*

Marsh, S. P. (1994). *Formalising Trust as a Computational Concept.* PhD thesis, University of Stirling, Stirling, Scotland, UK.

McCanne, S. and Jacobson, V. (1993). The bsd packet filter: a new architecture for user-level packet capture. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference, San Diego, California.*

Mellouk, A., Sherif, M. H., Li, J., and Bellavista, P., editors (2014). *Ad Hoc Networks - 5th International ICST Conference, ADHOCNETS 2013, Barcelona, Spain, October 2013, Revised Selected Papers*, volume 129. Springer.

Mills, D. L. (1991). Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493.

Mockapetris, P. V. (1987a). *RFC 1034 Domain names — Concepts and Facilietes.* Internet Engineering Task Force.

Mockapetris, P. V. (1987b). *RFC 1035 Domain names — implementation and specification.* Internet Engineering Task Force.

Momani, M. and Challa, S. (2010). Survey of trust models in different network domains. *ACM Computing Research Repository*, abs/1010.0168.

Movahedi, Z., Nogueira, M., and Pujolle, G. (2012). An autonomic knowledge monitoring scheme for trust management on mobile ad hoc networks. In *IEEE Wireless Communications and Networking Conference, WCNC 2012, Paris, France*, pages 1898–1903.

Petrenko, A., Boroday, S., and Groz, R. (2004). Confirming configurations in EFSM testing. *IEEE Trans. Software Eng.*, 30(1).

Poggi, N., Muthusamy, V., Carrera, D., and Khalaf, R. (2013). Business process mining from e-commerce web logs. In *Business Process Management*, pages 65–80. Springer.

Postel, J. (1981). Transmission control protocol. RFC 793, Internet Engineering Task Force.

Ray, I. and Chakraborty, S. (2004). A vector model of trust for developing trustworthy systems. In *Computer Security - ESORICS, 9th European Symposium on Research Computer Security, Sophia Antipolis, France*, pages 260–275. Springer.

Richardson, L. and Ruby, S. (2008). *RESTful web services.* O'Reilly Media, Inc.

Roesch, M. et al. (1999). Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238.

Roschke, S., Cheng, F., and Meinel, C. (2010). A flexible and efficient alert correlation platform for distributed ids. In *Proceedings of the IEEE Fourth International Conference on Network and System Security, NSS, Melbourne, Victoria, Australia*, pages 24–31.

Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E. (2002). Sip: Session initiation protocol, rfc 3261. Technical report, Internet Engineering Task Force.

Rozinat, A. (2013). *Disco Tour*. Fluxicon.

Smith, R., Estan, C., Jha, S., and Kong, S. (2008). Deflating the big bang: Fast and scalable deep packet inspection with extended finite automata. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 207–218, New York, NY, USA. ACM.

Song, W. and Phoha, V. V. (2004). Neural network-based reputation model in a distributed system. In *Proceedings of the IEEE International Conference on E-Commerce Technology*, pages 321–324.

Toumi, K., Andrés, C., and Cavalli, A. R. (2012). Trust-orbac: A trust access control model in multi-organization environments. In *Proceedings of Information Systems Security, 8th International Conference, ICISS, Guwahati, India*, pages 89–103.

Toumi, K., Mallouli, W., de Oca, E. M., Andrés, C., and Cavalli, A. R. (2014). How to evaluate trust using MMT. In *Proceedings of the Network and System Security - 8th International Conference, NSS, Xi'an, China*, pages 484–492.

Van der Aalst, W., Weijters, T., and Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *Knowledge and Data Engineering, IEEE Transactions on*, 16(9):1128–1142.

van der Aalst, W. M. P. (2011). *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag Berlin Heidelberg Springer-Verlag Berlin Heidelberg.

van Emden, M. H. and Kowalski, R. A. (1976). The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742.

Wang, J., Yi, Z., Zurada, J., Lu, B.-L., and Yin, H. (2006). A novel graph kernel based svm algorithm for image semantic retrieval. In *Advances in Neural Networks - ISNN 2006*, Lecture Notes in Computer Science. Springer Berlin Heidelberg.

Wang, X., Liu, X., Matwin, S., and Japkowicz, N. (2014). Applying instance-weighted support vector machines to class imbalanced datasets. In *Proceedings of the IEEE International Conference on Big Data (Big Data), Washingtong, DC, USA*.

Wang, Y. and Vassileva, J. (2003). Bayesian network-based trust model. In *Proceedings of the IEEE/WIC International Conference on Web Intelligence*, pages 372–378.

wei Hsu, C., chung Chang, C., and jen Lin, C. (2010). A practical guide to support vector classification.

Yu, H., Shen, Z., Miao, C., and An, B. (2012). Challenges and opportunities for trust management in crowdsourcing. In *Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology - Volume 02*, pages 486–493.

Zargar, S. T., Takabi, H., and Joshi, J. B. D. (2011). Dcdidp: A distributed, collaborative, and data-driven intrusion detection and prevention framework for cloud computing environments. In 332-341, editor, *Proceedings of IEEE 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom, Orlando, FL, USA*.

Zhang, L., Song, M., Liu, X., Bu, J., and Chen, C. (2013). Fast multi-view segment graph kernel for object classification. *Signal Processing*, 93(6):1597 – 1607. Special issue on Machine Learning in Intelligent Image Processing.

Zhao, K. and Pan, L. (2014). A machine learning based trust evaluation framework for online social networks. In *Proceedings of the IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, Beijing, China*, pages 69–74.

Titre : Monitorage En-Ligne et Distribué de Réseaux pour L'Évaluation de la Confiance

Mots clés : Confiance, Monitorage Distribué de Réseaux, Systèmes En-ligne

Les systèmes collaboratifs sont aujourd'hui devenus très populaires et sont de plus en plus utilisés dans de nombreux domaines divers. De fait, les interactions *de confiance* des differents systèmes sont devenus une priorité. La confiance, en tant que concept informatique, a été étudiée très récemment. Cependant, dans la littérature, très peu d'attention a été portée pour évaluer l'exactitude des interactions entre entités communicantes; même si la plupart des approches se sont basées sur les mesures cumulées de ces valeurs. Pour déterminer, de façon générale, l'exactitude de ces interactions, une approche nommée Monitorage des Réseaux En-Ligne et Distribué (MRED) a été proposée. De plus, des outils prototypes ont été développés pour tester automatiquement les *propriétés de confiance* entre entités dans des systèmes communicants. MRED est une forme de test passif; elle analyse les réponses des systèmes et teste l'exactitude des interactions en utilisant des traces de réseaux. Comme elle dépend des *propriétés* à tester, une nouvelle approche a été proposée pour faire l'extraction automatique de propriétés pertinentes que l'ont pourrait, in fine, tester dans un système sous test. Aussi, nous proposons de nouvelles méthodes afin d'améliorer les techniques fournies dans l'état de l'art pour: a) évaluer efficacement les propriétés avec une complexité en temps $O(n)$, ce en utilisant un Automate Fini Déterministe Prolongée (EFSA); et b) élargir l'expressivité du langage proposé pour exprimer correctement les contraintes systèmes, comme *les délais d'attente* pour éviter le manque de ressources. Finalement, nous proposons un nouveau cadre flexible utilisable dans de très nombreux domaines, qui permet la définition de *caractéristiques de confiance* afin d'évaluer les entités dans des contextes différents. De surcroît, avec les évaluations des caractéristiques de confiance, nous proposons un modèle de confiance basé sur l'apprentissage automatique, utilisant des Machine à vecteurs de support (SVM). A partir de ces modèles, des expérimentations ont été effectuées en simulant des caractéristiques de confiance pour estimer le *niveau de confiance*; une précision de plus de 96% a été obtenue.

Title : Distributed On-line Network Monitoring for Trust Assessment

Keywords : Trust ,Distributed Network Monitoring, On-line Systems

Collaborative systems are growing in use and in popularity. The need to boost the methods concerning the interoperability is growing as well; therefore, *trustworthy* interactions of the different systems are a priority. Trust as a computer science concept has been studied in the recent years. Nevertheless, in the literature, very little focus is placed on how to assess the correctness of the interactions between the entities; even if most approaches rely on the estimation of trust based on the accumulated measures of these values. To broadly determine the correctness of interactions without targeting a specific domain or application, an approach using Distributed On-line Network Monitoring (DONM) was proposed. Furthermore, a prototype tool-set was developed to automatically test the *trust properties*. DONM is a form passive testing; it analyzes systems' responses and test the correctness of the interactions via network traces. Since it relies on the stated *properties* to test, a novel approach was proposed to automatically extract relevant properties to test. Our approach deeply relies on the operation of On-line Monitoring Systems. That is the reason why we propose new methods to enhance the state of the art techniques to: a) efficiently evaluate properties in $O(n)$ time complexity using an Extended Finite State Automata (EFSA) auxiliary data structure; and b) to expand the language expressiveness to properly express the constraints of such systems, such as, *timeouts* in order to avoid resource starvation. Finally, using the evaluation of the entities' interactions provided by our approach, trust management engines will help *trustors* to decide with whom and how to interact with other users or applications. We propose a new framework that is flexible for any domain, allowing trustors to define the *trust features* used to evaluate trustees in different contexts. Furthermore, with the evaluations of the trust features, we propose a trust model which achieves close-to-human inference of the trust assessment, by using a machine learning based trust model, namely solving a multi-class classification problem using Support Vector Machines (SVM). Using the SVM-based trust model, experiments were performed with simulated trust features to estimate *trust level*; an accuracy of more than 96\% was achieved.