Game-Theoretic Foundations for Forming Trusted Coalitions of Multi-Cloud Services in the Presence of Active and Passive Attacks

Omar Abdul Wahab

A Thesis
In
The Concordia Institute
for
Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Information and Systems Engineering) at
Concordia University
Montréal, Québec, Canada

October 2017

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Omar Abdul Wahab**

Entitled: **Game-Theoretic Foundations for Forming Trusted Coalitions of Multi-Cloud Services in the Presence of Active and Passive Attacks**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Information and Systems Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Hua Ge	Chair
Dr. Robin Cohen	External Examiner
Dr. Juergen Rilling	Examiner
Dr. Rachida Dssouli	Examiner
Dr. Roch Glitho	Supervisor
Dr. Jamal Bentahar	Co-supervisor
Dr. Hadi Otrok	

Approved by

Dr. Chun Wang Graduate Program Director

November 22, 2017

Date of Defence

Dr. Amir Asif Dean, Faculty of Engineering and Computer Science

Abstract

Game-Theoretic Foundations for Forming Trusted Coalitions of Multi-Cloud Services in the Presence of Active and Passive Attacks

Omar Abdul Wahab, Ph.D.
Concordia University, 2017

The prominence of cloud computing as a common paradigm for offering Web-based services has led to an unprecedented proliferation in the number of services that are deployed in cloud data centers. In parallel, services' communities and cloud federations have gained an increasing interest in the recent past years due to their ability to facilitate the discovery, composition, and resource scaling issues in large-scale services' markets. The problem is that the existing community and federation formation solutions deal with services as traditional software systems and overlook the fact that these services are often being offered as part of the cloud computing technology, which poses additional challenges at the architectural, business, and security levels.

The motivation of this thesis stems from four main observations/research gaps that we have drawn through our literature reviews and/or experiments, which are: (1) leading cloud services such as Google and Amazon do not have incentives to group themselves into communities/federations using the existing community/federation formation solutions; (2) it is quite difficult to find a central entity that can manage the community/federation formation process in a multi-cloud environment; (3) if we allow services to rationally select their communities/federations without considering their trust relationships, these services might have incentives to structure themselves into communities/federations consisting of a large number of malicious services; and (4) the existing intrusion detection solutions in the domain of cloud computing are still ineffective in capturing advanced multi-type distributed attacks initiated by communities/federations of attackers since they overlook the attacker's strategies in their design and ignore the cloud system's resource constraints.

This thesis aims to address these gaps by (1) proposing a business-oriented community formation model that accounts for the business potential of the services in the formation process to motivate the participation of services of all business capabilities, (2) introducing an inter-cloud trust framework that allows services deployed in one or disparate cloud centers to build credible trust relationships toward each other, while overcoming the collusion

attacks that occur to mislead trust results even in extreme cases wherein attackers form the majority, (3) designing a trust-based game theoretical model that enables services to distributively form trustworthy multi-cloud communities wherein the number of malicious services is minimal, (4) proposing an intra-cloud trust framework that allows the cloud system to build credible trust relationships toward the guest Virtual Machines (VMs) running cloud-based services using objective and subjective trust sources, (5) designing and solving a trust-based maxmin game theoretical model that allows the cloud system to optimally distribute the detection load among VMs within a limited budget of resources, while considering Distributed Denial of Service (DDoS) attacks as a practical scenario, and (6) putting forward a resource-aware comprehensive detection and prevention system that is able to capture and prevent advanced simultaneous multi-type attacks within a limited amount of resources.

We conclude the thesis by uncovering some persisting research gaps that need further study and investigation in the future.

Acknowledgements

I would like to express my gratitude to Almighty GOD for granting me the health, ability, and patience to complete this thesis.

I would like to thank my Ph.D. supervisors Dr. Jamal Bentahar, Dr. Hadi Otrok and Dr. Azzam Mourad for their continuous guidance and full support. I was lucky to have been surrounded by such professional, inspirational, and caring advisors. Thanks for having believed in me and motivated me to pursue my graduate studies. You gave me the opportunity and curiosity to explore my academic capacity and paved the way to what I am now.

Moreover, I would like to thank my Ph.D. committee members Dr. Rachida Dssouli, Dr. Juergen Rilling, Dr. Roch Glitho, and Dr. Robin Cohen for their valuable time and effort in reviewing my thesis and providing me with insightful comments and recommendations. Your deep knowledge and expertise have made every encounter a great opportunity to discuss new ideas and enhance the quality of the research outcomes.

Furthermore, I would like to thank all my colleagues in the research lab at Concordia University especially Ahmad Bataineh, Mona Taghavi, Gaith Rjoub, Nagat Drawel, Afaf Mousa, and Laura Zapata-Aspiazu for providing me with a warm and friendly atmosphere to work in. I would like also to thank my non-concordian friends Hanine Tout and Adel Abusitta as well as my non-academic friends with whom I shared precious and enjoyable moments. Your presence has allowed me to appreciate my stay in Montreal and added lots of fun to my Ph.D. journey.

This research would not have been possible without the financial assistance of the Fonds de recherche du Québec - Nature et technologies (FRQNT) and Concordia University. This support was very important for me to alleviate the financial burdens and focus on my research duties.

Last but not least, I would like to thank my parents for their endless and unconditional love and support. Their guidance has been always important for me to overcome the difficulties of life. Without my mother (Mona), father (Mohammad), sister (Amanie), and two brothers (Amine and Ayman), I could not have been able to succeed throughout my life and become the person I am today.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we introduce the context of our research work, highlight the problems tackled in this thesis, pose the corresponding research questions, and finally identify the goal and objectives of our research work.

1.1 Research Context and Motivations

Services' communities[1] [82] have been proposed as an effective platform for addressing the discovery, composition, and resource scaling problems in the Service-Oriented Architecture (SOA). The idea is to group services sharing the same functionality into a set of homogenous clusters. Although numerous community formation models can be found in the literature, these models deal with Web services as traditional software systems and overlook the fact that these services are being (often) offered nowadays as part of the cloud computing technology, which raises additional challenges at the business, architectural, and security levels. Consequently, the aim of our research work is to propose a novel community-based services' architecture that accounts for these aforementioned new constraints. We refer to such an architecture as *Multi-Cloud Services' Communities*. The system model of this architecture consists of a set of services deployed in different cloud centers and belonging to different cloud services' layers as depicted in Fig. 1.1.

Fortunately, such an extension extends the benefits provided by the community-based architecture. In fact, besides the aforementioned benefits of the traditional services' communities, multi-cloud services' communities provide additional benefits for both providers and customers. Practically, such an architecture increases the flexibility of providers in

[1] In the rest of the thesis, the terms *coalition* and *community* are used interchangeably.

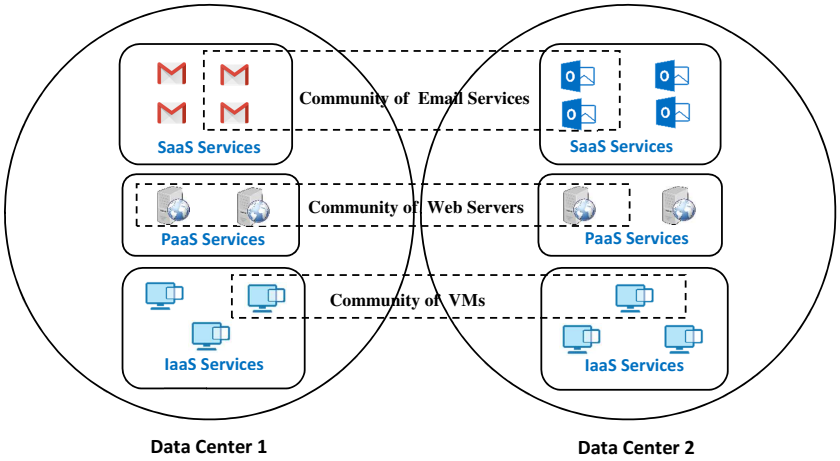


Figure 1.1: Architecture of the multi-cloud services communities

managing requests and meeting the Service Level Agreement (SLA) requirements by allowing them to move workloads from one provider's site to another in a seamless manner and allowing the cooperation among services belonging to different clouds. Moreover, the referred architecture allows for redundant computation in the sense that the same application logic may be deployed at different providers' sites, where every incoming request gets performed by more than one service instance. The motivation is to improve the availability by reducing the risk of an instance failing before completing the task. Similarly, the referred architecture allows us to break down the customer's data at the bit level in order to enable its parallel processing by service instances sharing the same application logic but residing at different providers' sites. Besides optimizing latency and throughput, protecting the privacy and security of the data is an additional motivation for customers to favor such a model, where each provider can be aware only of a small part of the data [65].

1.2 Problem Statement and Research Questions

The current community formation models [20, 166, 91, 21, 82, 70, 73, 63], although designed to work well when all services reside in a single cloud center, are unable to support a multi-cloud community formation model due to several limitations at the architectural, business, and security levels. From the architectural point of view, the common trend in forming services' communities is the use of a Service-Level Agreement (SLA) [24] contract to regulate the formation process, where a certain service is statically designated as a master for the community to control the formation and security issues. However, the distributed nature of cloud-based services plays against the existence of such a central entity. Practically, the fact that services are deployed in data centers that are located in disparate

parts of the World means that these services are managed by different parties and are owned by different providers. Therefore, finding a common third party that is trusted by all providers and that can manage all these services is quite challenging. This leads us to our first research question (RQ1):

- **RQ1: How can we decentralize the community formation process in order to allow the formation of communities gathering services that reside at different cloud centers?**

From the business point of view, the current community formation approaches overlook the business potential of the services during the formation process and treat all of those services in the same manner, which demotivates the contribution of the well-positioned services in that process. This makes the existing models ineffective in modeling a multi-provider multi-cloud environment wherein clouds have uneven business capabilities. Practically, in the real markets, the services that are deployed in strong cloud centers are likely to enjoy high reputation and market share, which makes them refuse to be treated in the same manner as the other services that are deployed in less strong cloud centers. Thus, our second research question (RQ2) is:

- **RQ2: How can we design the community formation model in such a way to motivate the participation of services belonging to different clouds having uneven business capabilities?**

From the security perspective, the existing community formation models rely on a honest or semi-honest adversary model wherein services and/or the community master are assumed to be trustworthy. However, in a multilateral multi-cloud environment whereby multiple clouds and providers are involved, malicious services are likely to exist. Practically, some services might misbehave by unilaterally deviating from the agreements made with the other services upon community formation (e.g., refuse to lend computing resources such as Central Processing Unit (CPU) and memory). Such malicious services are referred to as *passive malicious services* [151] since their objective is to illegally maximize their own benefits but they do not have a direct intention to attack and harm the communities/other services [150]. Therefore, our third research question (RQ3) is:

- **RQ3: How can we ensure the formation of trustworthy multi-cloud services' communities that minimize the number of passive malicious members?**

On the other hand, *active malicious services*, that launch active attacks (e.g., Denial of Service), find the multilateral nature of the community-based architecture an appealing

platform to carry out their malicious attacks. Worse, the severity and consequences of such attacks become more catastrophic when services are deployed in a cloud environment because of the virtual and elastic nature of cloud systems. Practically, in a cloud environment, multiple clients' services (concretized as Virtual Machines (VMs)) are allowed to share a single computing infrastructure. This exposes both the VMs and infrastructure to additional security threats, besides those that exist in the traditional computing systems [45]. Moreover, the elastic nature of cloud systems that offers users the ability to freely scale resources, constitutes a serious vulnerability from which attackers can execute painful attacks (e.g., Economic Denial of Sustainability (EDoS)). For example, some attackers might compromise VMs and manipulate them to send a large number of fake resources scaling requests; thus leading to drain the resources of the cloud system and make it unavailable to support further VMs. Unfortunately, the existing cloud-dedicated Intrusion Detection Systems (IDSs) consider the detection problem from the IDS's perspective solely and overlook the strategies of the attacker, which results in high false alarms and resources wastage during detection. Furthermore, the current IDSs do not explain how to deal with the cloud system's limited security resources problem in the detection process, thus assuming (directly or indirectly) that the cloud system is able to provide permanent and full detection coverage on all of its nodes. However, it is no secret that the magnitude of resources that can be devoted to detection is bounded by a certain budget that is determined in such a way that does not affect the portion of resources dedicated to serving clients. This necessitates thinking of resource-aware selective detection strategies that distribute the cloud system's detection load among the different VMs so that it respects the limited security resources' budget and maintains at the same time an optimal detection effectiveness. Knowing all these facts, our fourth research question (RQ4) is:

- **RQ4: How can we maximize the detection of active attacks using a limited budget of resources and knowing that attackers exploit the cloud's features to minimize this maximization?**

1.3 Research Objectives and Contributions

The ultimate goal of our research work is to form trusted multi-cloud communities among services having uneven business capabilities in the presence of active and passive malicious services. This goal can be divided into the following sub-objectives:

- **Objective 1:** Develop a distributed community formation model that motivates the participation of services belonging to different clouds having uneven business capabilities.

- **Objective** 2**:** Design a trust-based coalition formation model that allows services deployed in one or different cloud data centers to distributively group themselves into trusted communities, while avoiding the passive malicious services that renege on their agreements and seek to illegally maximize their profits.

- **Objective** 3**:** Elaborate a resource-aware intrusion detection strategy for the active malicious services in the multi-cloud community-based architecture that maximizes the detection of active intelligent multi-type attacks using a limited budget of resources.

The research methodology and objectives are highlighted in Figure 1.2 w.r.t the identified research questions (Section 1.2). In order to attain those objectives, the following contributions are offered by this thesis:

- **Contribution** 1**:** We conducted a systematic literature review on the concepts of trust and reputation in the SOA, discussed the main challenges of designing and implementing trust and reputation models in this area, identified the main research gaps, and highlighted some interesting research ideas for possible consideration in the future (**This contribution is discussed in Chapter 3**).

- **Contribution** 2**:** We developed a Stackelberg game theoretical model that enables services of different business capabilities to distributively structure themselves into communities (**This contribution is discussed in Chapter 4**).

- **Contribution** 3**:** We designed a comprehensive trust framework at the services' level followed by a trust-based hedonic game theoretical model that allows services belonging to one or multiple cloud centers to form up trustworthy communities wherein the number of passive malicious services is minimal (**This contribution is discussed in Chapter 5**).

- **Contribution** 4**:** We introduced a comprehensive trust framework between the cloud system and its guest VMs followed by a trust-based maxmin game theoretical model that allows the cloud system to optimally distribute the detection load among its guest VMs in such a way that maximizes the detection of DDoS attacks and maintains at the same time a reasonable budget of security resources (**This contribution is discussed in Chapter 6**).

- **Contribution** 5**:** We elaborated a repeated Bayesian Stackelberg game theoretical model which enables the cloud system to detect multi-type intelligent attacks and provides a defense mechanism to minimize the number of attacked services (**This contribution is discussed in Chapter 7**).

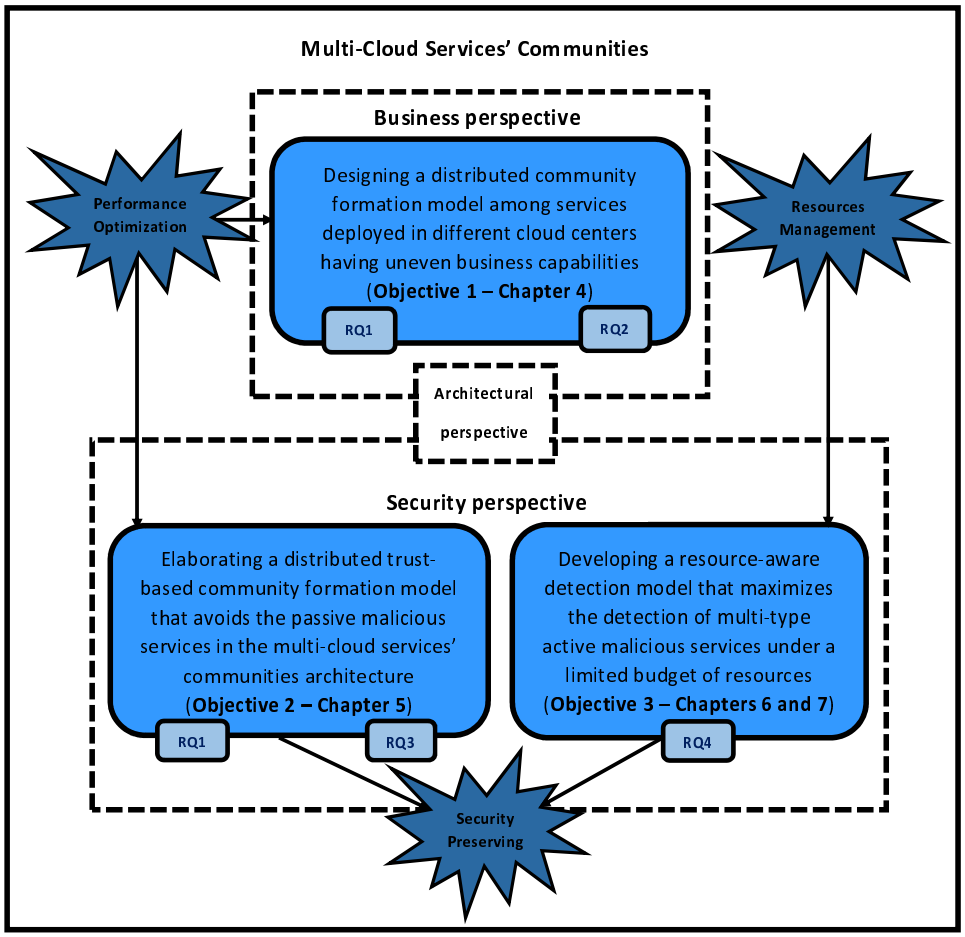


Figure 1.2: Research methodology and objectives w.r.t the identified research questions

It is worth noting that the content of this thesis has been published in [141, 144, 152, 142, 143, 146, 145].

1.4 Thesis Structure

We present in Chapter 2 the background needed to understand the different concepts of our research work. In particular, we give an overview of the main concepts of the SOA, namely those of Web service, cloud computing, cloud federation, community of services, trust, and reputation. Then, we give a detailed tutorial on the concept of game theory and explain its main categories using plain text, mathematical foundations, and illustrative examples. Afterwards, we provide literature reviews on the cooperation models proposed for SOA applications, the main approaches that used Stackelberg game theory to solve business and security problems, and the major intrusion detection systems proposed in

the domain of cloud computing.

In Chapter 3, we advance a systematic survey on the concepts of trust and reputation in the SOA by proposing a classification scheme for the services according to their architecture and offering a collection of criteria that are important to guarantee the effectiveness of any trust and reputation model for each particular services' architecture. Based on this classification scheme, we compare the existing trust and reputation models with regards to the proposed criteria and highlight some potential research gaps that need further investigation.

In Chapter 4, we discuss the business-oriented community formation model and highlight its industrial impact by means of a practical example that shows how this model can be effectively applied in practical markets applications.

In Chapter 5, we tackle the problem of forming trustworthy multi-cloud services' communities in the presence of passive malicious services. In particular, we put forward a comprehensive trust framework among services deployed in disparate cloud centers and propose a trust-based game theoretical model that allows these services to form trusted multi-cloud communities.

In Chapter 6, we address the problem of active malicious services and propose a resource-aware detection strategy that aids the cloud system to optimally distribute the available detection load among its guest VMs so as to maximize the detection of DDoS attacks.

In Chapter 7, we discuss a comprehensive detection and prevention mechanism for multi-type active malicious services that is able to cope with advanced simultaneous attack scenarios. Illustrative examples and experimental results are advanced in the different chapters to validate the effectiveness and efficiency of our solutions.

Finally, in Chapter 8, we summarize the thesis contributions and shed light on some research gaps that need further consideration by the research community.

Chapter 2

Background and Literature Review

In this chapter, we explain the main concepts that are needed to understand the core of the thesis and present profound and systematic literature reviews on the different aspects that our thesis aims to address. In Section 2.1, we explain the main concepts related to the SOA, namely those of cloud computing (Section 2.1.1) and Web services (Section 2.1.2). We define as well the notions of cloud federation (Section 2.1.3), community of services (Section 2.1.4), and trust and reputation (Section 2.1.5). In Section 2.2, we give a detailed tutorial on the concept of game theory, propose a classification scheme for the game theoretical models on the basis of the situation that they are designed to model, and explain the main types of game theory by means of theoretical foundations and numerical examples. In Section 2.3, we begin the detailed literature reviews on the main concepts addressed in our thesis. In particular, Section 2.3.1 is dedicated to discussing the main cooperation models proposed for SOA applications; particularly those of services' communities and cloud federations. Section 2.3.2 surveys the main approaches that used Stackelberg game theory for solving business and security problems in the literature. Finally, in Section 2.3.3, we classify the existing intrusion detection systems proposed for cloud-based applications into three major categories and discuss each category in detail in terms of advantages and limitations.

2.1 Service-Oriented Architecture (SOA)

In this section, we explain the main concepts related to the SOA, namely those of cloud computing, Web service, cloud federation, services community, and trust and reputation.

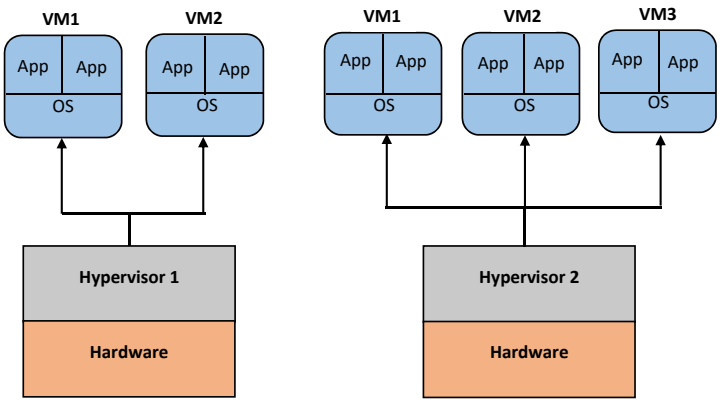


Figure 2.1: Virtualized cloud system: Hypervisors manage a set of hardware resources and host a set of VMs

2.1.1 Cloud Computing

Cloud computing can be regarded as a new paradigm of computing in which dynamically scalable (often virtualized) resources are being offered as services via the Internet [17]. These services are presented as a layered cloud computing architecture that consists of three main layers: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [42]. SaaS provides on-demand running of software applications remotely from the cloud. PaaS consists of computing platforms offered as a service such as operating systems, Web servers, databases, and programming languages' execution environments. IaaS provides users with computing resources such as virtualized computers whose processing power, bandwidth, and Internet access are guaranteed. Based on the foregoing definitions, one can easily observe the advantages of cloud computing, which are mostly related to the high availability of services, cost efficiency, and scalability in accommodating larger workloads.

Cloud systems capitalize on the concept of virtualization that helps pool infrastructure resources and achieve better agility and flexibility. As depicted in Fig. 2.1, a virtualized cloud system consists of a set of hardware resources $I = \{I_1, I_2, \ldots, I_k\}$ managed by a set of hypervisors $H = \{h_1, h_2, \ldots, h_n\}$; each of which is hosting a set of virtual machines $V_i = \{v_1, v_2, \ldots, v_l\}$ owned by a set of clients $C = \{c_1, c_2, \ldots, c_m\}$, where each client $c_i \in C$ owns one or more virtual machines . A virtual machine v_i is a pair $\langle O, A \rangle$, where O represents the operating system and A is the corresponding applications running inside v_i. A hypervisor $h_i \in H$ is a software agent residing between the cloud system's hardware and the guest VMs and whose goal is to allow the concurrent running of multiple operating systems abstracted as VMs on a shared hardware. Thus, the role of the hypervisor consists of emulating the hardware system and scheduling the access of the VMs to it.

Definition 1 (**Virtualized Cloud System).** *A virtualized cloud system consists of a set of hardware resources* $I = (I_1, ..., I_n)$ *managed by a set hypervisors* $H = (h_1, ..., h_n)$ *hosting a set of virtual machines* $V = (v_1, ..., v_l)$ *owned by a set of clients* $C = (c_1, ..., c_m)$ *to provide each* $v \in V$ *a view that its operating system and applications are operating directly on some physical hardware.*

2.1.2 Web Service

A Web service is a software application that can be integrated into other Web-based applications over the Internet using open standards such as XML, UDDI, SOAP, and WSDL. From a technical perspective, Web services might be implemented through different ways. The most two common ways of implementing and delivering Web services are those of *Simple Object Access Protocol (SOAP) Web Services* and *Representational State Transfer (RESTful) Web Services* [110]. SOAP Web services employ Extensible Markup Language (XML) messages for communication and consist of machine-readable characterization of the functionalities and operations brought by the services written using the Web Services Description Language (WSDL), an XML language for syntactical definition of interfaces. On the other hand, in RESTful web services, data and operations are treated as resources that are accessed and retrieved using Uniform Resource Identifiers (URIs). This type of Web services is based on a client/server architecture and is engineered to use a stateless communication protocol, namely the Hypertext Transfer Protocol (HTTP). The decision to use SOAP or RESTful Web services depends heavily on the requirements of the application at hand, where SOAP Web services are well-suited for enterprise applications integration scenarios that demand high Quality of Service (QoS)[1] requirements whereas RESTful Web services are often used for integrating applications over the Web. The emergence of agent-based software engineering [161, 53] is changing the notion of Web services from passive components to autonomous service agents. These agents are able to interact with one another in order to adapt to the real-time challenges (e.g., composition). This has the advantage of moving the micro-level management responsibilities from the user's side to the service's side. In this way, the user has only to specify the high-level goals (i.e., the "what"), leaving the details of executions (i.e., the "when" and "how") to the service agents [81]. Specifically, a service agent is a computer program that is encapsulated in the services to provide them with flexible and autonomous actions/reactions in response to the environment within which they operates and helps them hence meet their design objectives [53].

[1] The term "QoS" describes the overall performance delivered by the service as a result of a certain invoke.

2.1.3 Cloud Federation

A cloud federation [118] is a practical platform for joining together cloud computing environments of two or more providers with the aim of scaling up their capacity of handling a larger pool of users' requests. A federated cloud's scenario involves conducting an agreement among a set of cloud providers by which one or more commit to sell or rent out computing resources to other providers in order to expand the latter ones' capacity. Two major benefits are brought to providers by the cloud federation architecture. First, providers are given the chance to earn increased revenue through selling/renting their additional computing resources which might end up being under-utilized if not well-exploited. Second, by being part of a certain federation, providers have the opportunity of enlarging their geographical presence in new areas and countries without having to establish new points-of-presence. We provide in Section 2.3.1 of this chapter a detailed literature review on the main approaches that contributed in solving problems related to cloud federations.

2.1.4 Community of Services

A community of services is a virtual cluster grouping services sharing the same functionality but having different non-functional (i.e., QoS) properties. Communities allow services to interact and cooperate in order to deliver higher-levels of quality, performance, and interoperability. For example, services within a community may replace each other in case of execution problems. Such an architecture is beneficial for both providers and users. On the one hand, services residing in a community will have more chances to receive a bigger task pool from users and to be exposed to a greater number of services compositions. Moreover, by joining communities enjoying good reputation, market share, and capacity, providers will increase their chances of receiving more requests and increasing hence their total revenue. On the other hand, the users would be more satisfied with such an architecture since their requests are processed with better quality, availability, responsiveness, and performance. The main difference between cloud federations and services communities is that the former architecture operates at the provider's level whereas the latter one operates at the service's level. This expands the benefits provided by the services community architecture to cover, in addition to the resources scaling and geographical expansion, the optimization of the services composition processes, the facilitation of the services discovery, and the ease of security and privacy preservation. We present in Section 2.3.1 of this chapter a detailed literature review on the main approaches that contributed in solving problems related to services communities.

2.1.5 Trust and Reputation in the SOA

According to the Concise Oxford Dictionary, *"Reputation is what is generally said or believed about a person's or thing's character or standing"* [55]. Informally, reputation represents the combined measure of reliability inferred by feedback or ratings gathered from members in a certain community. Trust can be defined as *"a subjective probability an agent has about another's future behavior"* [43]. That is, the degree of trustworthiness one agent assigns to another agent/group of agents in performing a certain action [55]. The main difference between these two concepts is that trust is mainly a personal and subjective notion in contrary to the reputation which is public and combined. In other words, one agent A may still trust another agent B despite B's bad reputation if A has a close relation to B or even has some private information about B that surpass B's public reputation. Numerous trust and reputation models have been proposed for many open systems [78, 151, 150, 72, 131]. In the SOA, a trust and reputation model is a method that enables decision makers to distinguish good services from bad ones based on users' feedback. The importance of trust and reputation models stems from their ability to enable users and service providers to differentiate among the services that offer similar functionalities on the basis of how well these services behaved in the past history. This helps them make thoughtful selections and avoid the bad choices since making random choices in such an open system may expose users/providers to quality, cost, and even security problems. Practically, any provider has the freedom to publish bad-quality, expensive, and even harmful services, which makes wise selections of great importance. In the next chapter, we offer a detailed and systematic review on the main trust and reputation models proposed in the SOA, highlight the main challenges of building a trust and reputation model, and provide future insights that help researchers investigate novel trust and reputation models based on the literature's persisting research gaps.

2.2 Game Theory

Game theory is a formal study of conflict and cooperation in a multi-agent interactive environment [104]. It provides mathematical tools for modeling and delineating automated decision-making frameworks for rational agents in strategic scenarios. Such agents may represent individuals, machines, firms, software, or any grouping or combination of them. By rational agents we mean those agents whose ultimate goal is to maximize their own benefits. For this purpose, each agent is characterized by a *utility function* which quantifies its profit/loss resulting from any strategy adopted during the game. That is, each agent has clear preferences, is aware of the set of choices and alternatives, builds expectations about unknowns and uncertainties, and performs some optimization process before taking

Table 2.1: Game theory components

Symbol		Meaning
Agent	:⇔	A party that is entitled to act for or on behalf of another principal in order to get this principal into contractual relationships with other parties.
Game	:⇔	A formal characterization of a strategic situation between conflicting and/or co-operating self-interested rational agents.
Player	:⇔	An agent who is supposed to make decisions in the game.
Rational	:⇔	Profit maximizer.
Action	:⇔	A single move or step made by a player in the game.
Strategy	:⇔	In strategic-form non-cooperative games, a strategy is equivalent to an action. In extensive-form non-cooperative games, a strategy is a complete plan of actions available to a certain player during the different stages of the game.
Payoff/Utility	:⇔	A positive or negative value assigned to a player as a reward or punishment in response to a certain action during the game.
Solution Concept	:⇔	Methodical characterization of the manner according to which the game will be played by the players based on the best possible strategies and their associated outcomes.
Best Response	:⇔	The strategy (or strategies) that yield the largest payoff for a certain player given what the other players are doing.
Nash Equilibrium	:⇔	A solution concept for non-cooperative games in which each player uses the strategy that is the best response to the strategies of the other players such that no player has incentives to deviate unilaterally from its current strategy and adopt another strategy.
Dominant Strategy	:⇔	The strategy that generates the largest payoff value for its player no matter what strategies will be played by the opponents.
Coalition	:⇔	Group of players.
Grand Coalition	:⇔	The coalition of all players.
Worth of a Coalition	:⇔	The total payoff of a coalition in Transferrable Utility coalitional games.
N	:⇔	The set of all players.
$S \backslash \{i\}$	:⇔	Coalition S without player i.
Coalition Structure/Partition	:⇔	A set that partitions the players into disjoint coalitions.

actions. A strategic scenario is an interactive scenario in which the actions/benefits of one agent are influenced by the actions of one or more other agents. The basic idea is that individual decisions have influence on each other's welfare. Binmore [25] considers that whenever an interaction between two entities takes place, a game (in the context of game theory) is being played. For example, when a student and his supervisor discuss about next year's scholarship, they are playing a game! When small and big companies negotiate about forming up a joint alliance to increase their power in handling customers' requests, it also a game! The main advantages that distinguish game theory from the other optimization tools lie in the high level of abstractness that characterizes its tools and representations as well as its consideration of the different agents' actions/reactions during the optimization process. These appealing features have caused game theoretical models to be widely used in numerous critical domains such as: economics, security, networks, and social sciences [105]. Before pursuing our discussions, it is worth noting that the different notations related to game theory that will be used throughout this chapter are explained in Table 2.1.

As depicted in Fig. 2.2, the main classification that is still adopted till today is that of Von Neumann and Morgenstern [95] who classified game theoretical models into two main

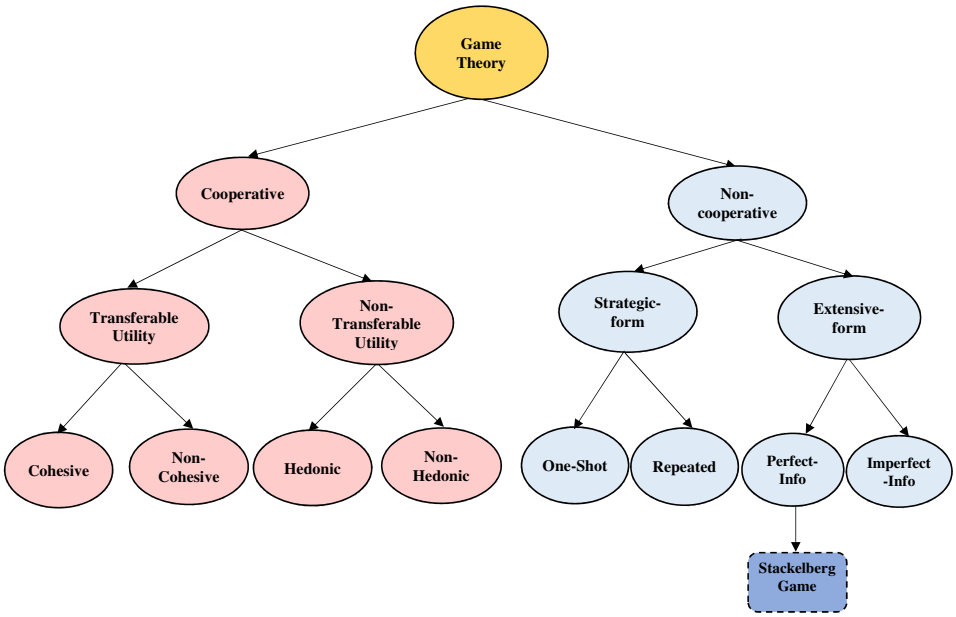


Figure 2.2: Classification scheme: Game theoretical models are classified based on the underlying situations

branches: *cooperative game theory*, and *non-cooperative game theory*. In contrary to what is suggested by the names, the difference between these two branches is not measured by the degree of cooperation that each model allows among agents. More specifically, the competition may be as extreme in cooperative game theory as in non-cooperative games. In fact, cooperative game theory models situations in which agents believe that forming coalitions will create an added-value that is in the benefit of all of them. Therefore, cooperative games are often referred to as coalitional games[2]. Nonetheless, this does not mean that the agents in such a kind of games are in a pure cooperative mood. Indeed, each player in the coalition is in a continuous competition with his/her colleagues to maximize his/her own share of profit produced by the coalition. Moreover, each coalition is often in an extreme competition with the other coalitions (e.g., political parties). On the other hand, non-cooperative game theory analyzes situations wherein each agent acts individually and tries to maximize his/her own payoff by choosing the best response to the actions of the other agents.

Each of these branches may be also divided into several sub-classes based on the underlying situations. Cooperative game theory can be classified into two subclasses: *Transferable Utility (TU)* games, and *Non-Transferable Utility (NTU)* games. In TU games, the payoff (i.e., profit) of the coalition is worth the same for all the players who, as a result,

[2]In the rest of this thesis, the terms *cooperative game theory* and *coalitional game theory* are used interchangeably.

can compare, distribute, and transfer this payoff (e.g., money). TU games may be either *cohesive* or *non-cohesive*. Cohesive TU games rely on the *super-additivity* property, which assumes that the larger the coalition is, the more its payoff will be. Thus, the goal of cohesive games is to group all the players into one *single* coalition referred to as *grand coalition*. Non-cohesive games, on the other hand, assume that the *super-additivity* property does not always hold and aim hence at generating *disjoint* coalitions. For the NTU games, the payoff of the coalition is non-comparable, non-distributable, and non-transferrable (e.g., happiness). NTU games can be classified into *hedonic* and *non-hedonic* games. In hedonic games, players have preferences over the coalitions and the preference assigned by each player to a certain coalition depends solely on the members of that coalition regardless of the structure of the other coalitions. In non-hedonic games, all the coalitions' partitions are considered by the players when building their preferences.

Non-cooperative games can be classified into *strategic-form* games and *extensive-form* games. Strategic-form games are those that are played *simultaneously* among agents. These games may be either *one-shot* (i.e., played once) or *repeated* (i.e., played over multiple runs). Extensive-form games are played *sequentially*, i.e., some player(s) play first and then play the other(s). Extensive-form games may be either of *perfect information* or *imperfect information*. In perfect-information extensive-form games, each player is perfectly informed about the actions that have been previously played by the other player(s) (e.g., chess games). On the contrary, imperfect-information extensive-form games model situations in which the actions played by some players are not always known by the other players (e.g., poker games). Additionally, imperfect-information extensive-form games can model situations in which the player does not remember his/her own actions that have already been played.

2.2.1 Coalitional Game Theory

A coalitional game is a model that is oriented to analyze the interactions among the decision makers when they gather into groups. Unlike non-cooperative games, a set of actions and payoffs are associated with every group of players and not only with individual players. The output of the coalitional game is a partition of the players' set into coalitions along with an action associated with each coalition in the partition. The following subsections are dedicated to explaining the two main classes of coalitional games: Transferable Utility (TU) games, and Non-transferable utility (NTU) games, respectively.

2.2.1.1 Transferable Utility (TU) Games:

In transferable utility games, the choice of actions of each coalitional group of players specifies the payoff of the whole coalition, which can be transferred and distributed among the coalition members. In other words, agents form coalitions and perform some common actions and obtain, as a group, a real-valued payoff that can be divided among them.

Definition 2. *A TU coalitional consists of the pair* (N, v)*, such that:*

- N : *is a finite set of players; and,*
- $v : 2^N \longmapsto \mathbb{R}$ *is a characteristic function that assigns a real-valued payoff* $v(S)$ *called "worth" to every coalition* $S \subseteq N$*, where this worth is distributable among the coalition members* $i \in S$*.*

In order to illustrate well the idea behind TU coalitional games, we give in what follows an example in which three software engineering companies are willing to collaborate and form coalitions in order to perform a joint project.

Example 1. *Assume that the three software engineering companies, SAP, IBM, and Ericsson are negotiating about developing a new joint project. It is anticipated that the project will yield a profit of $4M if these three companies collaborate altogether. However, if each of these companies decides to undertake the project by its own, then the profit is expected to be $0.5M for SAP alone, $0.6M for IBM alone, and $0.5M for Ericsson alone. Obviously, the example falls under the category of TU coalitional games due to the following reason. In this example, the payoff of the group is the maximum amount of money that they can achieve by collaborating together. Since this amount is expressed in a universal currency (i.e., USD), then players can distribute this amount and each player will have the chance to compare its own payoff when being in different coalitions and select the coalition that maximizes this payoff. Thus, the TU coalitional game of the example is the pair* (N, v)*, where:*

- $N = \{SAP, IBM, Ericsson\}$
- $V(N) = 4,\ V(\{SAP\}) = 0.5,\ V(\{IBM\}) = 0.6,\ and\ V(\{Ericsson\}) = 0.5.$

Most of the theory in TU coalitional games is dedicated to studying cohesive games in which the motivation for coalescing is extreme [104]. That is, the formation of the single coalition that includes all the players referred to as the grand coalition is in the best of all the players. In such situations, the grand coalition can yield outcomes that are at least as attractive for every player as those realizable by any partition of the players into sub-coalitions. The main research directions in this context aim to study the *stability* of the

grand coalition and the *fairness* of the payoff distribution mechanisms among the members of that coalition [125]. In order to better explain what is meant by stability and fairness, let's complete the story of Example 1. In this example, each pair of companies can also still coalesce and exclude one company from that collaboration even though the profit will be reduced. For example, if SAP and IBM collaborate together without Ericsson, their profit is expected to be \$3.1M. Similarly, SAP and Ericsson may decide to collaborate together without IBM and yield a profit of \$3M. If IBM and Ericsson decide to collaborate together without SAP, their anticipated profit is \$3.3M. Now, a new problem of how to distribute the gain among coalition members in such a way to guarantee the formation of the grand coalition arises. If the equal fair allocation model is adopted, then both SAP and IBM will get a payoff of \$1.43M (i.e., $\frac{4}{3}$) in the grand coalition and \$1.5M (i.e., $\frac{3.1}{2}$) by excluding Ericsson. Thus, the grand coalition is not stable in this case as SAP and IBM have incentives to deviate and form their own sub-coalition without Ericsson. Therefore, guaranteeing the stability is essential for the formation of the grand coalition. Nevertheless, a stable payoff distribution may not be fair. Practically, for guaranteeing stability, companies are not being rewarded based on their contributions to the coalition's payoff, but merely in such a way that makes each of them satisfied. Thus, a new question of how to distribute the payoff of the coalition in such a way to guarantee the fairness and reflect each company's contribution to the coalition arises.

These problems can be summarized by the two following questions: (1) which coalition(s) will compose? and (2) after coalitions are composed and the worth value is produced, how should this value be divided among coalition members? The key idea in answering the first question is to guarantee the *stability* of the coalitions in the sense that no members should be better off deviating from their current coalitions to other more profitable coalitions. The most famous concept that is used to analyze this property is the *core* [121]. For the second question, the key idea is to guarantee the *fairness* among players when distributing the worth of the coalition. The well-known solution concept of *Shapley value* is widely used to foster this property due to its ability to provide a unique means for worth distribution in a fair manner [123].

Core. The core is a stability solution concept whose goal is to guarantee that all the players have incentives to cooperate and form the grand coalition. In other words, the core is a payoff distribution mechanism that aims to ensure that all the members are satisfied within the grand coalition and having no incentive to leave it. A payoff distribution mechanism is said to be in the core if it satisfies the following axioms [11]:

- **Efficiency:** a payoff distribution is efficient if it allocates the whole worth of the grand coalition to all the players so that no utility is lost at the level of the population.

- **Individual rationality:** a payoff distribution is said to be individually rational if it makes each agent prefer to join a coalition rather than acting alone.
- **Group rationality:** a payoff distribution is said to be group rational if the sum of payoffs given to the players in the grand coalition is at least the value of any other coalition that may form.

Definition 3. *A payoff distribution $x \in \mathbb{R}^n$ is said to be in the core of the game (N, v) if it is efficient, individually rational, and group rational, i.e., $Core(N, v) = \{x \in \mathbb{R}^n | \sum_{i \in N} x(i) = x(N)$ and $\forall\ coalition\ C \subseteq N, x(N) \geq v(C)\}$.*

Intuitively, a payoff distribution lies in the core if it assigns the whole gain of the grand coalition to its members in such a way that demotivates any member from deviating and acting alone and any subset of members from deviating and forming their own sub-coalition. Thus, the formation of the grand coalition is guaranteed if the core exists and is non-empty. It is worth mentioning that the condition of individual rationality is not clearly expressed in Definition 3 since group rationality directly implies individual rationality.

In Example 1, a payoff distribution for the three software engineering companies *[x(SAP), x(IBM), x(Ericsson)]* lies in $Core(N, v)$ if it satisfies the conditions presented in Equation (1)

$$\begin{cases} x(SAP) + x(IBM) + x(Ericsson) = 4\ (Efficiency); \\ x(SAP) \geq 0.5\ (Individual\ Rationality); \\ x(IBM) \geq 0.6\ (Individual\ Rationality); \\ x(Ericsson) \geq 0.5\ (Individual\ Rationality); \\ x(SAP) + x(IBM) \geq 3.1\ (Group\ Rationality); \\ x(SAP) + x(Ericsson) \geq 3\ (Group\ Rationality); and \\ x(IBM) + x(Ericsson) \geq 3.3\ (Group\ Rationality) \end{cases} \tag{1}$$

Shapley Value. Shapley value takes its name from Lloyd S. Shapley who introduced this fairness solution concept in 1953 [122]. Shapley suggests that each player i in the coalition S should receive an amount that reflects its contribution to that coalition, i.e., $v(S \cup \{i\}) - v(S)$. In order to implement this vision, Shapley states that the fair payoff distribution mechanism should satisfy the following axioms [98]:

- **Efficiency:** As described in the context of the core, efficiency means that the total worth of the grand coalition should be distributed among its members.
- **Dummy player:** This property labels the players who do not make any contribution to the coalition as *dummy* and emphasizes hence that they should receive nothing.

- **Symmetry:** This property states that the players who make the same contribution to the coalition should be assigned the same amount of payoff.
- **Additivity:** This is mostly a technical property saying that the payoff value should be additive over the set of all games, i.e., for any two TU games (N, v) and (N, ω) and corresponding payoff profiles $x_1 \in \mathbb{R}^n$ and $x_2 \in \mathbb{R}^n$, the payoff profile should be $x_1 + x_2$ for the TU game $(N, v + \omega)$.

However, Shapley observed that this is not enough to guarantee the fairness. Practically, the formation process of the grand coalition may itself be unfair as the payoff of the players is dependent on the joining order. For example, if the set of players is $N = \{A, B, C\}$, then there are six possibilities on how the grand coalition may form, namely: *A-B-C*, *A-C-B*, *B-A-C*, *B-C-A*, *C-A-B*, or *C-B-A*. Therefore, Shapley suggested to allocate each player the average contribution made by him/her to the set of players that preceded him/her, over each possible ordering of these players, i.e., the Shapley value of player i in the TU game (N, v) is given by Equation (2).

$$\phi_i(N, v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \times (v(S \cup \{i\}) - v(S)). \tag{2}$$

Let's go back to Example 1 and see what payoff should be assigned to SAP according to Shapley value in order to ensure the fairness in the grand coalition. The worth produced by the different possible coalitions in Example 1 is given by Equation (3).

$$v(S) = \begin{cases} 0.5, & \text{if S=\{SAP\}} \\ 0.6, & \text{if S=\{IBM\}} \\ 0.5, & \text{if S=\{Ericsson\}} \\ 3.1, & \text{if S=\{SAP, IBM\}} \\ 3, & \text{if S=\{SAP, Ericsson\}} \\ 3.3, & \text{if S=\{IBM, Ericsson\}} \\ 4, & \text{if S=\{SAP, IBM, Ericsson\}} \end{cases} \tag{3}$$

The first step in computing the Shapley value for SAP is to calculate its marginal contribution over all the possible orderings of the grand coalition formation process. This is done as follows:

- If $S = \emptyset$, the marginal contribution of SAP is $v(\{SAP\}) - v(\emptyset) = 0.5 - 0 = 0.5$
- If $S = \{IBM\}$, the marginal contribution of SAP is $v(\{SAP, IBM\}) - v(\{IBM\}) = 3.1 - 0.6 = 2.5$

- If $S = \{Ericsson\}$, the marginal contribution of SAP is $v(\{SAP, Ericsson\}) - v(\{Ericsson\}) = 3 - 0.5 = 2.5$
- If $S = \{IBM, Ericsson\}$, the marginal contribution of SAP is $v(\{SAP, IBM, Ericsson\}) - v(\{IBM, Ericsson\}) = 4 - 3.3 = 0.7$

Given the marginal contribution of SAP over each possible ordering of the players in the grand coalition, its Shapley value is computed by averaging this contribution to the set of players that preceded it. Using Equation (2), the Shapley value of SAP is given by: $x(SAP) = \frac{2}{6} \times 0.5 + \frac{1}{6} \times 2.5 + \frac{1}{6} \times 2.5 + \frac{1}{6} \times 0.7 = 1.233$.

2.2.1.2 Non-Transferable Utility (NTU) games:

In non-transferable utility coalitional games [160], the choice of actions of each coalitional group of players specifies each player's payoff (not the payoff of the coalition as is the case in TU games), which is neither transferrable nor distributable. To clarify this definition and highlight the difference between TU and NTU games, we give the following example.

Example 2. *Four researchers, three professors and one Ph.D. student, in four different Canadian universities are willing to collaborate together and form a group (i.e., coalition) in order to produce and publish a research paper. The composition of the coalition in this case determines the quality of the paper that will be achieved. The payoff of the researchers represents a possible promotion or teaching load mitigation for the professors, and a further step toward obtaining the degree for the Ph.D. student, which are obviously non-transferable and non-dividable gains.*

The main type of NTU games is that of *hedonic games* [26], which has been widely used in numerous application domains. In this type of games, agents have preferences over coalitions. As its name hints, each agent is aware whether he/she will be "happier" if he/she joins the members of coalition x or the members of coalition y. The hedonic aspect stems from the fact that the preference assigned by a certain agent to a particular coalition depends only on the members of that coalition, irrespective of the existence/inexistence and structure of other coalitions.

Definition 4. *A Hedonic game is a pair $(N, (\geq_i)_{i \in N})$ where:*

- *N: is the set of players*
- *$\geq_i \subseteq 2_i^N \times 2_i^N$: is a complete, reflexive and transitive preference relation for player i, where $C_1 \geq_i C_2$ means that player i prefers to be in coalition C_2 at most as much as coalition C_1, and $C_1 \sim_i C_2$ means that player i is indifferent between coalitions C_1 and C_2.*

Now, let's modify the settings of Example 2 slightly to reflect the hedonic aspect of the game.

Example 3. *Three researchers are asked (e.g., by some granting agencies) to produce and publish two research papers within* 3 *months. To accelerate the process and meet the deadline constraint, these researchers decide to collaborate with one another as coalitions. The first step in achieving the hedonism is to assign names and affiliations for these researchers:*

- *Professor Alice from Concordia University;*
- *Professor Alex from University of Toronto; and*
- *Professor Guillaume from Université de Montréal*

These hedonic researchers have the following preferences over the possible coalitions:

- *All researchers favor coalitions of size two to coalitions of size one (less productivity) or three (a lot of names to add on the paper!).*
- *All agents are indifferent between coalitions of size one and coalitions of size three.*
- *Alice prefers to be with Alex than to be with Guillaume (since she considers that University of Toronto is more prestigious than Université de Montréal).*
- *Alex prefers to be with Guillaume than to be with Alice (since Guillaume has a higher record of publications).*
- *Guillaume prefers to be with Alice than to be with Alex (since both Guillaume and Alice are in Montréal and they can meet and exchange expertise either between each other or between their students, if necessary).*

Thus, the NTU hedonic coalitional game of this example is the pair $(N, \geq_i)$*, where:*

- $N = \{Alice, Alex, Guillaume\}$
- $\forall S \subseteq N, |S| = 2 \ >_N \ |S| = 1$ *and* $|S| = 2 \ >_N \ |S| = 3$
- $\forall S \subseteq N, |S| = 1 \ \sim_N \ |S| = 3$
- $\forall S \subseteq N, S \setminus \{Alice\} \cup \{Alex\} \ >_{Alice} \ S \setminus \{Alice\} \cup \{Guillaume\}$
- $\forall S \subseteq N, S \setminus \{Alex\} \cup \{Guillaume\} \ >_{Alex} \ S \setminus \{Alex\} \cup \{Alice\}$
- $\forall S \subseteq N, S \setminus \{Guillaume\} \cup \{Alice\} \ >_{Guillaume} \ S \setminus \{Guillaume\} \cup \{Alex\}$

Unlike cohesive TU games whose main goal is to form the single grand coalition, hedonic games are more interested in generating a coalition structure that partitions the players into disjoint coalitions [37]. Similar to TU games, hedonic games aim at guaranteeing the stability of the formed coalitions. For this purpose, the concept of core used in TU games can be extended to hedonic games in addition to other stability solution concepts [11]. In the following, we present the main stability concepts that are commonly used in hedonic games [26]:

- **Perfection**: A partition is called perfect if it is the most desirable for each player.
- **Pareto Optimality**: A partition is said to be Pareto optimal if there does not exist another partition that is strictly better for at least one player without being strictly worse for some others.
- **Nash Stability**: A partition is Nash-stable if none of its players has incentive to move from his/her current coalition to another (possibly empty) coalition.
- **Core Stability**: A coalition $S \subseteq N$ is said to *block* a coalition structure if every player prefers to be in S at least as much as to be in his/her current coalition. A partition that admits no blocking coalition is deemed to fall in the core (i.e., core stable).
- **Individual Stability**: A partition is individually stable if no player is better off moving from his/her current coalition to another existing coalition x without making the members of x worse off.
- **Individual Rationality**: A partition is individually rational if none of its players is better off acting alone.
- **Contractual Individual Stability**: A partition is contractually individually stable if there does not exist a player who may benefit by moving from his/her coalition to another existing coalition while making no member of either coalitions unhappy.

These stability concepts are characterized by inclusion relationships among each other in the sense that satisfying one of these stability concepts may directly imply the satisfaction of one or more other concepts [19]. In particular:

- *Perfection* $\subset$ *Core Stability* $\subset$ *Individual Rationality*.
- *Perfection* $\subset$ *Nash Stability* $\subset$ *Individual Stability*.
- *Perfection* $\subset$ *Pareto Optimality* $\subset$ *Contractual Individual Stability*.
- *Individual Stability* $\subset$ *Individual Rationality*.

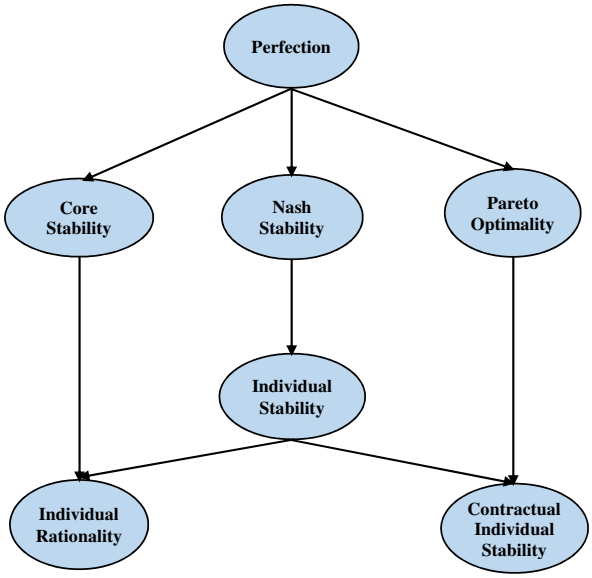


Figure 2.3: Inclusion relationships among stability concepts in hedonic games

- *Individual Stability* $\subset$ *Contractual Individual Stability*.

These inclusion relationships are summarized in Figure 2.3.

2.2.2 Non-Cooperative Game Theory

Non-cooperative games model situations in which each player acts individually and tries to maximize its own utility by choosing the best response(s) to the other players' strategies. In order to better explain non-cooperative games and clarify why they are important, we give in the following an illustrative example based on the well-known problem of *Prisoners' Dilemma* [64]. The presented game falls under the umbrella of non-cooperative games in strategic form (Fig. 2.2).

Example 4. *The Prisoner's Dilemma describes a crime investigation scenario wherein two suspected persons, say John and Bob, are arrested by the police for having probably committed a crime. The story starts when the policeman isolates these suspects and suggests each one of them the same following deal:*

1. *If you both confess, both of you will spend* 3 *years in jail (payoff:* -3*).*
2. *If one of you confesses against the other, the confessor will be jailed for* 1 *year (payoff:* -1*) and the offender will spend* 10 *years in prison (payoff:* -10*).*
3. *If both of you decline to confess, you will both be jailed for* 2 *years (payoff:* -2*) instead of* 3 *(lack of evidence).*

The players of this game are the two suspected persons. The actions available for them to choose from are either to confess or not. The payoff of each player represents the years of jailing this player will undergo, where the larger negative value represents more years to spend in jail. The objective of the game is to help each suspect make the ***optimal decision*** *on whether to confess or not in such a way to* ***minimize his jailing years (i.e., maximize his payoff)****,* ***while taking into account the actions that may be chosen by the other suspect (i.e., confess or not) and that influence his own welfare (i.e., jailing years)****. The bimatrix presented in Table 2.2 summarizes the foregoing deal, where the rows correspond to the actions available for John, the columns correspond to the actions available for Bob, and the cells correspond to the utility/payoff values for each player (the leftmost value in each cell refers to John's utility and the rightmost value refers to Bob's utility).*

Bob / **John**	Confess	Don't Confess
Confess	(-3, -3)	(-1, -10)
Don't Confess	(-10, -1)	(-2, -2)

Table 2.2: Payoff matrix of the players in the Prisoner's Dilemma game

The question that arises in each suspect's mind is: how should I behave in such a scenario? Each suspect will have the following rational thinking that is inspired by the well-known concept of Nash equilibrium [97]:

- *If the other guy confesses, I have to confess and get jailed for* 3 *years instead of being jailed for* 10 *years if I do not.*
- *If the other guy refuses to confess, I have also to confess since in that case I will get jailed for* 1 *year instead of being jailed for* 2 *years if I do not.*

Therefore, the solution of this game based on the Nash equilibrium concept is to confess for both players and spend 3 *years in prison. Intuitively, this can justified by the fact that each player is escaping from the worst-case payoff (*10 *years in jail) he may get in case he refrains from confessing.*

2.3 Literature Review and Discussions

In this section, we conduct detailed literature reviews on the different aspects addressed in this thesis. We start with the architectural and business perspectives and give a

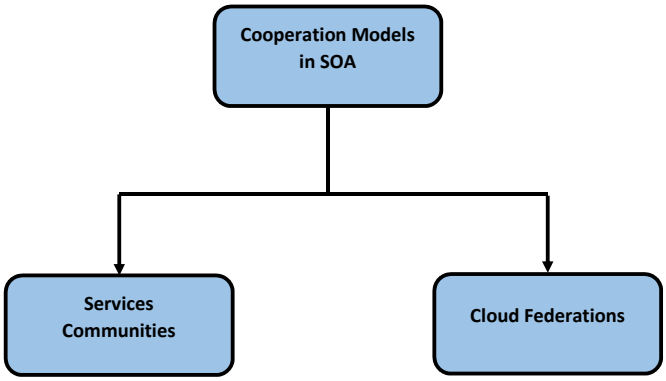


Figure 2.4: Classification scheme of the cooperation models in the SOA

review on the main cooperation approaches proposed for SOA applications, namely those of services communities and cloud federations. We offer as well an overview of Stackelberg game theory and its main applications, which is used in the formulation of our business and security-oriented solutions. Finally, we move to the security perspective and discuss the main intrusion detection approaches proposed for cloud-based applications.

2.3.1 Cooperation Models in SOA

As depicted in Fig. 2.4, the cooperation models proposed for SOA applications can be classified into two major categories: services' communities and cloud federations. In the following, we provide an overview of the main contributions related to these two concepts.

2.3.1.1 Services Communities

In [20], the authors proposed the concept of service containers as community gathering substitutable Web services that share a common functionality. These containers are no more than services created, advertised, discovered, and invoked just as elementary and composite Web services are. The purpose is to facilitate the composition process when the set of services is large and dynamic. This is done by allowing the invocation of the service container operations instead of invoking elementary or composite service operations. Maamar et al. [82] defined a similar concept for the communities of Web services. They considered the community as a group of services sharing the same functionality but differing in their QoS parameters. This approach differentiates between the *master* Web service that is charged with managing the operations of the community and the *slave* Web services that are simply all the other community members.

In [21], Benslimane et al. proposed a multi-layer approach for Web services composition made up of three constituents: *component*, *community*, and *composite*. The component layer comprises the Web services themselves, the composite layer illuminates the requirement of composing several services, while the community layer resides between the component and composite layers and has the role of organizing the Web services having common functionalities. The community is composed of abstract Web services describing the common functionality of the community and concrete services implementing this functionality. The composite layer is fed by the community layer with the needed components.

In [91], the authors proposed a framework for ontological organization of Web services using communities. The community serves as a cluster grouping the Web services based on their domain of interest. The community is created by the service providers who identify the community of interest and register their services in it. Communities provide a set of generic functions that may be customized by the underlying services. In a close work, Zeng et al. [166] addressed the problem of enhancing the runtime of the process of selecting the Web services to participate in large compositions. As the number of services offering the same functionality tends to be very high, they suggested grouping these services into communities that provide descriptors to a certain functionality.

All of the aforementioned approaches focus either on user's satisfaction or on performance optimization; thus ignoring the intelligent services' satisfaction. This would discourage services from participating in the community formation process due to the lack of any incentive for doing so. More recently, several approaches have been proposed to study and analyze the objectives of the Web services as rational agents in the process of creating communities. In [70], the authors proposed a 3-way satisfaction approach to help the master of the community select the services that will participate in fulfilling users' requests in an efficient manner. They considered the satisfaction of the three parties involved in this scenario; namely, Web services, users, and master. To this end, they formulated a satisfaction function for each party. For the user, the satisfaction is related mainly to the QoS provided to his/her requests. For services, the satisfaction is expressed in terms of participation level in the community's activity. The satisfaction of the master is expressed in terms of the revenue it earns. Then, a score function is formulated as a weighted sum of those satisfaction functions. The master uses this function to compare one selection against another during the selection process.

Liu et al. [73] have introduced a coalitional game theoretical model to create a cooperative scheme among autonomous Web services in a community-based context. Their approach allows Web services to reach an individually stable coalition partition wherein each Web service can maximize its utility through cooperation without reducing other Web services' utilities. In a close work, a coalitional game model has been introduced in [63]

with the aim of finding efficient ways for forming services' coalitions within communities. The key idea is to guarantee the fairness while distributing the gain among coalition members in order to maintain the stability of the coalitions. A coalition is deemed to be stable if no subset of its Web services can find significant gain by deviating from that coalition. Upon receiving a new membership request, the community coordinator first verifies whether the new coalition, taking into consideration that new member, will remain stable. If so, the request is accepted; otherwise, the coordinator rejects the membership request.

Concluding Remarks Overall, the problem of the existing community formation models is their ineffectiveness in a multi-cloud environment. On the one hand, they rely on a centralized architecture in which a central entity (i.e., the master) coordinates the operations of the community, which contradicts with the distributed nature of cloud-based services. On the other hand, they overlook both the business potential of the services involved in the community formation process and the problem of malicious services whose presence is likely in the multi-cloud environment.

2.3.1.2 Cloud Federations

In [118], Rochwerger et al. paved the way for the notion of cloud federations by introducing the concept of RESERVOIR whose main goal is to explore the technologies needed to handle the scalability problem faced by the single provider model. They discussed the notion of cloud federations wherein providers characterized by a large capacity of computing infrastructure may lease some of their resources to other providers who lack temporarily for such resources. Goiri et al. [44] addressed the problem of cloud federations from the perspective of increasing providers' profits. They proposed several equations to help providers decide when to outsource resources to other providers, when to insource free resources to other providers, and when to shutdown unused nodes to save power. In [135], Van den Bossche et al. formulated a Linear Programming model to assist providers with deciding which workloads to outsource and to which providers in such a way to maximize the utilization of internal data centers, minimize the cost of running outsourced tasks, and keep up high QoS constraints. Recently, game theory has been widely used to address the problem of forming cloud federations. In [102], Niyato et al. proposed a coalitional game among cloud providers. First, a stochastic linear programming game model that takes into account the random internal demand of cloud providers is formulated to analyze the resource and revenue sharing for a certain group of providers. Thereafter, a coalitional game that enables cloud providers to form cooperative groups for resource and revenue sharing is presented. The objective is to exploit the under-utilized resources when the internal demand in cloud data centers is less than the capacity of the providers. In [88],

Mashayekhy et al. investigated a hedonic coalitional game that focuses on the cooperation among IaaS services to improve resource scaling capabilities. The resources considered are provisioned as VM instances of different types (e.g., small, large, etc.). The objective is to form the federations that yield the highest profit in terms of cost and price of the underlying VMs. In [51], Hassan et al. proposed a coalitional game that takes into account the QoS of providers during coalitions formation. They assume the existence of a central entity that regulates the formation process by monitoring providers' QoS values.

Concluding Remarks In summary, cloud federations focus exclusively on improving the resource scaling capabilities among IaaS providers. On the other hand, the community is a more general architecture that supports, in addition to resource scaling, discovery, marketing and services composition facilitation. Thus, a cloud federation might be considered to be a subset of a services' community. Besides, the existing cloud federations formation models overlook both the business potential of the cloud providers involved in the federation process and the problem of having malicious services/providers, which constitutes a serious challenge for the success and applicability of such an idea.

2.3.2 Stackelberg Games

Stackelberg game theory is an extensive-form non-cooperative game of perfect information that has been widely used to model the situations that are characterized by a hierarchical structure. It was developed in 1934 by Heinrich von Stackelberg in his book "Market Structure and Equilibrium" [138][3] to model the imperfect competition in the market study. It represented a turning point in the study of market structure, particularly the analysis of duopolies (i.e., the cases when two firms have full control over the market). The basic idea of Stackelberg is that one player (leader), thanks to its historical precedence, size, reputation, innovation, information, and so forth, has the right to make the first move. Then, the other player (denoted as follower), that is less strong, observes the leader's strategy and decides about its own accordingly. Stackelberg games enjoy several characteristics that make them appealing to model the situations that are characterized by a hierarchical structure such as: (1) it is a sequential game (non-simultaneous), where one player plays first and then plays the other one; (2) the leader knows ex ante that the follower observes his action; (3) the leader's action is irreversible; and (4) it is characterized by the *first mover's advantage* property, which states that the player who plays first yields a payoff that is higher than that of the second player.

[3] This book has been translated to English in [139] by Damien Bazin et al.

In the following, we highlight some application domains wherein the Stackelberg game has shown to be successful. In [167], the authors used a Stackelberg game to model the spectrum allocation problem in Cognitive Radio Networks. The idea is to allow primary (licensed) network users to use secondary (unlicensed) users as cooperative relays to improve the performance of primary transmission. This situation has been modeled as a two-stage Stackelberg game where primary users, denoted as leaders, decide about the slot of bandwidth to be used for direct transmission as well as the set of secondary users that they are willing to cooperate with. The secondary users, acting as followers, decide about the payment to make under the pre-decided bandwidth slot with the target of maximizing the transmission rate without having to make large payments. A Stackelberg game approach has been used also to model the problem of efficient bandwidth allocation in cloud-based wireless networks [96], where desktop users watching the same live program may be willing to share their live-streaming with the nearby mobile users. This situation is modeled as a Stackelberg game that involves two-stages: (1) a non-cooperative game among desktop users to decide about the bandwidth size to be shared with the mobile users along with the price of sharing, and (2) an evolutionary game among mobile users to decide about the desktop users to connect with under the offered size and price.

Stackelberg games have been widely used in the security domain to characterize the attacker-defender models [115, 94, 132, 31]. In [115], the authors used a Bayesian Stackelberg game to help Los Angeles International Airport (LAX) police officers protect the airport against adversaries. The challenges that led to the use of a Bayesian Stackelberg game are related to (1) the impossibility to provide full security coverage at all times; (2) the ability of the adversaries to observe the security arrangements over time and adjust their malicious strategies accordingly, and (3) the uncertainty the police agents face over the type of adversaries (e.g., thieves, terrorists). In the proposed game, police officers play the role of the leader and adversaries are the followers. The goal is to find the optimal strategy the police should commit to, given that followers may be aware of the leader's strategy when choosing their own strategies and that the police is not able to learn the follower's type a priori. In [94], a Stackelberg game has been used to model the adversial learning in which the adversary tries to manipulate the data miner's data to reduce the accuracy of the classifier. A Stackelberg game is employed, where the adversary is the leader, the data miner is the follower, and the objective is to help the data miner decide whether to retrain the classifier or maintain the Status Quo given the action of the adversary.

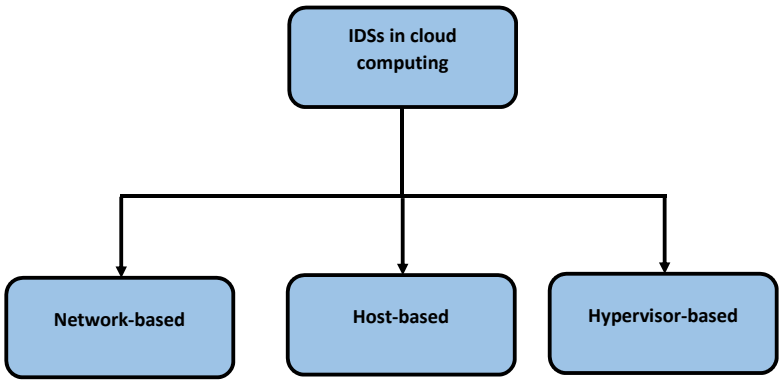


Figure 2.5: Classification scheme of the intrusion detection systems in cloud computing

2.3.3 Intrusion Detection Systems in Cloud Computing

As depicted in Fig. 2.5, the current Intrusion Detection Systems (IDSs) proposed for cloud-based applications can be classified into three main branches: host-based, network-based, and hypervisor-based systems. Network-based systems put the monitoring agents at the network's level to monitor the circulating traffic and recognize any malicious behavior. The fact that these systems operate at the network's layer only makes them unable to catch insider attacks that sneak into the internal virtualized system. To remedy this shortcoming, host-based IDSs deploy the monitoring agents at the VMs' layer to monitor their activities and report any abnormal behavior. The main limitation of this approach lies in the burdens it puts on the users who are required to spend their own resources and efforts to maintain the health of the monitoring agents. To alleviate these burdens, hypervisor-based systems place the monitoring agents at the cloud system's layer and assign to the host hypervisors the role of observing the VMs' system metrics and identifying malicious activities. In the following, we explain the main contributions in each of these branches and highlight the main advantages and limitations of each branch of IDSs.

2.3.3.1 Network-based Detection Systems

In [66], the authors discuss an intrusion detection framework that monitors network traffic using a cluster-based architecture to support multiple security domains. The basic idea is to export the intra-VM network traffic to be processed by a physical IDS. Moreover, a traffic deduplication technique is advanced to remove redundant network traffic and minimize the overhead.

In [15], a customer-controllable on-demand IDS is introduced. The network interactions among VMs within a pre-defined virtual network are monitored and suspicious activities are

registered and analyzed. The performance of the framework is adaptable based on the volume of traffic load in the network, where, for example, the number of IDS components can be adjusted on the basis of the amount of traffic circulating inside the network.

In [75], the authors propose a cooperative IDS for Denial of Service (DoS) attacks. They assume that an IDS is deployed in each cloud computing region to collect network packets and analyze them. If the type of the analyzed packet matches any type defined in the block table (storing the bad packet to be blocked), then this packet is immediately dropped. If no match exists but the packet is categorized as anomalous, then the degree of severity of that suspicious packet is checked. If the packet is classified as serious, then the IDS drops it and notifies the other IDSs accordingly. If the packet is classified as moderate, the IDS performs data clustering and threshold check to find outliers and updates the alert level accordingly. Finally, if the packet is identified as slight, then the system ignores the alert.

In [71], the authors address the DoS attacks in cloud environments by proposing a scheme for tracing back the botmaster (i.e., the malicious user that administrates the botnet). In the proposed scheme, the local network administrator of the victim machine collects information, files them to a traceback server, and asks the latter for a traceback service. The traceback server then embeds Pebbleware, a piece of code that reveals its host machine's information, on the communication packets from the victim node to the botmaster. Once the Prebbleware reaches the botmaster, the latter's machine is obliged to send its IP address to the traceback server.

In [83], the authors investigated the problem of collaboration among providers in a federated cloud to provide a holistic security defense mechanism. The idea is that providers in a certain federation exchange with each other the attack features logged in their clouds, which allows for continuous update of the vulnerability database and incremental improvement in the security defense system. In order to detect attacks, a virtual router is deployed on the VMs in each cloud provider's domain to monitor the incoming and ongoing traffic.

In summary, the basic idea of network-based IDSs is to monitor the incoming and outgoing network traffic in order to detect intrusions. The main limitation of these approaches lies in their ineffectiveness in capturing the internal attacks wherein attackers penetrate into the internal cloud system.

2.3.3.2 Host-based Detection Systems

In [77], the authors propose a distributed detection mechanism for detecting Distributed Denial of Service (DDoS) attacks in cloud environments. The basic idea is to deploy and configure an IDS within each VM whose responsibility is to collect alerts. The alerts from

the different VMs are then sent and stored into a Mysql database hosted in the front-end server. Thereafter, the alerts are converted into basic probabilities assignments and analyzed using the Dempster-Shafer's combination rule [151].

In [156], the authors propose Varanus, a multi-tier detection model for large-scale IaaS cloud services. In Varanus, VMs are partitioned into a set of groups based on the similarity between their software configuration features (e.g., web servers, database servers, etc.) using the k-nearest neighbor clustering algorithm [49]. Each VM participates in a gossip-based monitoring scheme by propagating and receiving information (e.g., CPU usage, disk capacity, etc.) to/from the other VM agents in the same group. In each group, the under-utilized VMs are then nominated to perform the data analysis. Finally, the aggregate value for each group is communicated among the different groups of the same cloud domain.

In [34], the authors discuss a host-based intrusion detection technique that selectively monitors (only) the failed system call traces of the VMs. These traces are then analyzed and classified either normal or malicious using k-nearest neighbor. Finally, users are alerted of any malicious activity in their system.

Summarizing, the basic idea of host-based IDSs is to deploy a monitoring agent on each VM to monitor its states and behavior and identify any malicious behavior. Although these systems are more effective than network-based IDSs in identifying both internal and external attacks, such systems entail additional management responsibilities for the users and can be impeded by masterful attackers who violate the VM instances.

2.3.3.3 Hypervisor-based Detection Systems

In [158], the authors propose an online anomaly detection technique that operates at the hypervisor's layer. The system architecture consists of four main components, namely the Cloud Resilience Manager (CRM), System Resilience Engine (SRE), Network Analysis Engine (NAE), and System Analysis Engine (SAE). At the first stage, the CRM deployed on each cloud node collects features from the VMs and their local networks and sends this data to the NAE and SRE components. These two latter components employ the one-class SVM to carry out a local anomaly detection. Based on the output generated by the classifier, the SRE takes the responsibility of recovering actions.

The authors of [23] advanced *Collabra*, a distributed IDS that is integrated into Xen hypevisors to preserve the security of the cloud system. Collabra scans each hyper-call made by every application of the VMs to guarantee the integrity of the cloud infrastructure and ensure fail-safe transaction processes. Collabra performs in a collaborative fashion to enhance the results of the real-time detection.

In [76], the authors propose a virtualization-supported security architecture for cloud resources whose basic idea is to monitor the integrity of guest VMs and hosting infrastructure components, while being invisible to the end users. To detect attacks, system-call invocations performed by VMs are continuously monitored by an *Interceptor* entity that is located into the kernel space of the hosting platform. Suspicious activities are then logged by a *Warning Recorder* entity into the *Warning Pool* that prioritizes the order of evaluation of these activities. The *Warning Recorder* asynchronously computes checksums for critical host infrastructure and guest kernel code, data, and files. This information is passed then to the *Evaluator* entity to decide on whether the system's security has been breached or not.

In [101], the authors propose a hypervisor-based IDS for malicious activities on VMs. Every second, endpoint agents installed on hypervisors retrieve the performance metrics of the guest VMs such as CPU utilization, block device read/write data, and network data transmitted/received. This data is transmitted to the controller node, a service residing within the cloud environment, which is responsible for analyzing the received data against stored signatures and confirming the existence/inexistence of attacks.

In summary, hypervisor-based IDSs put the monitoring responsibilities on the hypervisor's site by allowing it to inspect and gather information from the guest VMs. The main limitation of this approach is that, in real systems, such a mechanism requires monitoring and analyzing a huge number of events from each single VM.

Concluding Remarks The existing IDSs stop at the borders of monitoring and analyzing events to identify intrusions. Thus, these systems consider the detection problem from the perspective of the IDS only without accounting for the strategies of the attacker who seeks to minimize the probability of detecting his/her attacks. This results in high false alarms and large levels of resources wastage in the detection process.

Chapter 3

Trust and Reputation Models in the Service-Oriented Architecture: Classification, Challenges, and Future Directions

Services selection constitutes nowadays a major challenge that is still attracting the research community to work on and investigate. The problem arises since decision makers (1) cannot blindly trust the service or its provider, and (2) ignore the environment within which the service is operating. The fact that no security mechanism is applicable in such a completely open environment, where identities can be easily generated and discarded makes social approaches such as trust and reputation models appealing to adopt and apply. In this chapter, we classify and compare the main findings that contributed in solving problems related to trust and reputation in the SOA. First, a high-level classification scheme partitions services into three main architectures: single, composite, and communities. Thereafter, a low-level classification within each architecture categorizes the trust and reputation models according to the technique used to compute the trust value. Based on this classification, a detailed analysis describing the advantages and shortcomings of each class of models is presented, leading to uncovering possible topics that need further study and investigation. In particular, we discuss the challenging problem of having malicious services in the composite and community-based architectures by means of deep analysis and simulation experiments. These findings can be used by future researchers as a roadmap to explore new trust and reputation models in the SOA, while taking into account the shortcomings of the existing models[1].

[1] The content of this chapter is published in [141]

3.1 Motivations of the Survey

Several reviews [155, 87, 114, 35] targeting trust and reputation in the SOA have been advanced. The motivation for this survey stems from two main reasons. First, the existing survey papers lack a comprehensive view of services' architectures. To the best of our knowledge, this is the first review work that classifies services according to their architecture and provides a collection of criteria that are important for the success of the trust and reputation models in each architecture. The second motivation is the lack of a profound and systematic review of trust and reputation in the SOA. This survey presents a high-level classification scheme for the services according to their architectures and a sub-classification in each architecture based on the underlying technique used to construct the trust/reputation value. Moreover, we define for each architecture a set of criteria that are necessary for the success and effectiveness of the trust and reputation models targeting this particular architecture. The classification scheme aims to help (1) providers improve the quality and performance of their services, (2) customers enhance the quality and credibility of their ratings, and (3) the research community to study and investigate some open challenges that are not yet solved in this domain.

3.2 Existing Surveys

Several surveys can be found in the literature about trust and reputation in the SOA [155, 87, 114, 35]. Wang et al. proposed in [155] a classification scheme for trust and reputation systems in Web services based on three criteria: (1) centralized or decentralized, i.e., there exists a central party charged of managing the reputation for all the members or not; (2) person or resource, i.e., they target persons or resources such as Amzon and eBay; and (3) global or personalized, i.e., collected based on opinions from general population that is visible to all members or based on opinions from a group of members.

In [87], the authors focused on the trust management models and issues related to semantic Web services. They classified the trust models based on the method used to compute the trust value, resulting in three categories: (1) Trust Computation Related to Services, where services establish trust for each other; (2) Trust Computation on Consumer View, where consumers provide feedback on the services based on their interactions; and (3) Trust Computation for Content and Context, which uses meta-data information to analyze the semantic data published on the Web.

In [114], the authors present a comparison summary between the reputation-based approaches proposed in the Service-Oriented Computing domain based on four criteria: maturity, majority, cost, and infrastructure. The maturity stresses the need for users' ratings

when building trust. Majority points out that a certain trust mechanism should be independent from the credibility of the majority of ratings that may be dishonest. Cost refers to the complexity and extensibility of the trust mechanism, while infrastructure refers to the ability to support distributed infrastructure such as Web services. In our work, maturity and majority are expressed in criteria $C2$ and $C7$ (Table 3.1) respectively.

Dragoni proposed in [35] a classification scheme for the trust-based services selection approaches based on their rationale; resulting in three classes: (1) Direct experience-based approaches in which consumers use the direct past experience with a certain service to build the trust for that service; (2) Trusted Third-Party (TTP) approaches in which consumers consult a trusted third party to build a trust for a certain service; and (3) Hybrid approaches that combine techniques from the two aforementioned classes to build integrated frameworks.

More generally than Web services, the topic of trust and reputation in online systems has been tackled in many review papers [56, 137, 74]. In [56], the authors presented a broad discussion about the notions of trust and reputation and proposed a classification for the trust and reputation models based on the reputation computation engines; resulting in six classes: Simple Summation or Average of Ratings, Bayesian Systems, Discrete Trust Models, Belief Models, Fuzzy Models, Flow Models. The focus of this survey is to discuss the trust and reputation in the deployed systems such as security and commerce rather than systematically reviewing the research literature.

In [137], the authors proposed a reference model for building reputation systems for e-services. They introduced a collection of criteria whose main objective is to ensure that the assessed reputation values reflect the actual trustworthiness of users. Several criteria may be inferred by combining criteria $C3$, $C4$, and $C7$ (Table 3.1) in our work. Examples of these criteria include: reputation should be assessed using a sufficient amount of information, and reputation system should be able to discriminate incorrect ratings.

In [74], the authors target the centralized online reputation systems by proposing a structure and providing a set of criteria for each component in this structure. Some of these criteria, related to the quality of the ratings, are reflected in $C3$, $C4$, and $C7$ (Table 3.1) in our work. Other criteria focus on the efficiency of the reputation systems as well as the aggregation algorithms by highlighting some relevant requirements such as complexity and robustness.

Similar to the aforementioned surveys, we do not claim to cover all the criteria needed for the trust and reputation models; however, our criteria are proposed to answer the research questions raised in Section 3.4. Numerous criteria proposed in other surveys, even not explicitly expressed in our work, can be inferred by combining some of our proposed criteria. Other criteria such as those related to the efficiency and complexity of the

reputation systems and aggregation algorithms are out of the scope of this study as the approaches selected for comparison do not consider these aspects. Overall, the unique features of our survey are (1) defining services' architectures and describing their points of convergence and difference; (2) providing a sub-classification within each architecture on the basis of the technique used to compute the trust/reputation value; (3) proposing a taxonomy of criteria for each architecture and comparing the class models and approaches in each architecture based on these criteria; and finally (4) discussing the limitations and future directions specific to each of these architectures.

3.3 Research methodology

The proposed criteria are selected to answer a collection of research questions we raise for each services' architecture. These questions target the major challenges that each architecture may face in the context of trust and reputation. The challenges have been identified as those ones that received most of the attention in the papers used for the survey. Many of these challenges are also explicitly identified as major issues in the surveyed papers. In the single architecture, most of the research is oriented to tackle the issues of bootstrapping, credibility of ratings, trust dynamism, and representativeness of the trust and reputation sources [90, 84, 85, 100, 126, 130]. For the composite architecture, the identified major challenges are determining the contribution of each component (i.e., service) in the composition process and the problem of task allocation among components [93, 50, 107, 163, 164] with the aim of determining the trust value of each single component. In the community-based architecture, joining communities and the influence of that joining on the performance and reputation of the community have been the key challenges [61, 62, 60, 22, 59]. Moreover, we have noticed that the topic of malicious attackers, that have major impacts on the trust and reputation of services and that has been a major challenge in many important domains such as networks [57, 162, 149], has been disregarded in the context of services. To this end, we raise this topic and highlight its importance by means of detailed analysis and discussion. In summary, the main challenges in the single architecture are related to the quality and credibility of the procedure used to build the trust/reputation values. For the composite architecture, the challenges are expanded to cover the issues of estimating the contribution and performance of the service constituents in the composition process as well as the task allocation problem among these constituents and the security concerns that may be engendered by the malicious constituents. As for the community-based architecture, the issues of making thoughtful joining strategies for the communities and protecting them against malicious attacks are additional concerns. Numerous criteria exist in other surveys, where each survey focuses

on certain aspect(s) related to trust and reputation. Similar to these surveys, we do not claim to cover all the criteria needed for the trust/reputation models; but our criteria are defined to answer the proposed research questions. It is worth mentioning as well that numerous criteria proposed in other surveys, even not explicitly expressed in our work, can be inferred by combining some of our proposed criteria. Other criteria such as those related to the efficiency and complexity of the reputation systems and aggregation algorithms are out of the scope of this study as the approaches selected for comparison do not consider these aspects.

The approaches chosen for comparison in each services' architecture are selected from papers published as of 2009 in refereed journals and international conferences. Moreover, we included papers that are major (based on the number of citations) in the domain of trust and reputation in the SOA published before 2009 and that are important to understand the core of the topic such as [90, 126]. The objective is not to gather and compare all the approaches that tackled trust and reputation in the SOA. The reason is that we advance a two-phase classification (based on the services' architecture and the technique used to compute the final trust value) of the current approaches and compare each class of models based on the proposed criteria, where approaches in the same class share the same basic idea but differ in some minor and technical details.

3.4 Problem Statement and Research Questions

In this section, we illustrate the need for a trust and reputation model in the scope of services by describing a real-life scenario and raise the research questions that our survey aims to address. Consider the case of a flight booking application. A customer makes a request containing the flight dates, origin and destination, type of tickets (one way or return), and number of guests to the *Flight Booking* service and asks for the information related to such a flight (i.e., companies, timing, prices). To gather such information, the *Flight Booking* service has to make a series of invocations. Practically, it would inquire the name of the companies, ticket prices, and timing on the specified route from the *Airline Reservation* service. Moreover, it will contact the *Hotel Reservation* service to get the prices and availabilities of hotel accommodations in the given destination. It will also invoke the *Car Rental* to get the options and prices of cars.

In this scenario, two types of interactions take place: (1) customer-to-service explicit interaction, and (2) service-to-service transparent interaction. The first type of interactions refers to the single architecture of services, while the second describes the composite architecture. Although the second type of interactions does not impact the customer directly,

it will affect the quality of the whole transaction. In fact, the overall quality of a composite service is affected by the quality perceived by each single service in this composition. Suppose that the *Hotel Reservation* service is overloaded by a huge number of requests. This would increase the response time of the overall transaction. Therefore, as much as the customer is interested in the selection of the appropriate service to obtain the "best" possible quality, the *Flight Booking* service is interested in selecting the appropriate services to be part of the composition in a way that allows it to maintain a good record among other *Flight Booking* services. Thus, both the customer and *Flight Booking* service have to make appropriate decisions in this context. Such a decision cannot be made randomly due to the fact that in such an open environment, anyone can offer services that may be of low quality, time consuming, expensive or even harmful. This raises the need for mechanisms that enable decision makers to distinguish good from bad services. Applying the usual security mechanisms such as authentication and access control cannot help us make optimal decision in such a case. In fact, learning the credentials of services is not enough to predict how well these services will perform, but having an idea about their past interactions would signal their trustworthiness. Without a reputation-based selection, it would be difficult for both customers and providers to select the appropriate services to deal with.

By using a reputation-based mechanism, the customer is increasing his/her chance to get higher QoS values. The provider, in his/her turn, is decreasing the risk of getting harmed because of non-reputable external components (i.e., services owned by other providers). While achieving these goals, several challenges arise. Some of these challenges are generic for all the services, while others are specific for each architecture. For the single services' architecture, the main challenges can be summarized by the following research questions:

- **Q1**: How and based on which parameters to evaluate the reputation of the services?
- **Q2**: How to assign initial trust values for the new services?
- **Q3**: How to adapt the trust values to the dynamic change in the services' performance?
- **Q4**: How to protect the trust/reputation values against collusion and deception problems?

These challenges apply as well in the composite architecture in addition to supplementary challenges imposed by the composite architecture such as:

- **Q5**: How to evaluate the performance of the individual constituents in the overall composite service?

- **Q6**: How to assess the trust of the constituents when their performance cannot be fully observable?
- **Q7**: How to manage the collaboration and task allocation issues among the constituents?
- **Q8**: How do malicious constituents affect the reputation and performance of the composite service?

In the community-based architecture, several services offering the same functionality are grouped into clusters to ease their discovery process and increase the overall performance. Thus, if a *Hotel Reservation* service, say H_1, does not have enough QoS requirements to fulfill the request coming from the *Flight Booking* service, it can cooperate with the other community members offering the same functionality (e.g., H_2 and H_3) or delegate the request for them to perform it with better performance. In such an architecture, dealing with trust and reputation becomes more challenging with more issues to consider, in addition to those of the single and composite architectures, arise such as:

- **Q9**: How to evaluate the reputation of a community in such a dynamic environment where services continuously join and leave?
- **Q10**: Would the community members cooperate with each other and why? How does this affect their reputations and the reputation of the whole community?
- **Q11**: How and based on which parameters should services be selected to be part of the community?
- **Q12**: How do malicious services influence the reputation and performance of the community?

Several approaches were proposed trying to answer some of these questions, while other questions still need further study and investigation. In what follows, we present and classify the main contributions to date that addressed issues related to trust and reputation in these three architectures, derive a collection of criteria for the trust and reputation models in each architecture from the aforementioned questions, compare the class models, and identify gaps from which researchers can find important topics to work on and explore.

3.5 Trust and Reputation in SOA

In this section, we present a high-level classification for the trust and reputation models according to the architecture of services they target and a low-level classification in

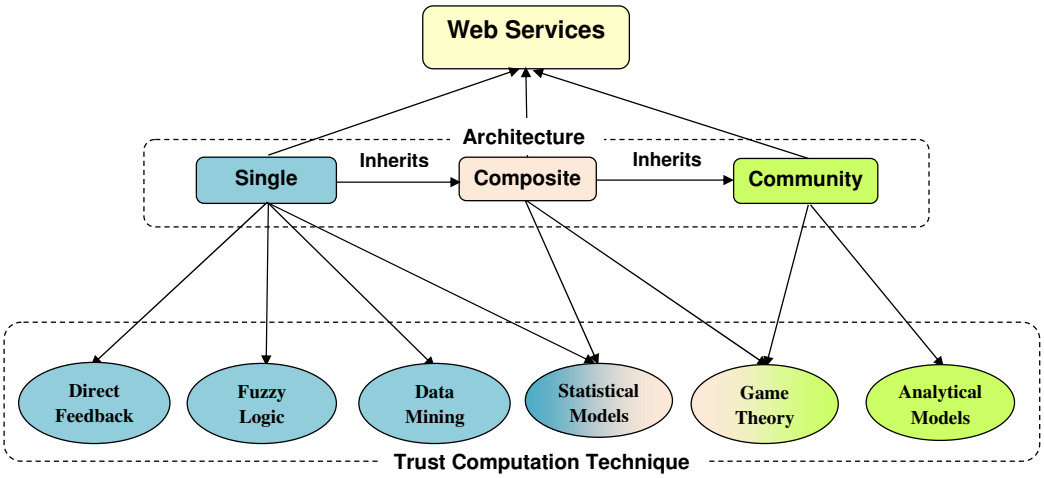


Figure 3.1: Classification scheme: Trust and reputation models are classified based on the architecture of services they target (high-level classification), and the technique they use to build the trust within each architecture (low-level classification)

each architecture based on the technique used to build the trust value. The classification scheme is depicted in Figure 3.1. We define as well a set of criteria for trust and reputation models in each architecture; based on this we conduct a high-level comparison among the classes of models and a low-level comparison among the major approaches in each architecture.

3.5.1 Single Services

Single services are referred to as those services working in a standalone manner to achieve users' requests (e.g., a weather forecasting service displaying the weather status in a certain city). Trust and reputation models proposed for this architecture aim mainly to help users select the appropriate service that best achieves their requests. The following criteria are important for the trust and reputation models while achieving this goal [100, 86, 164, 137, 114, 90, 74]:

- **Criterion** #1: Cover multiple QoS metrics (e.g., response time, throughput, availability, etc.) to enable users to distinguish well among functionally-similar services.
- **Criterion** #2: Consider the user preferences since users may be interested in different quality metrics (i.e., one user may be interested in the response time while another user may look for lower cost).
- **Criterion** #3: Account for both subjective (feedback from users) and objective (QoS monitoring) perspectives while evaluating the trust and reputation of services.

Table 3.1: Criteria for the trust and reputation models in the single services' architecture

ID	Criterion	References
C1	Cover multiple Quality of Service (QoS) metrics such as response time, throughput, and availability.	[90, 100]
C2	Consider the user preferences.	[114, 100]
C3	Account for both subjective and objective perspectives.	[100, 137, 74]
C4	Assess the credibility of raters.	[100, 137, 74]
C5	Have a bootstrapping mechanism.	[86, 164]
C6	Consider the trust dynamism.	[90, 164]
C7	Be independent from the credibility of the majority of ratings.	[114, 137, 74]

- **Criterion** #4: Assess the credibility of raters (giving feedback on the behavior of the services based on direct interactions) to avoid collusion and deception.
- **Criterion** #5: Have a bootstrapping mechanism to assign initial trust values for the newcomer services (i.e., newly deployed services).
- **Criterion** #6: Consider the trust dynamism issue since the performance of services is subject to change over the time (ameliorate or deteriorate).
- **Criterion** #7: Avoid the dependency between the recommendation given to a certain service and the credibility of the majority of ratings.

These criteria are summarized in Table 3.1. Most of the trust and reputation models proposed for the single architecture of services use direct feedback collected from users to compute the trust value for the services. Few statistics-based, fuzzy-logic-based, and data-mining-based models were proposed for this purpose. More details on these models and their associated approaches are given in what follows. Thereafter, Table 3.2 compares the discussed approaches according to the criteria presented in Table 3.1.

3.5.1.1 Feedback-based models

Feedback-based models [90, 84, 10, 89] rely on the idea of collecting reviews concerning a certain service. These reviews are used then to build a trust value for the service in question. The source of reviews is either the provider or the consumer [67]. Provider-generated information includes the descriptions of the service recorded in the service registry. Consumer-generated information are, on the other hand, online reviews provided by the users who had dealt with the service during past interactions. For example, Maximilien and Singh [90] proposed a multi-agent framework based on an ontology of QoS metrics allowing providers to proclaim their services, users to state their preferences, and ratings about services to be built and shared. The ratings are based on the QoS metrics, which include well-known computing parameters such as latency and throughout but may involve

also application-specific parameters such as shipping delay. The proposed framework relies on three main concepts: provider quality advertisement, customer quality preference, and service reputation. Using the provider quality advertisement, providers advertise their services by specifying the minimum and maximum possible quality values for the offered service as well as the promised value for this service. Consumers, in their turn, describe their preferences by specifying the minimum and maximum acceptable quality thresholds as well as the preferred quality value. Thereafter, a trust function is formulated based on the reputation function, the consumer's preferences, and the provider's advertisements. The aim of this function is to rank the services based on how well they satisfy users' requirements in order to help make selections. The framework also provides a mechanism to periodically monitor the services in order to allow users to replace the poorly-performing services by other well-performing ones.

Although feedback-based models have the advantage of considering the opinions of the users, which tends to be the most rational and meaningful metric for building the reputation of any service, these models suffer from major problems. First, feedback-based models provide no bootstrapping mechanism for computing initial trust for services. Second, the quality and credibility of the ratings is a main problem that encounters feedback-based models. More precisely, providers tend to hide the bad characteristics of their services and stress the good aspects for marketing and commercial purposes. On the other hand, the feedback provided by the consumers tend to be more realistic due to two main reasons. Firstly, the feedback presented by the consumers are usually user-oriented in the sense that they focus on the aspects that concern the user such as QoS and cost in contrast to the providers that tend to proclaim the service-oriented information. The second reason is that consumers have higher probability than providers of mentioning the weaknesses along with the strengths of the services as they are assumed to be neutral parties who have no direct interest in the promotion/demotion of certain services. However, this does not mean that the reviews presented by the consumers are always truthful. In fact, consumer-based reviews are usually not organized in a standard manner in the sense that each user has his own style in writing the reviews that is different from other users (e.g.,$\{0, 1, 2\}$ vs. $\{$excellent, good, bad$\}$). Besides, users usually tend to refrain from submitting reviews as they have no incentives for doing so, which leads to biased computation of the aggregated trust value. Most importantly, consumers are rational agents who may be tempted to provide dishonest feedback resulting in benefit for them as a result of a certain collusion scenario. For example, some consumers may collude with the providers to submit positive feedback on their services and/or negative feedback on the services of their competitors versus obtaining reduced service fees.

This problem was tackled by several approaches [10, 84, 120], where the authors consider the existence of malicious raters that may provide untrustworthy ratings. The main

limitation of these approaches is that they are based on the idea that the majority of raters are credible in the sense that the rating of a certain consumer is assumed trusted if it agrees with the majority of ratings and untrusted otherwise. In this way, malicious raters can still impose their opinions and get high reputations by merely submitting a large number of fake feedback in such a way that allows them to form the majority.

3.5.1.2 Statistics-based models

In general, statistical models [157] are used to describe the relationship among a set of variables by means of mathematical equations. In the context of single services, a few statistical models [100, 18] are used to compute trust values for the services. These models attempt to overcome the problems of the feedback-based models, which rely solely on the reviews provided by providers and/or users and that may be incredible, by considering multiple sources of trust and using statistical methods to combine them.

For example, Nguyen et al. [100] proposed a trust model based on a Bayesian Network (BN) that integrates both subjective and objective trust sources such as: direct opinion (ratings from users), recommendation (combination of public and personalized metrics), and conformance (between promised and actual QoS values). Based on these sources, the final trust value is calculated as the weighted sum of the three metrics.

In RATEWeb [85], the authors proposed a set of metrics inspired by the social networks methodologies with the aim of enhancing the accuracy of ratings and dynamically assessing the changing conditions. These metrics involve the credibility of the raters (to target malicious ratings), personalized preferences (weighted preferences over the QoS metrics), temporal sensitivity (to assign more weight to the most recent ratings), and first-hand knowledge (to cope with services' performance dynamism). Finally, a statistical technique is used to combine these metrics and compute the trust value.

Although statistics-based models provide powerful mechanisms for building the trust value by collecting and combining multiple sources of trust, these models still cannot compute initial trust values for the newcomer services as they provide no bootstrapping mechanism that tackles this problem.

3.5.1.3 Fuzzy-logic-based models

Fuzzy logic [27] is a reasoning approach that supports approximate rather than exact values. In the context of trust and reputation in SOA, fuzzy models [126, 99] are used to analyze the semantics and rationale behind the feedback left by the users. The motivations behind this are to (1) facilitate the construction of recommendations by aggregating the

feedback left by users having the same preferences together, i.e., a user interested in the response time will be more interested in knowing the feedback related to the response time than those related to the price; (2) detect the bogus ratings provided by malicious users, e.g., users who are always submitting positive feedbacks on a certain service although the performance of this service was bad in multiple invocations. In [126], the authors proposed a fuzzy-logic-based reasoning model that combines both the subjective perspective represented by users' ratings and the objective perspective referred to as the compliance between promised and actual performance. To this end, they propose to assign a rating for each service based on its compliance value and compare it with users' ratings. This rating is computed in way that makes it biased towards a certain parameter (e.g., response time). Thereafter, the rating given by the user is compared against all the estimated ratings and the rating that best matches the user's rating is deemed to be equivalent to the user's rating.

Although fuzzy-logic-based models try to understand the semantic behind the ratings provided by the users, which constitutes an important topic in the context of trust and reputation, these models offer only a set of rules and comparisons as ultimate output (e.g., if the compliance of response time and availability is poor but the compliance of performance is good, then the rating is considered to be poor) but provide no mechanism for computing the final trust value and are not able hence to help users and/or services make selections. They cannot compute initial trust values for the new services as well. Moreover, they do not take into account the dynamism of the trust.

3.5.1.4 Data-mining-based models

Data mining is an interdisciplinary subject that describes the process of extracting hidden patterns from huge datasets [54]. Data mining is becoming increasingly adopted in many domains such as medicine, engineering, science, business, etc. Despite its importance, this emergent discipline has not been well-exploited to address the problems related to trust and reputation for services. A data-mining-based approach was presented in [130], which uses the text mining to analyze the reviews provided by the users in order to evaluate the services and facilitate thus their selection. However, this approach is based on the simplistic assumption that the reviews presented by the users are always credible. Moreover, the authors didn't provide an in-depth methodology of how the text mining will be effectively performed. Additionally, the bootstrapping and trust dynamism issues are ignored in this approach. Further steps are required to take advantage of the promising techniques offered by data mining (e.g., clustering, classification, frequent patterns, association rule, etc) [147], that seem to be useful to solve problems related to trust and reputation.

Table 3.2: Comparison summary between the main trust and reputation approaches in the single architecture

Approach	Model	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**
Maximilien and Singh [90]	Feedback-based	✓	✓	✓			✓	✓
Malik and Bouguettaya [84]	Feedback-based	✓	✓		✓		✓	
Malik and Bouguettaya [85]	Statistical	✓	✓		✓		✓	
Nguyen et al. [100]	Statistical	✓	✓	✓	✓			✓
Sherchan et al. [126]	Fuzzy logic	✓	✓	✓	✓			✓
Thurow et al. [130]	Data mining	✓	✓					✓

3.5.2 Composite Services

Services' composition involves integrating and organizing a set of services to achieve certain complex functional and/or non-functional requirements that cannot be accomplished by a single service [109]. In this architecture, trust and reputation models aim to help composition designers select the appropriate services to be part of the composition process resulting in benefit for both designers (better reputation) and users (better quality). To achieve this goal, several criteria have to be taken into consideration. As composite services are no more than a set of single services working together to achieve a certain objective, the requirements proposed for the single architecture (Table 3.1) apply as well for the composite architecture in addition to other important requirements such as [50, 93, 168, 127, 164, 163, 57, 162]:

- **Criterion** #8: Capture the responsibility of each constituent in the overall quality of the composite service in order to improve current compositions and facilitate future selections.
- **Criterion** #9: Consider that the responsibility of each constituent cannot be fully observed since users usually deal with the composite service as a monolithic entity.
- **Criterion** #10: Take into account the dynamism in the behavior of the constituents even when this change does not affect the overall quality of the composite service. For instance, consider the case of a service composed of two constituents X and Y. Initially, X is good and Y is bad. If X changes to bad and Y changes to good, the model should be able to capture this change although the overall performance of the composite service is not affected.
- **Criterion** #11: Monitor the variations in the QoS parameters of the constituents since predicting the performance based on the previous behavior cannot always yield reliable results as the performance can change in irregular manner (e.g., on demand).
- **Criterion** #12: Study the collaboration and task allocation problems among the constituents of the composite service to guarantee building reliable and well-performing

Table 3.3: Criteria for the trust and reputation models in the composite services' architecture

ID	Criterion	References
C8	Capture the responsibility of each constituent.	[93, 50, 127]
C9	Consider that the responsibility of each constituent cannot be fully observed.	[93, 50, 127]
C10	Take into account the dynamism of the behavior of the constituents even when this change does not affect the overall quality of the composite service.	[50]
C11	Monitor the variations in the QoS parameters of the constituent.	[168]
C12	Study the collaboration and task allocation among composite service's constituents.	[163, 164]
C13	Account for the active malicious constituents.	[57, 162]

compositions.

- **Criterion** #13: Consider the existence of *active malicious constituents* whose objective is to join some compositions and launch attacks against the composite service or some partner constituents.

These criteria are summarized in Table 3.3. Numerous trust and reputation models have been advanced for the composite architecture. The dominant trend of these models use statistical techniques to compute the trust for the constituents, while others employ game theory to model the collaboration and task allocation issues. More details about these models and their associated approaches are discussed in what follows. Thereafter, the discussed approaches are compared in Table 3.4 w.r.t the criteria presented in Table 3.1 and Table 3.3.

3.5.2.1 Statistics-based models

In the context of composite services, statistical models [107, 93, 92, 50, 68] have been widely used to model the relationships among the individual constituents and learn the responsibility of each constituent in the overall composite service. The objective is to help providers improve the quality of their existent compositions and make wise future selections. The challenges that led to the adoption of statistical models are the dynamic nature of the composite architecture and the difficulty of observing each constituent's quality. In fact, the dynamic aspect of the composition process makes it difficult to learn the order of the constituents, which is important to learn the performance of each individual service(through mapping its order in the composite service with the performance obtained at a certain time moment). Moreover, the quality of each constituent cannot be always observed. For instance, when dealing with a hotel reservation service, the user may observe sometimes that a certain constituent always responds before the others. However, such information may not be always observable. Thus, statistical techniques are used to predict the quality of the constituents from the overall composite service's quality.

In [93], the authors employed Bayesian Network to assess the trustworthiness of the constituents through a reputation-based trust mechanism. For this purpose, a probabilistic approach that is able to learn the composition structure (i.e., the order of the constituents in the composition process) of the composite services and compute the trust scores for the constituents is advanced.

Another statistical method, the Beta Mixture [80], was employed in [50] to assign trust for the components of the composite services. Trust is assigned for components based on their responsibilities, while taking into consideration the dynamism in the QoS and the fact that not all the observations can be noticed.

Although statistics-based models account for the dynamic characteristics of the QoS parameters, they cannot provide decisive solutions for this problem. In fact, these models suggest tracking the most recent behavior of the services to predict their current performance. Nonetheless, the QoS of the services may change on demand (not in a regular manner) [168], which makes tracking the most recent behavior less likely to make reliable predictions. For instance, an online car rental service may face important degradation in its performance during the promotion time due to the pointedly increased number of orders. In this case, the current performance is unlikely to be predicted from the recent performance since the change in the QoS does not happen in a regular manner. Therefore, a monitoring mechanism that can capture the variations in the performance is recommended [168]. Moreover, these models ignore the collusion scenarios that may occur among the constituents of the composite service and that may lead to false estimations of these constituents' trust values. For instance, constituents may collude according to different scenarios to mislead the predictions. For instance, those constituents might collude to promote each other in order to obtain high reputation scores. Additionally, statistics-based models do not study the collaboration and task allocation issues among the constituents. Furthermore, the topic of malicious constituents that join compositions to perform malicious objectives was not addressed yet.

3.5.2.2 Game-theoretic-based models

Game theory is a formal study of conflict and cooperation that applies whenever the actions of several agents are interdependent. A few game-theoretic-based models [164, 163] were proposed to address the topic of trust and reputation in the composite architecture. The objective of these models is to model the competition among constituents seeking to be allocated tasks in the compositions and select hence the appropriate candidate with the aim of maximizing the probability of performing the allocated tasks successfully.

As an example, Yahyaoui [164] proposed a trust-based game whose objective is to

Table 3.4: Comparison summary between the main trust and reputation approaches in the composite architecture

Approach	Model	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
Mehdi et al. [93]	Statistical	✓	✓				✓	✓	✓	✓				
Hang et al. [50]	Statistical	✓	✓			✓	✓	✓	✓	✓	✓			
Paradesi et al. [107]	Statistical	✓	✓	✓			✓	✓	✓					
Yahyaoui [163]	Game theory	✓		✓			✓	✓	✓				✓	
Yahyaoui [164]	Game theory	✓		✓		✓	✓	✓	✓				✓	

model the competition among services seeking to get allocated with tasks and to select the appropriate candidate. To achieve this, services use a Bayesian model to compute a trust value for every other service willing to collaborate with and play a game to select the appropriate candidates.

Game-theoretic-based models address an important topic in the context of composite services, which is the task allocation regulation. This issue is important since it helps increase the probability of the composite service for achieving the allocated task with better performance. However, these models did not capture the whole picture of the task allocation problem. More precisely, they ignore the collusion scenarios that may occur among services. Practically, some services may collude to promote/demote each other or some other services, which may lead to inappropriate selection and create unreliable compositions. As with the statistics-based models, game-theoretic-based models ignore as well the topic of malicious constituents that join compositions to perform malicious attacks. Different from statistics-based models, game-theoretic-based models do not evaluate the responsibility of constituents in the composition process.

3.5.3 Communities of Services

Communities of services can be viewed as groups of services sharing the same functionality but differing in their non-functional properties [82] (e.g., a community of car rental services). Creating communities has a two-fold objective resulting in benefit for both services and users. Services will be exposed to wider groups of users and will have chances to contribute in a greater number of compositions. Users, in their turn, will get their requests fulfilled with better quality as a result of the cooperation that takes place among the services within communities [82]. The topic of trust and reputation has been extensively addressed in the communities of services, where the objective is to enable services to work and cooperate within a truthful environment. To attain this objective, a collection of requirements have to be satisfied. As communities are composed of single services and can involve some kinds of functionally-similar compositions among community members, the requirements proposed for both the single and composite architectures (Table 3.1 and

Table 3.3 respectively) apply as well for the community-based architecture in addition to other important requirements such as [38, 62, 22, 59, 140, 57, 162, 36]:

- **Criterion** #14: Investigate the community joining strategies in a thoughtful manner, i.e., in a way that enhances/maintains the community's performance and reputation.
- **Criterion** #15: Adapt the trust values to the highly dynamic environment of the communities wherein services are continuously joining and leaving.
- **Criterion** #16: Consider the existence of *selfish* or *passive malicious services* in the communities whose objective is to manipulate the reputation values by means of malicious actions.
- **Criterion** #17: Consider the existence of *active malicious services* whose objective is to join communities to lunch some attacks leading to disrupt the functioning of these communities.
- **Criterion** #18: Study a fully malicious model that mimics the reality, where all the parties that are intelligent agents are assumed to behave maliciously, seeking foremost their own objectives.

These criteria are summarized in Table 3.5. The topic of trust and reputation in the services communities was first addressed in [38], where the authors tried to adapt the services community architecture to support trust and reputation models. This was achieved by proposing an architecture of four components: user-agent; provider-agent; extended Universal Description, Discovery and Integration (UDDI)[2]; and reputation system. They defined as well some metrics to evaluate the reputation of the community from the perspectives of both users and providers. Most of the existing trust and reputation models proposed for the community-based architecture build on and extend this reputation model. These models fall into two major classes: analytical models, and game-theoretic-based models. More details about these models and their associated approaches are discussed in the following subsections. Thereafter, the discussed approaches are compared in Table 3.6 w.r.t the criteria presented in Table 3.1, Table 3.3, and Table 3.5.

3.5.3.1 Analytical Models

Analytical models are mathematical models that use equations to analyze the relationships among a set of variables. These models have been used for the services communities to analyze the relationships among the reputation parameters of the services in order to help them decide whether to join communities or to work alone.

[2]UDDI is a platform-independent XML-based mechanism to register and find Web service applications

Table 3.5: Criteria for the trust and reputation models in the community-based architecture

ID	Criterion	References
C14	Investigate the joining strategies in a thoughtful manner.	[38, 62, 59]
C15	Consider the highly dynamic environment of the communities.	[62, 59]
C16	Consider the existence of selfish services.	[62, 22, 140]
C17	Account for the active malicious services.	[57, 162]
C18	Study a fully malicious model.	[36]

In [61], the authors perform an analysis on the incentives that would motivate a community (containing one or more elements) of services to join another community or to stay alone. For this purpose, they formulate a performance function composed of two factors: use of allocated services, and simultaneous obtained feedback (on the services' performance). Based on the proposed function, the authors stated that a community will be encouraged to join another community if: (1) it is overloaded by a huge number of requests, or (2) it is unable to attract enough services.

In [60], the authors analyze the impacts that reputation parameters have on each other in order to help services decide whether to join a community or to stay alone. Two cases are considered: The service is overloaded or the service is idle. In the first case, the analysis results show that (1) the large increase in the number of requests would result in a decrease in the service's reputation (since the service might need to drop some incoming requests in this case), and (2) the change in the reputation in the current time either positively or negatively leads to a negative change in the reputation in the next time unit. In the second case, the analysis revealed that a positive rate of reputation change at a certain time results in a positive rate of change in the next time slot.

In [62], the authors developed an analytical model that analyzes the incentives that would demotivate the community master from behaving maliciously by either increasing its reputation level or decreasing other communities' reputation levels illegally. To tackle this issue, a third-party called the agent controller is assigned the role of recognizing the misbehavior by comparing the community's reputation change (improvement or degradation) between two slots of time and matching this change with a predefined threshold.

Although analytical models tend to provide strong solutions since they are based on mathematical proofs, these models fail to provide solid decision making frameworks for the services since they restrict the analysis to a few parameters. For example, [61] restricts the reputation assessment to three metrics; thus ignoring some important factors such as capacity of handling requests. Similarly, the analysis presented in [60] is limited to two reputation parameters computed by services, thus eliminating the reputation parameters related to the users. Likewise, the authors in [62] limit the analysis to three reputation metrics. Moreover, analytical models provide no bootstrapping mechanism to compute initial trust values for the new services and communities. Furthermore, they do not account for

Table 3.6: Comparison summary between the main trust and reputation approaches in the community-based architecture

Approach	Model	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18
Khosravifar et al. [61]	Analytical	✓	✓				✓	✓	✓						✓	✓			
Khosravifar et al. [62]	Analytical	✓	✓	✓	✓		✓	✓	✓	✓			✓		✓	✓	✓		
Khosravifar et al. [60]	Analytical	✓					✓	✓	✓						✓	✓			
Bentahar et al. [22]	Game theory	✓	✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓	✓		
Khosravifar et al. [59]	Game theory	✓	✓				✓	✓	✓				✓		✓	✓			

the malicious services that join communities to launch attacks deteriorating communities' QoS and reputations.

3.5.3.2 Game-theoretic-based Models

Game theoretical models have been widely investigated in the community-based architecture, where they are mainly used to address the shortcomings of the analytical models and tackle the concept of joining communities in a more systematic manner.

A one-stage game theoretical model has been developed in [59] to provide services with a decision making framework that helps them adopt strategies inside and outside communities. Using the proposed game, the authors derive a threshold to be compared with the expected performance. If the expected performance exceeds the threshold, then the strategy will be joining for the single service and accepting the invitation to join for the community. Another threshold is derived to control the strategies of the services inside the communities. If the expected performance exceeds this threshold, the strategy of the single service would be leaving the community; otherwise it would prefer to remain.

Game theoretical models were used as well to model the collusion scenarios that occur among services acting as intelligent agents. The objective is to guarantee a truthful environment where the involved entities act honestly. In this context, a repeated game model was derived in [22] in order to maintain sound reputation mechanism in the presence of malicious services seeking to enhance their reputations by means of fake feedback. To this end, the authors discussed four scenarios the controller of the community (charged with monitoring the feedback file against manipulations) may face such as: malicious act not penalized, truthful act penalized, truthful act not penalized, and malicious act penalized. Thereafter, a repeated game of two players (service and controller) is analyzed to derive the best strategy for both players. This analysis revealed that if the service is made aware of the penalties that it may undergo as well as of the controller's detection accuracy, then the system will fulfill a sound and secure state.

Game-theoretic-based models introduced an in-depth reasoning about the behavior and actions of the different agents involved in the community-based architecture and are hence able to provide effective and powerful decision making frameworks for these agents.

Table 3.7: Comparison summary among the class models in each architecture

Architecture	Model	Purpose	Limitations
Single	Feedback-based	Build trust value from users' reviews	Unfair ratings. Dependency on the credibility of the majority of raters. Provide no bootstrapping mechanism.
	Statistics-based	Combine different sources of trust	Cannot compute trust values for the new services.
	Fuzzy-logic-based	Infer the rationale behind users' reviews	Provide no mechanism to compute the final trust value. Provide no bootstrapping mechanism. Do not consider the trust dynamism.
	Data-mining-based	Analyze users' reviews	Lack of in-depth methodology. Unfair ratings. Provide no bootstrapping mechanism. Do not consider the trust dynamism. Ignore the objective sources of trust.
Composite	Statistics-based	Learn the responsibility of the composite service's constituents	Cannot obtain reliable predictions on the variations in the QoS parameters. Ignore the collusion scenarios among the composite service's constituents. Do not consider the malicious constituents that join compositions to perform malicious attacks. Do not study the collaboration and task allocation issues among constituents.
	Game-theoretic-based	Regulate the task allocation among composite service's constituents	Ignore the collusion scenarios among the composite service's constituents. Do not consider the malicious constituents that join compositions to perform malicious attacks. Do not evaluate the responsibility of constituents in the composition process.
Community	Analytical	Analyze the relationships among the reputation parameters	Provide no bootstrapping mechanism. Limited to a few reputation parameters. Do not account for the malicious services that join communities to launch attacks.
	Game-theoretic-based	Provide decision making frameworks for services and communities	Rely on fully-honest or semi-honest adversary models. Provide no bootstrapping mechanism.

Nonetheless, the main problem of the game-theoretic-based models in this architecture is that they rely on fully-honest or semi-honest adversary models that assume the existence of one or more trusted parties. For example, the work presented in [59] does not consider the possible malicious nature of the services joining the communities. It assumes hence that all the parties involved in the game (master, single services, and users) are trusted. Moreover, the controller agent in [22] is responsible for supervising the feedback file against false feedbacks without considering the case where the controller agent may be itself involved in the collusion between services and consumers. In addition, game-theoretic-based models provide no bootstrapping mechanism to compute initial trust values for the new services and communities.

3.5.4 Summary of Findings

Table 3.7 provides a comparison summary among the classes of models defined in each architecture. The Table illustrates the purpose behind using each class of the architecture in question and highlights its main limitations. Moreover, we summarize in Table 3.8 the discussed trust and reputation approaches to help readers visualize and understand them.

Table 3.8: Summary of the main trust and reputation approaches proposed for services

Approach	Addressed Problem	Contribution
Maximilien and Singh [90]	Trust-based selection for services	Ontology-based framework that considers user preferences, providers advertisements, and QoS monitoring.
Malik and Bouguettaya [85]	Assessing the reputation of service providers	Reputation assessment framework that considers raters credibility, personalized preferences, temporal sensitivity, and first-hand knowledge.
Nguyen et al. [100]	Combining different sources of trust	Trust and reputation model that integrates different kinds of trust sources and evaluates the credibility of raters.
Sherchan et al. [126]	Infer the rationale behind users's reviews	Investigate the relationship between the objective dimension and the subjective dimension using fuzzy approach.
Thurow and Delano [130]	Analyze users' reviews	Text mining technique that extracts information about services' QoS parameters from the users' reviews.
Mehdi et al. [93]	Assessing the trustworthiness of composite services' components	Probabilistic approach that learns the composition structure and computes trust.
Hang et al. [50]	Assigning trust for composite service components	Trust-based approach that dynamically learns the responsibilities of components and computes trust.
Yahyaoui [164]	Collaboration among services	Trust-based game that models the competition among services for tasks allocation.
Elnaffar et al. [38]	Assessing services communities using reputation-based approach	Extension of the Web services architecture to support communities and reputation model design.
Khosravifar et al. [61]	Analyzing incentives that encourage services to join communities	Performance function formulation to help communities adjust their joining strategies.
Khosravifar et al. [62]	Evaluate the reputation of communities in the presence of malicious coordinators	Sound logging mechanism that motivates the well-behavior of community coordinators.
Khosravifar et al. [60]	Analyzing the impacts that reputation parameters have on each others	Theoretical analysis that helps services decide whether to work alone or to join communities.
Bentahar et al. [22]	Maintaining sound reputation in the presence of malicious services	Game theoretical model that investigates the incentives that would encourage services to act truthfully.
Khosravifar et al. [59]	Help services adopt strategies inside and outside communities	Game model between services and coordinator to analyze the payoffs for both parties based on different strategies.

3.6 Discussions and Research Directions

A collection of trust and reputation models has been introduced in the SOA. These models differ in the topics they address, which are imposed mainly by the architecture of services in question. They differ as well in the manner they use to construct the trust value for the services. Therefore, we base our classification of the existing trust and reputation models on these two perspectives. In fact, we present a two-level classification scheme that classifies the trust and reputation models based on the (1) architecture they are targeting as a high-level classification; and (2) technique they use to construct the trust value as a low-level classification. Detailed analyses and comparisons are derived from these classifications; uncovering prospective topics for future study and investigation. In the following, we discuss the results obtained from these comparisons, highlight some possible research topics in each architecture, and illustrate the future perspectives that are entailed by this work.

3.6.1 Single Architecture

Trust and reputation mechanisms have been widely used and investigated in the single architecture of services. Different approaches were proposed targeting numerous topics. Since each approach focuses on a specific perspective, some important criteria are missed. Practically, some approaches focus on the subjective perspectives and ignore the

objective perspectives. Some approaches do not account for the dynamism of the trust. Additionally, some approaches disregard the bootstrapping issue, which constitutes an important challenge for any trust and reputation mechanism. Some proposals don't assess the credibility of the ratings used to build the trust and reputation model, which may trigger collusion and deception problems. Besides, some approaches are based on the assumption that the majority of the ratings are truthful, which is not always realistic. Therefore, a more comprehensive trust and reputation model considering all the mentioned criteria is needed.

3.6.2 Composite Architecture

Numerous approaches were proposed to tackle trust and reputation in the composite architecture. These approaches are either statistics-based or game-theoretic-based. The goal of the statistics-based models is to learn the responsibility of the composite service's constituents in order to enhance the current compositions and facilitate future selections. The main problem of these models is that they predict the change in performance of the services based on the most recent performance, which cannot yield accurate predictions. It would be recommended to investigate a monitoring mechanism that is able to capture the variations in the QoS parameters of the constituents. Moreover, statistics-based models do not address the task allocation among composite services' constituents and do not consider as well the collusion scenarios that may take place among these constituents and that may influence the predictions. It would be interesting to develop a more comprehensive approach that is able to learn the responsibilities of the constituents based on a monitoring mechanism that captures the dynamism in the performance and under a colluding scenario. On the other hand, game-theoretic-based models focus on the topics of collaboration and task allocation among the constituents of the composite service. Similar to the statistics-based models, game-theoretic-based models ignore the collusion scenarios that may be initiated by the services. More specifically, services may collude to promote each other and get higher chances to get allocated with tasks and/or promote/demote other services. Thus, it is important to consider the collusion scenarios to obtain fair and reliable task allocations. Furthermore, the topic of active malicious constituents that join compositions to perform malicious objectives is not addressed yet. These malicious constituents might take advantage of several vulnerabilities that exist in the composite architecture to perform their goals such as: long-term partnerships and services' resource constraints. The main attacks that can be launched against the composite architecture are merged and presented with those of community-based architecture in Section 3.6.3 as the same attacks apply for both architectures since communities can be viewed as long-term compositions among services sharing the same functionality. The studied attacks are limited

to those that have major impacts on the reputation and QoS of the composite services, associated with the main metrics that influence the trust value assigned by users towards composite services. Moreover, the simulation results that show the impact of malicious services that launch these attacks on the composite services can be found in Section 3.7.

3.6.3 Community-based Architecture

Trust and reputation models in the community-based architecture fall into two main classes: analytical and game-theoretic-based. The aim of the analytical models is to analyze the relationships among the reputation parameters of the services in order to help them choose strategies either to join communities or to stay alone. The problem of these models is that they are limited to a few reputation parameters. More thorough analysis involving a wider set of important parameters is required to provide effective decision making frameworks. On the other hand, game-theoretic-based models provide more thoughtful decision making frameworks for the services and have the advantage of considering the existence of *malicious agents* that constitute a serious challenge to the community-based architecture. These malicious services may, individually or as a result of collusion scenario with some customers or communities, join the communities for the purpose of launching attacks leading to harm or deteriorate some other community members or the community as a whole. In addition to the vulnerabilities of the composite architecture mentioned in Section 3.6.2 which are applicable also in the community-based architecture, malicious services can exploit additional vulnerabilities specific to the community-based architecture such as: dynamic topology (freedom to join and leave communities), scalability (no restriction on the number of community members).

Some existing game-theoretic-based models [62, 22] tackled the existence of *passive malicious services* whose objective is to increase their reputations compared to other members. These approaches fail to provide strong protection against such a misbehavior since they rely on the existence of a central party such as controller agent that will monitor and take decisions. These parties are intelligent agents that may be tempted to get involved in the collusion scenarios, which may lead to false decisions. A more nested scenario wherein all the parties are assumed to potentially behave maliciously is recommended. In addition, the topic of *active malicious services* whose objective is to harm or destroy other members and/or communities by launching active attacks was ignored. In the following, we highlight the main attacks that are applicable for both the composite and community-based architecture. Recall that the studied attacks are restricted to those that significantly affect the trust, reputation, and QoS of the services compositions/communities.

1. **Request Drop Attack:** Composite services and communities are usually based on

the assumption that single services are willing to cooperate in order to respond to the complex requests with better performance. However, some malicious services may join a certain composition/community and refuse to cooperate and fulfill the requests. The simplest form of this attack is when a certain component/member refuses all the requests it receives. However, this malicious component/member faces the risk of being easily detected and fired by the composition designer or community coordinator. A more intelligent derivation of this attack is when malicious components/members perform the requests dropping in a selective manner. In such a way, these components/members will drop the requests coming from certain clients or services, every t slots of time, or every r requests. This kind of attacks is called *Selective Request Drop* (SRD) attack [148].

2. **Denial of Service (DoS) Attack:** This attack aims at deteriorating or reducing the service's availability. In this attack, a malicious component/member working within a composition/community may send a request to its partners to exhaust their resources (e.g, memory capacity) in such a way that makes them unavailable for responding to further requests. The most two important DoS attacks on XML-based services such as Web services are *Coercive Parsing* and *Oversize Payload* [46]. In the Coercive Parsing attack, a pointedly nested XML document is used to consume the service's memory. In the Oversize Payload attack, an extremely large XML document is employed for this purpose.

3. **Sybil Attack:** This attack takes place when malicious components/members create illegitimately a large number of fake identities (fabricated identities) or impersonate other legitimate services in the composition/community (stolen identities). The goal of the attacker in this case is to appear and operate as multiple distinct services in such a way that enables it to take control over the whole composition/community. This attack may occur only in case of communities and long-term compositions and might be exploited by the attackers to achieve several malicious objectives in different aspects.

4. **Outage Attack:** This attack occurs when malicious components/members committed to performing a certain task within a whole process suddenly stop their functioning, which leads to interrupt the functioning of the whole process.

5. **Sinkhole Attack:** In this attack, malicious components/members seek to lure nearly all the requests (from clients/or from other services). This attack is done by making a compromised component/member look attractive to clients/services by claiming bogus high reputation score.

6. **Eavesdropping Attack:** This attack happens when malicious components/members

collect information from the composition/community (e.g., application-specific messages content) they belong to in favor of other competitor compositions/communities. Thereafter, these malicious components/members may decrease their performance in a way that ends them up being fired from the current composition/community. This allows them to join other compositions/communities and use the collected information for malicious purposes.

7. **Composition Exclusion (CompEx) and Community Exclusion (CommEx) Attacks:** In such attacks, malicious components/members deteriorate the reputation of the composition/community they belong to in such a way that makes this composition/community appear undesirable to deal with by any client or service. This attack can be performed by applying the request drop, DoS, outage, or sybil attacks in such a way that makes the composition/community look unable or unwilling to fulfill the incoming requests.

8. **Component Exclusion (CptEx) and Member Exclusion (MembEx) Attack:** Such attacks happen when malicious components/members start launching attacks (e.g., DoS) leading to exclude a specific victim from the composition/community by decreasing its reputation in a drastic manner.

We present and discuss in Section 3.7 the simulation results that show the impact of active malicious services launching each of these attacks against the community-based architecture.

3.6.4 Future Perspectives

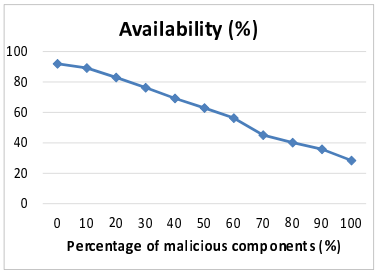
In this chapter, we classified and compared the trust and reputation models proposed in the SOA on the basis of a set of defined criteria. The results of the work may be used in the benefit of services' providers, consumers, and research community. By developing a classification scheme and proposing a set of criteria for each class, we aim to help providers enhance the quality of their services by letting them learn the factors that affect the user's judgement on the services and consider hence those factors designing their services. From criteria $C1$, $C2$, and $C6$ (Table 3.1), providers will learn that users care about a wide variety of QoS metrics when building their reputation towards services and that they should keep up the quality of services at a good level since the trust is subject to change over the time. Criteria $C8 - C13$ (Table 3.3) help providers enhance the quality of their compositions by stressing the importance of the issues of learning the performance of the composite service's constituents and managing the task allocation in a thoughtful manner. Criteria $C14 - C18$ (Table 3.5) help providers design high-quality and secure communities of

services. As a result, consumers will enjoy services with better quality and performance. They will be motivated as well to provide truthful feedback by learning from criteria $C3$, $C4$, and $C7$ (Table 3.1) that the reputation system should be able to discard untrustworthy ratings. Moreover, our analysis reveals some topics that are worth studying and investigating. More specifically, we raise the topic of active malicious services in the composite and community-based architectures by defining such malicious services, clarifying their objectives, highlighting some vulnerabilities that they might exploit, and elucidating their negative impacts by means of simulation experiments conducted on a real-life dataset. Thus, our work can be used by the security research community as a starting point to study and explore security-based models targeting these malicious services.

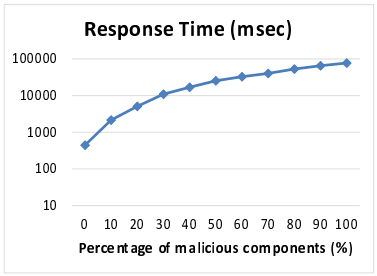
3.7 Impact of malicious services on the composite and community-based architectures

To study the impact of the active malicious services, Figures 3.2, 3.3, and 3.4 describe their effects on the composite architecture while Figures 3.5, 3.6, and 3.7 depict their impacts on the community-based architecture. It is well-predictable that the existence of malicious services leads to negative implications on the QoS and reputation parameters. However, by advancing simulations on various types of attacks, we are providing readers with the ability to visualize and compare the implications of these attacks. This helps them infer the security plans that should be designed to prevent and/or detect such attacks. For example, one may notice from Figure 3.4a that the availability of the composite service begins to drop in a severe manner starting from 10% of Sinkhole attackers. This is due to the fact that although the number of attackers is relatively small, malicious services in this type of attacks work on attracting nearly all the requests incoming to the composition by claiming bogus reputation, which allows them to perform the drop in an extremely severe manner. Similarly, it is worth observing as well that the availability in the Sinkhole attack drops more severely than that of both the Selective Request Drop (Figure 3.2a) and DoS (Figure 3.3a) attacks. The same intuition applies as well for the community-based architecture. As a result, the reader may conclude that targeting the Sinkhole attack is extremely urgent and that the existence of even a small number of such attackers should not be tolerated.

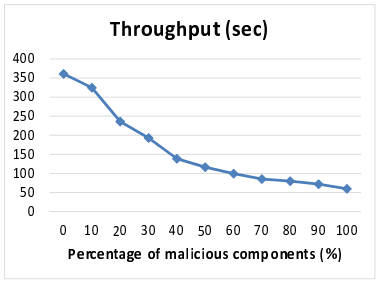
Several metrics are used throughout simulations such as: composition/community reputation (a value from the interval $[0, 1]$), availability (time period in which a service is ready for use), response time (time between the submission of the request and the receipt of the response), and throughput (number of requests that can be processed per time unit). The simulation application is written in C# using Visual Studio and the domain of flight booking



(a) SRD decreases availability

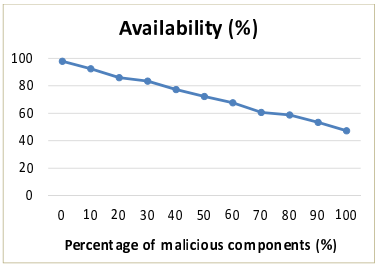


(b) SRD increases response time

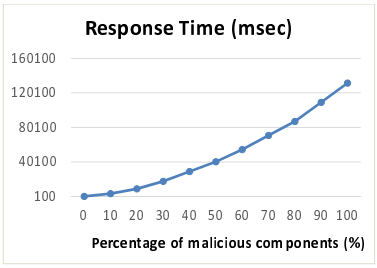


(c) SRD decreases throughput

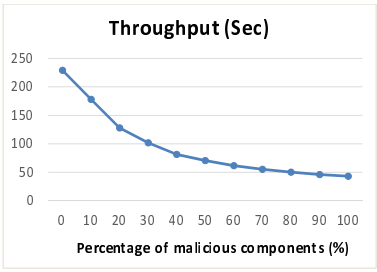
Figure 3.2: Impact of selective request drop attack on the composite architecture



(a) DoS decreases availability



(b) Outage increases response time

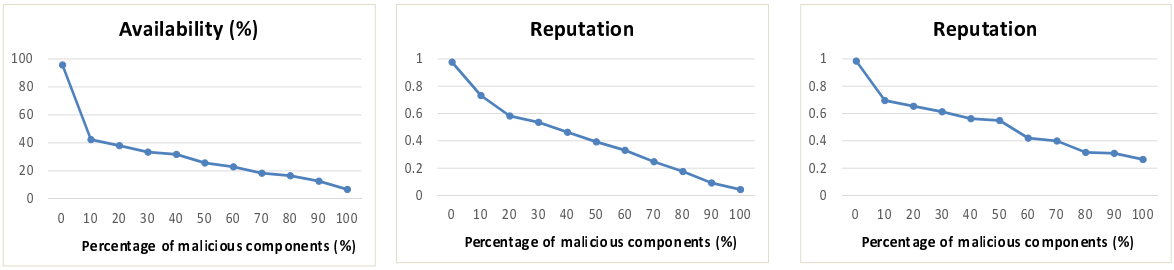


(c) Sybil decreases throughput

Figure 3.3: Impact of DoS, Outage, and Sybil attacks on the composite architecture

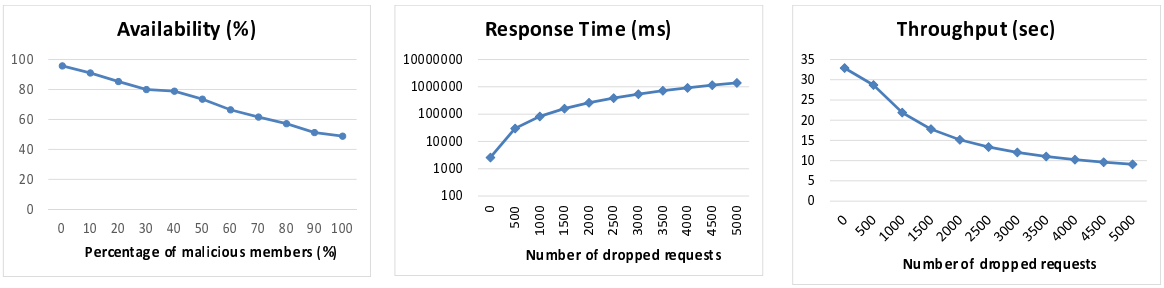
is addressed. The information related to services is populated from the Quality of Web Service (QWS) real dataset [12] that includes 2507 real services functioning on the Web and containing the QoS values of 9 parameters [63]. Each user's request contains the flight dates, the origin and destination, type of tickets (one way or return), and number of guests. The response contains different flights pertaining to different companies, prices, timing, etc.

Selective Request Drop (SRD) attackers decrease the availability in both the composite and community-based architectures as shown in Figures 3.2a and 3.5a respectively. They increase as well the response time in both architectures as shown in Figures 3.2b and 3.5b respectively and decrease the throughput as shown in Figures 3.2c and 3.5c respectively. DoS attackers reduce the availability of the composite and community-based architectures as shown in Figures 3.3a and 3.6a respectively. Outage attackers cause an increase in the response time in the composite and community-based architectures as shown in Figures 3.3b and 3.6b respectively. Figures 3.3c and 3.6c show that sybil attackers decrease the throughput of the composite and community-based architectures respectively. Sinkhole attackers considerably decrease the availability of the composite and community-based architectures as shown in Figures 3.4a and 3.7a respectively. As for the composition exclusion attack, it causes a significant drop in the composite services' reputation as shown in Figure 3.4b. Similarly does the community exclusion attack for the community-based



(a) Sinkhole decreases availability *(b) CompEx deceases reputation* *(c) CptEx deceases reputation*

Figure 3.4: Impact of Sinkhole, Composition Exclusion, and Component Exclusion attacks on the composite architecture



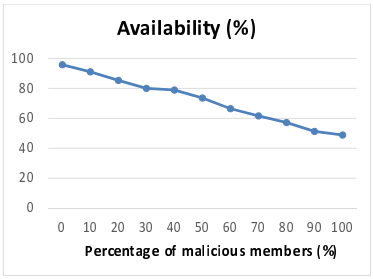
(a) SRD decreases availability *(b) SRD increases response time* *(c) SRD decreases throughput*

Figure 3.5: Impact of selective request drop attack on the community-based architecture

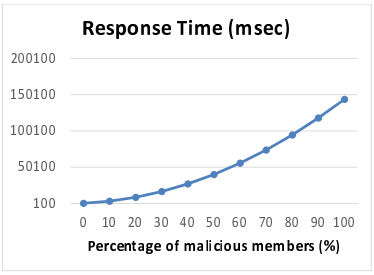
architecture as depicted in Figure 3.7b. Component exclusion attackers cause as well a considerable drop in the composite services' reputation as described in Figure 3.4c. Similarly, member exclusion attackers lead to a significant drop in the communities' reputation as shown in Figure 3.7c.

3.8 Conclusion

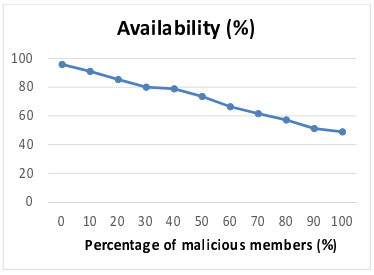
In this chapter, we proposed a two-level classification scheme for the trust and reputation models in the SOA according to the architecture of services they target, and the technique they use to compute the trust value. A collection of criteria were defined for each architecture based on which the class models as well as the major approaches in each architecture were compared. This comparison revealed some important issues that need further study and investigation. One of the important challenges is the existence of active malicious services in the composite and community-based architectures whose objective is to harm the compositions/communities by decreasing their performance and reputation. In this context, we described eight possible attacks that have major impacts



(a) DoS decreases availability

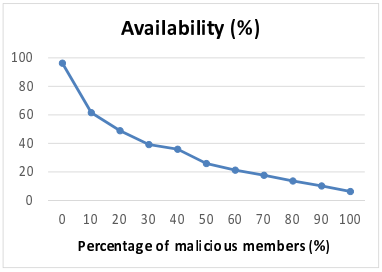


(b) Outage increases response time

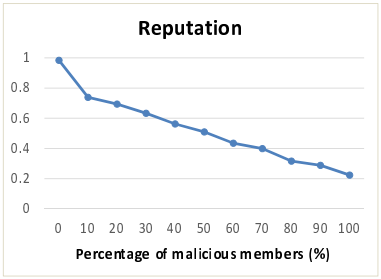


(c) Sybil decreases throughput

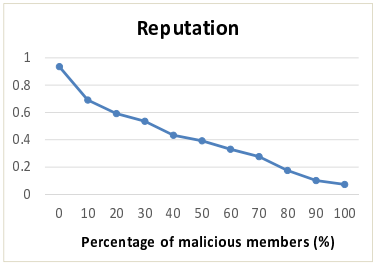
Figure 3.6: Impact of DoS, Outage, and Sybil attacks on the community-based architecture



(a) Sinkhole decreases availability



(b) CommEx deceases reputation



(c) MembEx deceases reputation

Figure 3.7: Impact of Sinkhole, Community Exclusion, and Member Exclusion attacks on the community-based architecture

on compositions/communities' performance and reputation. We conducted as well a series of simulation experiments on a real-life services dataset to show the negative impacts of such malicious services using several QoS metrics. Simulation results reveal that the existence of malicious services has the potential to considerably increase the response time and decreases the reputation, availability, and throughput. This opens the door for future researchers to investigate security-based models for the purpose of protecting the composite and community-based architectures from such potential attacks.

The results of this chapter are used in Chapter 5 and Chapter 6 for the purpose of building effective trust models to be used toward the accomplishments of the security-oriented research objectives (i.e., **Objective** 2 and **Objective** 3 discussed in Chapter 1), which aim, respectively, to minimize the number of passive malicious services in the multi-cloud services community architecture and build effective intrusion detection solutions for the active malicious services within a limited budget of resources.

Chapter 4

Forming Communities Among Services Deployed in Clouds Having Uneven Business Capabilities

With the advent of cloud computing and the rapid increase in the number of deployed services, the competition between functionally-similar services is increasingly governing the markets of services. For example, Amazon and Google are in an intense competition to dominate the market of cloud-based Web services. Such a highly competitive environment motivates and sometimes obliges services to abandon their pure competitive strategies and adopt a cooperative behavior in order to increase their business opportunities and survive in the market. Several approaches have been advanced in the literature [82, 58, 70, 73, 63] to model the cooperation among services in a community-based environment. However, the existing approaches suffer from two main drawbacks that limit their effectiveness in the real-world services market. First, they rely on a centralized architecture wherein a *master* entity is responsible for regulating the community formation process, which creates a single point of failure. Second, they ignore the business potential of the services and treat all of those services in the same way, which demotivates the participation of the well-positioned ones in such communities. In this chapter, we propose a Stackelberg game theoretical model [139] for building communities among services belonging to different clouds having uneven business capabilities. We define two types of services: *leaders* and *followers*. Leaders are those services that are deployed in cloud centers enjoying high business capabilities in terms of reputation, market share, and capacity of handling requests, whereas followers are those services that reside at cloud centers having less strong business capabilities. Thereafter, we model the community formation problem as a

virtual trading market between these two types of services[1].

4.1 Proposed Community Formation Model

4.1.1 Solution Overview

As is the case in the real-world business markets, two types of services can be distinguished: *leaders* and *followers*. Leaders refer to those services that are deployed in strong cloud centers, which allows them to enjoy high levels of reputation, market share, and capacity of handling simultaneous requests[2]. Practically, being deployed in a strong cloud center means that services have an abundance of software and hardware infrastructure components (e.g., CPU, memory, servers, storage, networking, etc.), which thus increases their ability to handle a larger pool of simultaneous requests, enlarges their market shares, and allows them consequently to have high reputation scores. On the other hand, followers are those services that are deployed in cloud centers having less business capabilities, which makes their parameters less impactful compared to those of the leaders. The proposed community formation model can be summarized as follows. Due to its superiority in the market (in terms of parameters), the leader has the right to pre-select a set of followers it is interested in confederating[3] with (e.g., top 10 hotel reservation services). It decides also about the quota of followers that will be considered for final selections (e.g., 5 services out of 10). The leader publishes its offer consisting of its own reputation, market share, and capacity to the set of pre-selected followers. Followers compete then with each other to decide about the payment to be given to the leader under the offered parameters with the purpose of convincing the leader to consider them when selecting the final quota without having to make too much payment. For the leader, the objective is to select the quota of followers that allows it to reach the optimal utility in terms of reputation, market share, capacity, and revenue. This forms a typical two-stage Stackelberg game wherein leaders optimize their strategies (i.e., quota) after having learned the effects of their decisions on the behavior of the followers who play their best responses to the leader's offer. To solve the game, *backward induction* reasoning [16] is used for computing the equilibrium points for both parties. This is done by finding the optimal payment that followers tend to make as a best response to the leader's offer and substituting this information into the leader's utility function.

[1] The content of this chapter is published in [144]

[2] In the rest of this chapter, the term "parameters" is used to represent the reputation, market share, and capacity of handling requests of the service.

[3] The term "confederation" is used to describe the act when services join each other to form a joint community.

4.1.2 Aggregation Functions

The decisions made by the services (both leaders and followers) are influenced by four main metrics: reputation, market share, capacity, and payment/revenue. As rational agents, services tend to increase their reputation scores to gain more reliability vis-à-vis users, which helps them receive a bigger task pool. Thus, a key factor in forming communities is to ensure that the reputation of the group is at an acceptable level. Therefore, an aggregation function is needed for the services to help them calculate their expected reputation score after confederation and to compute the effects of this score on their utility functions. It is worth mentioning that we assume that the reputation score is bounded by 0 and 1. We propose a heuristic function each leader L and follower F uses to compute its expected aggregate reputation $E_R(L,F)$ after confederating with one another based on their current reputation scores R_L and R_F respectively. This function is given by Equation (4):

$$E_R(L,F) = f(R_L, R_F) \tag{4}$$

The function f should satisfy the following properties:
Property 1. R_L is always greater than R_F .
Property 2. f is monotonically increasing for the leader if the gap between R_L and R_F is small.
Property 3. f is strictly decreasing for the leader when the gap between R_L and R_F is big.
Property 4. f is strictly increasing for the followers if the gap between R_L and R_F is small.
Property 5. f is monotonically increasing for the followers if the gap between R_L and R_F is big.
Property 1 is a common property of Stackelberg games and represents a natural consequence of being a leader. *Property 2* and *Property 3* are important to motivate the leader to select followers having acceptable reputation scores and restrict hence the selection space of the leader to a certain number of well-reputable followers. *Property 4* and *Property 5* will motivate the services having low reputation values to improve their performance and get their reputation scores increased in such a way that gives them the opportunity of being selected by the leader to be part of future communities.

A possible definition of f is given by Equation (5).

$$E_R(L,F) = \min(R_L, R_F)^{|R_L - R_F|} \tag{5}$$

To illustrate how the aforementioned properties are important for maintaining healthy communities in terms of reputation, we give two examples that explain the aggregation process according to Equation (5).

Example 5. *Consider a leader service deployed in a strong cloud center* C_1 *and having a reputation score of* 0.8*. This leader decides to confederate with a follower deployed in a less strong cloud center* C_2 *and having a reputation value of* 0.7*. The aggregate reputation value resulting from the confederation between these two services according to Equation* (5) *will be:* $E_R = 0.7^{0.1} = 0.96$*. In this example, the gap between the two reputation values is* 0.1*, which is relatively small. Thus, the reputation of the leader got increased in a monotonic (i.e., small) manner (i.e.,* 16%*), while the reputation of the follower got increased in a strict (i.e., grand) manner (i.e.,* 26%*). Consequently, both the leader and follower are encouraged to confederate with each other in this example scenario.*

Example 6. *Suppose now that a leader having a reputation score of* 0.8 *decides to confederate with a follower having a reputation value of* 0.2*. In this example, the gap between the two reputation values is* 0.6*, which is obviously large. The aggregate reputation value according to Equation* (5) *will be:* $E_R = 0.2^{0.6} = 0.38$*. Thus, the reputation of the leader is decreased in a strict manner (from* 0.8 *to* 0.38*), while the reputation of the follower is increased in a monotonic manner (from* 0.2 *to* 0.38*). Consequently, the leader will not be motivated to confederate according this example scenario.*

As for the market share, this metric constitutes an important factor for the services deciding whether to confederate or not. It is an indicator of the competitiveness of a certain community amongst the other communities. Thus, ensuring an acceptable level of this metric is critical for the services. As is the case for reputation, an aggregation function is needed for the leaders and followers to help them calculate their expected aggregate market share and compute the effects of this metric on their utilities. For this purpose, a heuristic function is proposed to allow each leader L and follower F to compute its expected aggregate market share $E_{MS}(L,F)$ after confederating with one another based on their current market shares MS_L and MS_F respectively. This function is given by Equation (6):

$$E_{MS}(L,F) = g(MS_L, MS_F) \tag{6}$$

We assume that reputation and market share are similar in terms of value (i.e., both are in the interval $[0,1]$), thus the aggregation function proposed for the market share should also satisfy the properties proposed for the reputation aggregation function (i.e., Properties 1-5). However, the aggregation function for the market share should account for one additional constraint, namely the fact that the market share of a certain service is considered proportionally to the market shares of the other existing services. Therefore, the aggregation function of the market share should be divided by 2 (representing one leader and one follower) in order to reflect the fact that this aggregate market share will be partitioned

between the leader and follower. Thus, a possible definition of g is given by (7):

$$E_{MS}(L,F) = \frac{\min(MS_L, MS_F)^{|MS_L - MS_F|}}{2} \tag{7}$$

The capacity of handling requests has an important effect on the confederation decision since it indicates the extent to which the possible community will be able to handle simultaneous users' requests. Getting services together should increase the group's ability to deal with an increased number of simultaneous requests. Thus, the expected aggregate capacity $E_C(L,F)$ for the leader L and follower F after confederating with one another is calculated according to Equation (8) based on their current capacities C_L and C_F respectively.

$$E_C(L,F) = h(C_L, C_F) = C_L + C_F \tag{8}$$

4.1.3 Utility Functions

The utility function of the follower services is defined to be the sum of the variations in the follower's reputation, market share, and capacity after confederating with the leader multiplied by its proportional payment (that should be given to the leader) w.r.t the payments made by all other followers in the set pre-selected by the leader for possible confederation, minus its own payment given to the leader. The reasons behind considering the proportional payment are that (1) the payments made by the group of followers will affect the decision of the leader while making its final decision about the final quota and will influence thus each follower's utility, and (2) the payments made by the other followers are used by each follower to adjust its own payment. Thus, the utility function of each follower F is given by Equation (9).

$$U_F = \frac{P_F}{\sum_{i \in S} P_i} [\Delta(R_L) + \Delta(M_L) + \Delta(C_L)] - P_F, \tag{9}$$

where $\Delta(R_L)$ is the percentage of variation (e.g., +15%) in the follower's reputation after confederating with the leader L, $\Delta(M_L)$ is the percentage of variation in the follower's market share after confederating with the leader L, and $\Delta(C_L)$ is the percentage of variation in the follower's capacity after confederating with the leader L. P_F represents the payment given by the follower F to the leader, S is the set of followers pre-selected by the leader for possible confederation, and $\sum_{i \in S} P_i$ is the sum of payments given to the leader by each pre-selected follower $i \in S$. The variation percentages in the follower's reputation $\Delta(R_L)$, market share $\Delta(M_L)$, and capacity $\Delta(C_L)$ are given by Equations (10), (11), and (12) respectively:

$$\Delta(R_L) = (E_R(L,F) - R_F) * 100 \tag{10}$$

$$\Delta(M_L) = (E_{MS}(L,F) - MS_F) * 100 \tag{11}$$

$$\Delta(C_L) = (E_C(L,F) - C_F) \tag{12}$$

The utility function of the leader is defined to be the sum of variations in the leader's reputation, market share, and capacity after confederating with the followers set, along with the total payments collected from those followers.

$$U_L = \sum_{F \in S} [\Delta(R'_F) + \Delta(M'_F) + \Delta(C'_F) + P_F], \tag{13}$$

where $\Delta(R'_F)$ denotes the percentage of variation in the leader's reputation (e.g., +5%) after confederating with the follower F, $\Delta(M'_F)$ denotes the variation percentage in the leader's market share after confederating with the follower F, $\Delta(C'_F)$ denotes the variation percentage in the leader's capacity after confederating with the follower F, and P_F denotes the payment earned from each follower $F \in S$. The variation percentages in the leader's reputation $\Delta(R_F)$, market share $\Delta(M_F)$, and capacity $\Delta(C_F)$ are given by Equations (14), (15), and (16) respectively:

$$\Delta(R_F) = (E_R(L,F) - R_L) * 100 \tag{14}$$

$$\Delta(M_F) = (E_{MS}(L,F) - MS_L) * 100 \tag{15}$$

$$\Delta(C_F) = (E_C(L,F) - C_L) \tag{16}$$

4.1.4 Followers Payment Selection Game

Based on the utility functions illustrated in the previous section, *backward induction* reasoning [16] is used to analyze the equilibrium of the game. Given the reputation, market share, and capacity parameters decided by the leader along with the set S of pre-selected followers, each follower F in S decides an initial payment to make to the leader. This payment represents the variation percentage per unit in the follower's parameters. This value is computed according to Equation (17):

$$InitialPayment(F) = [\Delta(R_L) + \Delta(M_L) + \Delta(C_L)]/3, \tag{17}$$

Then, the followers in the same set share their initial payments and compete with one another in a noncooperative game model G to select the optimal payment to be given to the leader in such a way to maximize their own utilities. This forms a non-cooperative payment selection game $G = \langle S, \{P_F\}, \{U_F(.)\} \rangle$.

Definition 5. *A Payment Selection Game is*

$G = \langle S, \{P_F\}, \{U_F(.)\} \rangle$, *where:*

- S denotes the set of followers pre-selected by the leader (i.e., the players of the game).
- P_F is the strategy set available for each follower F (i.e., the payment).
- $U_F(.)$ represents the utility function of the follower F.

Informally, each follower $F \in S$ selects its strategy from the strategy set P_F with the aim of maximizing its utility function $U_F(P_F, P_{-F})$, where P_{-F} represent the strategies selected by all other players in S except for F.

Definition 6. *A Payment vector $p = (p_1, ..., p_k)$ is a Nash equilibrium of $G = \langle S, \{P_F\}, \{U_F(.)\} \rangle$ if, for every $F \in S$, $U_F(p_F, p_{-F}) \geq U_F(p'_F, p_{-F})$ for all p'_F available for F, where:*

- $p_{-F} = (p_1, ..., p_{F-1}, p_{F+1}, ..., p_n)$*, i.e., the payment vector profile P without follower F's payment,*
- $(p'_F, p_{-F}) = (p_1, ..., p_{F-1}, p'_F, p_{F+1}, ..., p_n)$,
- $U_F(p_F, p_{-F})$ *is the resulting payment for the follower F given the other followers' payment selection result p_{-F}.*

The strategy space is defined to be $P = [P_F]_{F \in S} : 0 \leq p_F \leq \bar{p}$, where $\bar{p}$ denotes the maximal value of payment that might be made. Obviously, the utility function of the followers defined in (9) is continuous in p_F. Thus, derivatives is the suitable technique to find the best responses of the followers. Thus, the problem can be turned into a problem of proving that the utility function given by (9) is concave down in P_F and then computing P_F when $\frac{\partial U_F}{\partial P_F} = 0$

Theorem 1. $U_F = \frac{P_F}{\sum_{i \in S} P_i}[\Delta(R_L) + \Delta(M_L) + \Delta(C_L)] - P_F$ *is concave down in P_F*

Proof.

$$\frac{\partial U_F}{\partial P_F} = \frac{[\Delta(R_L) + \Delta(M_L) + \Delta(C_L)] \sum_{i \in S, i \neq F} P_i}{(\sum_{i \in S} P_i)^2} - 1 \tag{18}$$

$$\frac{\partial^2 U_F}{\partial^2 P_F} = \frac{-2[\Delta(R_L) + \Delta(M_L) + \Delta(C_L)] \sum_{i \in S, i \neq F} P_i}{(\sum_{i \in S} P_i)^3} < 0 \tag{19}$$

According to Equation (19), the second order derivative of U_F with respect to P_F is always less than 0, which means that U_F is concave down in P_F. Hence, we get that P_F is optimal for U_F when $\frac{\partial U_F}{\partial P_F} = 0$. □

Theorem 2. *The equilibrium of the game G is given by*

$$P_F^* = \sqrt{[\Delta(R_L)+\Delta(M_L)+\Delta(C_L)]\sum_{i\in S, i\neq F} P_i} - \sum_{i\in S, i\neq F} P_i \tag{20}$$

Proof.

$$\frac{\partial U_F}{\partial P_F} = 0 \Rightarrow \frac{[\Delta(R_L)+\Delta(M_L)+\Delta(C_L)]\sum_{i\in S, i\neq F} P_i}{(\sum_{i\in S} P_i)^2} - 1 = 0$$

$$\Rightarrow \frac{[\Delta(R_L)+\Delta(M_L)+\Delta(C_L)]\sum_{i\in S, i\neq F} P_i}{(\sum_{i\in S} P_i)^2} = 1$$

$$\Rightarrow (\sum_{i\in S} P_i)^2 = [\Delta(R_L)+\Delta(M_L)+\Delta(C_L)]\sum_{i\in S, i\neq F} P_i$$

$$\Rightarrow \sum_{i\in S} P_i = \sqrt{[\Delta(R_L)+\Delta(M_L)+\Delta(C_L)]\sum_{i\in S, i\neq F} P_i}$$

$$\Rightarrow P_F + \sum_{i\in S, i\neq F} P_i = \sqrt{[\Delta(R_L)+\Delta(M_L)+\Delta(C_L)]\sum_{i\in S, i\neq F} P_i}$$

$$\Rightarrow P_F^* = \sqrt{[\Delta(R_L)+\Delta(M_L)+\Delta(C_L)]\sum_{i\in S, i\neq F} P_i} - \sum_{i\in S, i\neq F} P_i$$

□

Taking into account the boundary constraints $[0, \bar{p}]$ of the payment value, the equilibrium of the followers' payment game G can be rewritten as in Equation (21).

$$P_F^* = \begin{cases} 0, & \text{if } \sqrt{[\Delta(R)+\Delta(M)+\Delta(C)]\sum_{i\in S, i\neq F} P_i} \leq \sum_{i\in S, i\neq F} P_i. \\ \sqrt{[\Delta(R)+\Delta(M)+\Delta(C)]\sum_{i\in S, i\neq F} P_i} - \sum_{i\in S, i\neq F} P_i, & \text{if } \sqrt{[\Delta(R)+\Delta(M)+\Delta(C)]\sum_{i\in S, i\neq F} P_i} \geq \sum_{i\in S, i\neq F} P_i. \\ \bar{p}, & \text{if } \sqrt{[\Delta(R)+\Delta(M)+\Delta(C)]\sum_{i\in S, i\neq F} P_i} - \sum_{i\in S, i\neq F} P_i > \bar{p} \end{cases} \tag{21}$$

The algorithm that describes the followers payment selection game is presented in Algorithm 1. Clearly, the algorithm has a linear complexity in the number of preselected set of followers (i.e., O($|S|$)).

Algorithm 1: Followers Payment Selection Game

Input: Pre-selected set of followers S
Input: Leader's reputation R_L
Input: Leader's market share M_L
Input: Leader's capacity C_L
Output: Optimal payment for followers P_F^*
procedure FOLLOWERSPHASE
    for each follower $F \in S$ **do**
        Compute $\Delta(R_L)$ according to Equation (10)
        Compute $\Delta(M_L)$ according to Equation (11)
        Compute $\Delta(C_L)$ according to Equation (12)
        Compute initial payment according to Equation (17)
        Get information about the payments of all other followers in S
        Compute P_F^* according to Equation (20)
    end for
end procedure

4.1.5 Leader's Utility Maximization Game

Algorithm 2: Leader's Utility Maximization Game

Input: Quota size $|Q|$
Input: Optimal payment for followers P_F^*
Output: Optimal quota of followers S^*
procedure LEADERPHASE
    Pre-select a set of followers S
    Publish reputation, market share, and capacity to S
    Repeat
        Enumerate all possible combinations C in S so that $|C| = |Q|$
        Compute $\Delta(R_F)$ according to Equation (14)
        Compute $\Delta(M_F)$ according to Equation (15)
        Compute $\Delta(C_F)$ according to Equation (16)
        Compute utility $U_L(C)$ based on P_F^* according to Equation (13)
    Until $C = \arg\max_C U_L(C)$
    $S^* = C$
end procedure

Based on the analytical results of the followers' payment selection game G, the leader optimizes its strategy by selecting the optimal quota S^* amongst S that maximizes its revenue according to (13). Substituting (20) into (13), the utility function of the leader becomes:

$$U_L = \sum_{i \in S} (\Delta(R_i') + \Delta(M_i') + \Delta(C_i') + [\sqrt{(\Delta(R) + \Delta(M) + \Delta(C)) \sum_{i \in S, i \neq F} P_i} - \sum_{i \in S, i \neq F} P_i]) \quad (22)$$

It is worth mentioning that the size of the pre-selected set of followers tends to be limited to those services that are deemed to be first-class services (in terms of reputation, market share, and capacity). This is due to the aggregation functions proposed in Section 4.1.2 that are designed in such a way that demotivates leaders from selecting followers having

dramatically bad parameters. Moreover, the final quota of followers tends also to be small since the leader is aware that the gain and resources will be distributed among all the community members. Consequently, as the community size increases, the share of each member, including the leader, will be decreased. This may motivate some community members to leave the community if they consider that their shares are below expectations. This fact pushes leaders to consider the minimal quota of followers that maximizes their utilities in order to increase their own gains and maintain the stability of their communities. The algorithm of the leader's utility maximization game is given by Algorithm 2.

As for the complexity of Algorithm 2, it is clear that steps (1) and (2) can be performed in polynomial time (i.e., $O(|S|)$). The main complexity lies in step (5), where the leader has to enumerate all the possible combinations of the followers' preselected set based on the quota size. Thus, step (5) is bounded by $O(|S|^{|Q|})$. Steps (6)-(9) can be executed in polynomial time and take $O(|Q|)$ at most. Thus, the performance of the algorithm depends mainly on the quota size $|Q|$. As explained earlier, we argue that leaders tend to minimize the size of the followers' preselected set as a result of the design of the aggregation functions (Equations (4), (6), and (8)) that restrict the choice of the leaders to those followers enjoying high parameters' values. Consequently, the quota size tends also to be small, i.e., $|Q| \leq |S|$. These claims are supported in Section 4.3 by means of simulation experiments.

4.2 Industrial Impact: A Complete Scenario

In this section, we answer the "why" and "how" questions regarding our model by illustrating why it is useful and how it can be practically implemented in real applications of services. Consider a flight booking market consisting of five airline Web services S_1, S_2, S_3, S_4, S_5, and S_6. In this market, S_1 and S_6 are strong enough to take the lead, thanks to their high reputation, market share, and capacity. The fact that these services offer the same functionality (i.e., flight booking) and thus target the same customer community makes them in a continuous competition. However, the providers of these services have another *strategic* choice, which may be more beneficial for them rather than adopting pure competitive strategies. More specifically, the high reputation of the leaders S_1 and S_6 creates high demands for their services, especially during promotions and peak times, in such a way that might make their available resources (e.g., bandwidth, storage space) insufficient enough to cope with such demands. In this situation, these leaders have to choose between (1) dropping or delaying some of the incoming requests and (2) cooperating with other services to increase their power in responding to the requests. Obviously, the first choice is quite costly for such services since it may lead to harming their reputation and market share. For example, if S_1 chooses to delay some of its requests, it may end up

losing a part of its customers and reputation for the benefit of S_6 and vice versa.

On the other hand, follower services (S_2, S_3, S_4, and S_5) are likely to suffer, from time to time, from a lack of the incoming requests, which entails the problem of unused resources. In the same context, these latter services have poor chances to be selected to participate in composition sequences in the presence of stronger services (i.e., the leaders). Therefore, follower services have motivations to cooperate with the other stronger services in order to increase their market share and hence efficiently exploit their unused resources. Thus, both leader and follower services are better off collaborating together in a community-based environment using our proposed model. Customers, in their turn, will benefit from such a collaboration by enjoying high-quality and possibly cheaper services.

Suppose now that S_1 decides to confederate with some other less strong airline services (i.e., S_2, S_3, S_4, and S_5) to form a strong community and gain advantage over the other market leader S_6. The parameters associated with S_1, S_2, S_3, S_4, and S_5 are given in Table 4.1. As a first step, S_1 sets minimum requirements for the services it is willing

Table 4.1: Airline Web services' parameters

Service ID	S_1	S_2	S_3	S_4	S_5
Reputation	0.85	0.68	0.66	0.60	0.2
Market Share	0.35	0.25	0.2	0.18	0.02
Capacity	20	16	14	12	5

to confederate with. Assume for example that these requirements are given respectively for the reputation, market share, and capacity as follows: $\min_{Rep} = 0.6$, $\min_{MS} = 0.15$, and $\min_{Cap} = 10$. According to Table 4.1, only the services S_2, S_3, and S_4 satisfy these requirements. Thus, the pre-selection space of S_1 is restricted to only those three services, i.e., $S = \{S_2, S_3, S_4\}$. S_1 sets also a quota of 2 for the services to be selected during the final phase. Now, S_1 publishes its offer consisting of its own reputation, market share, and capacity, i.e., *O:{Rep=0.85, MS=0.35, Cap=20}*, to the services in S. Each service in S computes the variations in its parameters under the published offer and computes an initial payment to be given to the leader. To do so, it has to determine first the aggregate parameters that would result from the possible confederation with the leader S_1. The aggregate reputation score after the confederation with S_1 is calculated according to Equation (5) as follows:

- $f(R_{S_1}, R_{S_2}) = 0.68^{0.17} = 0.94$
- $f(R_{S_1}, R_{S_3}) = 0.66^{0.19} = 0.92$
- $f(R_{S_1}, R_{S_4}) = 0.6^{0.25} = 0.882$

Similarly, the followers compute their new aggregate market shares according to Equation (7):

- $g(MS_{S_1}, MS_{S_2}) = (0.25^{0.1})/2 = 0.43$
- $g(MS_{S_1}, MS_{S_3}) = (0.2^{0.15})/2 = 0.39$
- $g(MS_{S_1}, MS_{S_4}) = (0.18^{0.17})/2 = 0.37$

The aggregate capacities are calculated according to Equation (8) and are given as follows:

- $h(C_{S_1}, C_{S_2}) = 20 + 16 = 36$
- $h(C_{S_1}, C_{S_3}) = 20 + 14 = 34$
- $h(C_{S_1}, C_{S_4}) = 20 + 12 = 32$

Thereafter, followers compute the variations in their parameters based on the obtained aggregate functions. For the reputation (Equation (10)):

- $\Delta(R_{S_2}) = 0.94 - 0.68 = 0.26 = 26\%$
- $\Delta(R_{S_3}) = 0.92 - 0.66 = 0.26 = 26\%$
- $\Delta(R_{S_4}) = 0.88 - 0.6 = 0.28 = 28\%$

For the market share (Equation (11)):

- $\Delta(MS_{S_2}) = 0.43 - 0.25 = 0.18 = 18\%$
- $\Delta(MS_{S_3}) = 0.39 - 0.2 = 0.19 = 19\%$
- $\Delta(MS_{S_4}) = 0.37 - 0.18 = 0.19 = 19\%$

For the capacity (Equation (12)):

- $\Delta(C_{S_2}) = 36 - 16 = 20\%$
- $\Delta(C_{S_3}) = 34 - 14 = 18\%$
- $\Delta(C_{S_4}) = 32 - 12 = 20\%$

Based on the computed variations, each follower service decides about an initial payment for the leader consisting of the unitary variation percentage in the former's parameters according to Equation (17). The calculations are described in the following:

- $InitialPayment(S_2) = (26+18+20)/3 = 21.33$
- $InitialPayment(S_3) = (26+19+18)/3 = 21$
- $InitialPayment(S_4) = (28+19+20)/3 = 32.33$

Thereafter, the services in S exchange their initial payments and each follower computes its optimal payment. The optimal payment is calculated according to Equation (21) as follows:

- $P^*_{S_2} = \sqrt{(26+18+20) \times 53.33} - 53.33 = 5.09$
- $P^*_{S_3} = \sqrt{(26+19+18) \times 53.66} - 53.66 = 4.48$
- $P^*_{S_4} = \sqrt{(28+19+20) \times 42.33} - 42.33 = 10.92$

The followers compute now their utilities according to Equation (9) as follows:

- $U_{S_2} = 5.09/20.49 \times (26+18+20) - 5.09 = 10.81$
- $U_{S_3} = 4.48/16.01 \times (26+19+18) - 4.48 = 13.15$
- $U_{S_4} = 10.92/9.57 \times (28+19+20) - 10.92 = 65.53$

Moving to the leader's side, S_1 computes its new aggregate parameters after a possible confederation with each of the services in S. For the reputation (Equation (14)):

- After confederating with S_2, $\Delta(R_{S_1}) = 0.94 - 0.85 = 0.09 = 9\%$
- After confederating with S_3, $\Delta(R_{S_1}) = 0.92 - 0.85 = 0.07 = 7\%$
- After confederating with S_4, $\Delta(R_{S_1}) = 0.88 - 0.85 = 0.03 = 3\%$

For the market share (Equation (15)):

- After confederating with S_2, $\Delta(MS_{S_1}) = 0.43 - 0.35 = 0.08 = 8\%$
- After confederating with S_3, $\Delta(MS_{S_1}) = 0.39 - 0.35 = 0.04 = 4\%$
- After confederating with S_4, $\Delta(MS_{S_1}) = 0.37 - 0.35 = 0.02 = 2\%$

For the capacity (Equation (16)):

- After confederating with S_2, $\Delta(C_{S_1}) = 36 - 20 = 16\%$
- After confederating with S_3, $\Delta(C_{S_1}) = 34 - 20 = 14\%$

- After confederating with S_4, $\Delta(C_{S_1}) = 32 - 20 = 12\%$

Next, the leader computes its utility for each possible combination C of two (quota size) among the three pre-selected services according to Equation (13). The calculations go as follows:

- $C = \{S_2, S_3\}$, $U_L = (9+8+16+5.09+7+4+14+4.48) = 67.57$
- $C = \{S_2, S_4\}$, $U_L = (9+8+16+5.09+3+2+12+10.92) = 66.01$
- $C = \{S_3, S_4\}$, $U_L = (7+4+14+4.48+3+2+12+10.92) = 57.4$

Thus, the set $s^* = \{S_2, S_3\}$ is chosen by the leader S_1 since it gives the maximal utility value compared to the other combinations. To show that increasing the size of the community is not always the best choice for the leader, we calculate the utility of the leader S_1 if it decides to add S_5 to the selected set, knowing that S_5 has dramatically low parameters. Following the principles of calculations described above, the optimal payment of S_5 to be given to the leader S_1 will be $P_5^* = 0$ (according to the first constraint of Equation (21)). Consequently, the utility of the leader after adding S_5 will decrease considerably from $U_L = 67.57$ to $U_L = 14.57$ (as its reputation, market share, and capacity are significantly decreasing without receiving any payment). It is worth mentioning as well that our approach satisfies the *first mover's advantage* property of the Stackelberg game [139], which states that the utility of the leader tends to be greater than that of the followers since laders have the advantage of making the first move. In our example, the utility of the leader $U_{S_1} = 67.57$ is greater than all followers' utilities ($U_{S_2} = 10.81$, $U_{S_3} = 13.15$, and $U_{S_4} = 65.53$).

4.3 Experimentations and Empirical Analysis

4.3.1 Experimental Setup

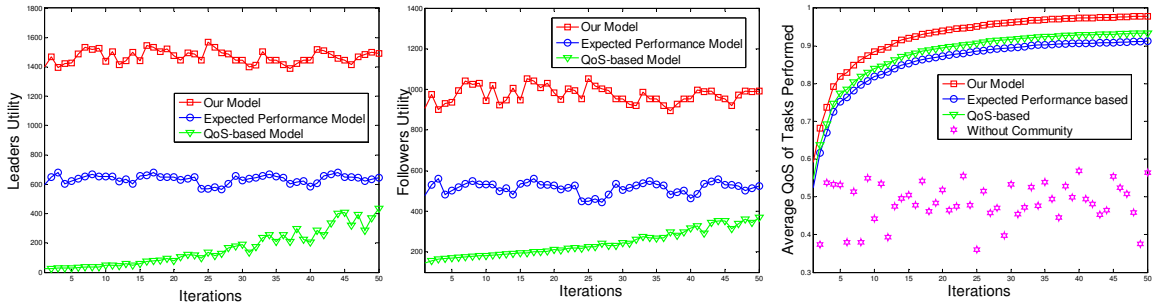
In this section, we provide empirical results to validate the theoretical proofs discussed in the previous section. The objective is to study the satisfaction of the intelligent service agents in terms of utility and reputation as well as the satisfaction of the users in terms of QoS provided to their requests. The simulation application is written in C# using Visual Studio. The information related to services is populated from a real-life dataset that includes 2507 real services operating on the Web [13]. The topic of flight booking has been used for the simulations. The dataset is based on a scenario in which users send XML-based requests containing the flight dates, origin and destination, number of seats, and type of tickets (i.e., one-way or return) and receive an XML-based response consisting of different

flights hosted by different companies along with the related information such as prices, timing, and so on. $200,000$ flights are collected and stored in the dataset that records the values of nine QoS metrics, namely throughput, availability, reliability, response time, successability, compliance, latency, accessibility, and cost [13]. The reputation is computed by aggregating the different QoS metrics using the concept of Web Service Relevancy Function (WsRF) presented in [12] to rank Web services based on their QoS metrics. The basic idea is to represent the QoS parameters of Web services as a matrix called WsRF matrix in which each row represents a single Web service and each column represents a single QoS parameter. Based on this matrix, QoS parameters are normalized by comparing each element in the WsRF matrix against the maximum QoS value in the corresponding column. Having normalized the QoS parameters, the WsRF value for each service can now be computed by summing up all the normalized QoS parameters for that service. The reputation score is then obtained by normalizing the WsRF value to a number between 0 and 1. Web services are divided into leaders and followers in the simulations based on their reputation score, market share, and capacity of fulfilling requests.

To verify the effectiveness of our model, we compare it against two other models used as a benchmark, which we call "expected performance model" and "QoS-based model". In the QoS-based model, the leader accounts only for the QoS values of the followers while making selections of the community members. For the expected performance model, the approach presented in [59] is used and updated slightly to fit our scenario. In fact, this approach uses a one-stage non-cooperative game theoretical model to derive a threshold according to which the master (or leader) of the community decides whether to accept a service to be part of its community or not. The master compares the expected performance of the community after the joining of a certain service with the actual performance while considering a risk factor and decides to accept this service only in case the expected performance exceeds the actual performance. The risk factor indicates how flexible the master is in loosing some performance. For example, if the risk factor associated with a certain master is 20%, then it would consider any situation in its strategy analysis where estimated performance is more than 80% of the community's current performance. Thus, a master m will accept the joining if $\overline{E_m} > (1 - S_m)E_m$, where $\overline{E_m}$ denotes the expected performance of m, E_m denotes its actual performance, and S_m is the associated risk factor. The performance of a certain Web service x is calculated based on its reputation R_x, market share M_x, capacity C_x, and number of received requests Rq_x according to Equation (23).

$$E_x = \frac{R_x \times M_x}{|Rq_x - C_x| + 1} \tag{23}$$

This approach has been slightly modified to fit our community formation scenario since it originally tackles the problem of Web services joining preexisting communities (not a



(a) Leaders' Utility *(b) Followers' Utility* *(c) QoS provided to users' tasks*

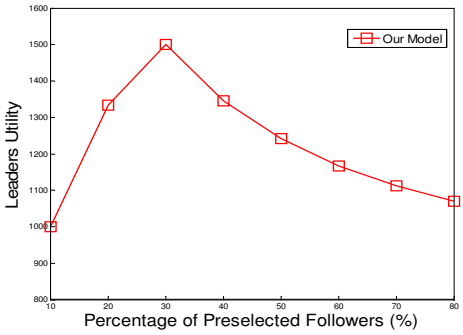
Figure 4.1: Satisfaction of leaders, followers, and users respectively

distributed formation model). In the modified scenario, the leader compares its expected performance after confederating with each service in the pre-selected set and considers this service in the final selection if the expected performance exceeds the actual performance. The risk factor considered in the simulations is 50% [59].

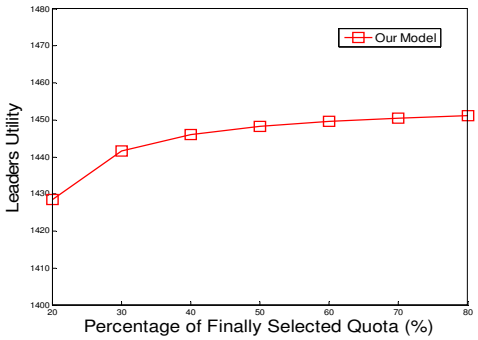
4.3.2 Experimental Results

We begin by measuring the satisfaction of the Web service agents. The simulations run for 50 iterations. At each iteration, the set of leaders and followers is changed and the average utility for both leaders and followers is computed. The final selection of followers (i.e., quota) considered by the leaders represents 35% of the pre-selected followers set. Figures 4.1a and 4.1b describe the average gain of leaders and followers respectively according to our model and the two aforementioned models. As shown in the Figures, both leaders and followers can achieve higher utilities by using our model. This is due to the fact that the game in our model is played sequentially at two stages, which makes both leaders and followers aware of each other's strategies and enables them hence to play their best responses that maximize their utility. In the expected performance model, the game is played one-shot so that players are playing simultaneously, which limits their learning space and introduces some uncertainty in their decisions. For the QoS-based model, the decision is done according to one particular metric, which makes the selection biased towards one exclusive parameter without considering other important metrics. For example, leaders may choose to confederate with followers enjoying low response time but having poor market shares and/or capacities, which negatively affects their utility.

We move now to measuring the satisfaction of users by studying the average QoS provided to their requests. After communities are formed, we generated 2500 random requests initiated by 1000 consumer agents to investigate how efficient are the formed



(a) Impact of preselection size



(b) Impact of quota size

Figure 4.2: Impact of the preselection set size and quota size on the leaders utility

communities in fulfilling users' requests. Each request is modeled as a class that is characterized by the QoS value needed to perform this task and the QoS actually provided to the task. First, we evaluate the average QoS provided to the users' tasks according to the different aforementioned models while considering an additional scenario, called "Without Community". This latter scenario represents the case where Web services respond to the requests individually without being organized into communities. Figure 4.1c shows that the community-based models (our model, expected performance, and QoS-based models) are able to increase and stabilize the QoS provided to users' requests. This is justified by the fact that the community guarantees higher performance (i.e., availability, response time, and so on) as a result of the cooperation and interoperability that takes place among community members. Moreover, Figure 4.1c reveals that our model outperforms the expected performance and QoS-based models in terms of QoS. This is due to the fact that our model enables services to maximize their utilities upon forming communities, which makes the community stable and allows it hence to respond to the users' requests with better performance. In contrast, although the QoS-based model selects the Web services based on their QoS values, the services in this model may be encouraged to leave their communities and join other communities if they find better utility, which makes the community unstable and affects hence its performance in fulfilling requests. The same intuition applies for the expected performance model, where the services may be encouraged to leave their communities.

In order to study the impact of quota size and preselected followers' set size on the leader's satisfaction, we vary these sizes and show how such variations influence the utility of the leaders. For example, a 10% percentage of preselected followers means that 10% of the follower services are preselected by the leader for possible confederation. Similarly, a quota percentage of 20% means that 20% of the preselected followers are considered

for final confederation by the leader. As depicted in Figure 4.2a, the utility of the leaders increases initially with the increase in the size of preselected followers and begins to decrease at a certain point. This is due to the fact that as the size of preselected set continues to increase, the possibility of selecting services having drastically bad parameters increases also, which decreases the leader's utility as a result of the aggregation functions described in Section 5.3. Therefore, leaders tend to restrict this set as much as possible to the top Web services in terms of reputation, market share, and capacity. Thus, we can conclude that Algorithm 2 is computationally tractable. Moreover, Figure 4.2b reveals that increasing the quota size results in a continuous increase in the utility of leaders since the leader is selecting from the preselected set, which is already filtered based on the Web services parameters. However, this does not mean that increasing the quota size is always the rational choice for leaders. In fact, the leader is aware that increasing the community size will have some negative side effects on the satisfaction of its community members as well as on its satisfaction since the gain and resources are distributed per member and thus increasing the community size would reduce the share of the community members, including the leader itself.

4.4 Conclusion

In this chapter, we investigated the problem of forming communities among services deployed in cloud centers having uneven business capabilities by proposing a Stackelberg game theoretical model whose players are leader and follower market services. Our solution enjoys three main advantages over the state-of-the art: (1) it considers a fully distributed environment, where all the services are completely autonomous in their decisions; (2) the community formation scenario is inspired by the real-world business context and the business-related objectives of the service forming communities are clearly defined; and (3) a two-stage sequential Stackelberg game model is used to ensure the formation of optimal and stable communities in the long-term. As confirmed by the simulation results, the proposed model is able to increase the utility and hence the satisfaction of both leader and follower services up to $\approx 20\%$ compared to the "expected performance" and "QoS-based" models, which increases the stability of the formed communities. Moreover, our solution allows communities to achieve high levels of QoS while performing users' requests, which increases the satisfaction of the users as well. Thus, we have achieved our first research objective (**Objective** 1) discussed in Chapter 1, which aimed at enabling the formation of communities comprising services belonging to clouds having uneven business capabilities.

A major challenge against this model is the existence of malicious services; particularly those that cheat about their actual business parameters to guarantee being structured in

strong communities and/or those that renege on their agreements with the other services after communities are formed. Such services are often called *passive malicious services* [150] since their objective is to illegally maximize their own profits rather than harming other services. To tackle this challenge, we propose in the next chapter a trust-based multi-cloud community formation model whose goal is to minimize the number of passive malicious services in the formed communities.

Chapter 5

Towards Trustworthy Multi-Cloud Services Communities

The prominence of cloud computing has led to unprecedented proliferation in the number of services deployed in cloud data centers. In parallel, service communities have gained recently increasing interest due to their ability to facilitate discovery, composition, and resource scaling in large-scale services' markets. The problem is that the traditional community formation models may work well when all services reside in a single cloud but cannot support a multi-cloud environment. Particularly, these models overlook the problem of encountering malicious services that misbehave to illegally maximize their benefits, which arises from grouping together services owned by different providers. Besides, they rely on a centralized architecture whereby a central entity regulates the community formation, which contradicts with the distributed nature of cloud-based services. In this chapter, we aim to address those shortcomings by putting forward a trust-based multi-cloud services community formation model. We start by describing our proposed trust framework called DEBT (Discovery, Establishment, and Bootstrapping Trust) that allows services to establish credible trust relationships in the presence of collusion attacks. Thereafter, we model the problem of forming trusted multi-cloud communities as a hedonic coalitional game [26], propose the appropriate preference function and coalition formation algorithm, and analyse the properties of the game[1].

[1] The content of this chapter is published in [152]

5.1 System Model and Assumptions

5.1.1 System Model

Let $S = \{S_1, ..., S_n\}$ be a finite set of services, where each service $S_i \in S$ is characterized by a set of available resources $R = \{R_1, ..., R_n\}$ (e.g., amount of memory, number of cores, etc.). Let $G = (S, E, J)$ be a directed social network graph representing the relationships between these services, where each edge $(S_i, S_j) \in E$ indicates an interaction (e.g., participation in the same composition sequence) between service S_i and service S_j. Thus, if $(S_i, S_j) \notin E$ then services S_i and S_j had no interaction in common. Each edge (S_i, S_j) is associated with a judgement $J(S_i, S_j) \neq J(S_j, S_i) \in \{T, M\}$ that denotes each service's judgment on any other service based on their previous interactions. For example, the judgment pair $(S_1 \rightarrow S_2 : T, S_2 \rightarrow S_1 : M)$ between services S_1 and S_2 in Fig. 5.1 indicates that S_1 rates S_2 as trustworthy, whereas S_2 rates S_1 as (passive) malicious (i.e., $J(S_1, S_2) = T$ and $J(S_2, S_1) = M$). To decide about this judgement, services use the following well-known satisfaction metric [86]:

$$J(S_i, S_j) = \frac{Sat(S_i, S_j)}{Tot(S_i, S_j)}, \tag{24}$$

where $Sat(S_i, S_j)$ denotes the number of interactions between S_i and S_j that S_i considers as satisfactory and $Tot(S_i, S_j)$ denotes the total number of interactions between S_i and S_j.

This metric allows us to express the satisfaction between services from both performance and security perspectives. In particular, the transactions that S_i would consider satisfactory with S_j are those in which S_j performed well in terms of QoS metrics and behaved well from the security perspective. Based on Equation (24), if $J(S_i, S_j) > \beta$ then S_i rates S_j as trustworthy, where β is a threshold that is set by each service depending on the type of the other service(s) being evaluated. For example, the interactions with an online payment service would be weighted higher than those of a weather forecasting service. Otherwise, S_j would be rated as malicious. As mentioned earlier, the objective is to form trusted multi-cloud services communities between services geographically distributed across multiple cloud data centers using a distributed trust model. The trust towards a certain service S_j is built by collecting judgments on this service from its direct neighbors $N(S_j)$ in G (i.e., the services that had interacted with S_j). To this end, each service S_i holds a fixed number of inquiries it is allowed to make and denoted by $Inq(S_i)$. The motivation behind this assumption is to motivate the participation of the services in the trust establishment process through linking the number of inquiries that they are allowed to make with the degree of their participation in the trust framework. Such an assumption is realistic since we are considering *selfish* services in the sense that a service will respond to a certain inquiry coming from another service only if the latter has previously responded the former's

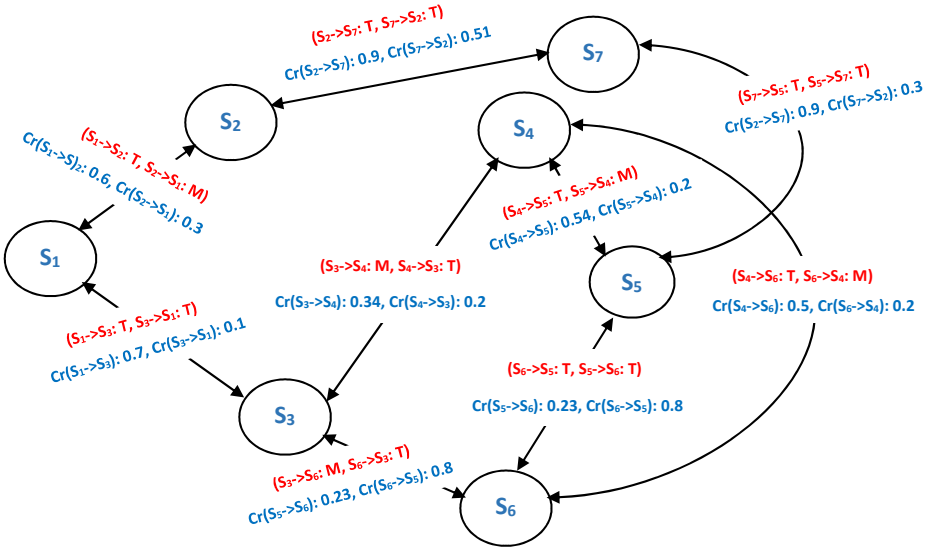


Figure 5.1: Social network graph: Vertices represent services and edges represent the interactions among services

inquiries. Initially, all services have an equal amount of this metric, which is updated later during the trust establishment process (See Section 5.2). Since services may be either *truthful* or *collusive* in judging the other services [120], each pair of services $(S_i, S_j) \in S$ has a belief in credibility $(Cr(S_i \rightarrow S_j) = n, Cr(S_j \rightarrow S_i) = m)$ that represents each service's accuracy level in judging the other services, where n and m are two decimal numbers. Based on the collected judgments, each service S_i builds a belief in trustworthiness denoted by $belief_{S_i}^{S_j}(T)$ and a belief in maliciousness denoted by $belief_{S_i}^{S_j}(M)$ for any other service S_j it is interested in forming community with. It's worth noting that such a mechanism does not entail any privacy breach as only the value of $J(S_i, S_j)$ is shared between services without revealing any sensitive information such as the volume or type of interactions.

The community formation is modeled as a hedonic coalition formation game [26], where each coalition $C \subseteq S$ represents a given community[2]. Let Π denote the set that partitions services S into non-empty coalitions and that is referred to as a *coalition structure* [26].

Definition 7 (**Coalition Structure).** *A coalition structure or partition is a set of coalitions* $\Pi = \{C_1, ..., C_S\}$ *that splits the set of services* S *into disjoint coalitions such that* $\forall\, l \neq l', C_l \cap C_l' = \emptyset$ *and* $\bigcup_{k=1}^{l} C_k = S$. *The coalition to which service* S_i *belongs is denoted by* $C_l^{S_i}$.

Let $U_{S_i}(C_k)$ denote the utility of service S_i in a certain coalition $C_k \in \Pi$. $U_{S_i}(C_k)$ is obtained by summing up S_i's beliefs in trustworthiness in C_k's members. Thus, $U_{S_i}(C_k)$ is computed

[2]In the rest of this chapter, the terms *coalition* and *community* are used interchangeably.

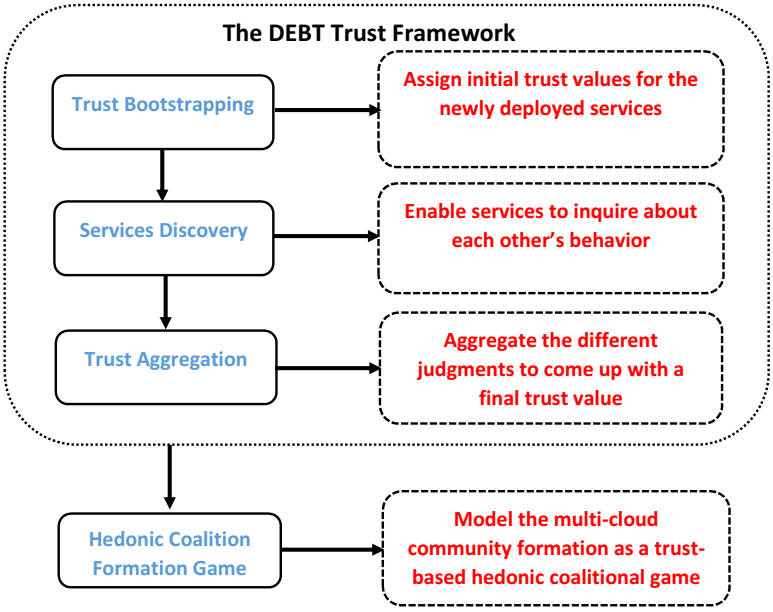


Figure 5.2: Methodology of the trust-based multi-cloud services communities model

as follows:

$$U_{S_i}(C_k) = \sum_{S_j \in C_k} belief_{S_i}^{S_j}(T) \tag{25}$$

The methodology followed in the rest of this chapter is depicted in Fig. 5.2.

5.1.2 Attack Model and Assumptions

Attacks may occur in our model either (1) during trust establishment and/or (2) during and after communities formation. During trust establishment, collusion attacks may take place between services to mislead the results. During and after communities formation, passive malicious services might misbehave to save their resources and gain illegal advantage over the other services. At this stage, we consider only passive attacks in the sense that the malicious services considered during and after communities formation are assumed to misbehave in order to increase their own benefits without having a direct intention to harm the other services and/or communities. Therefore, active attacks such as Sinkhole and Denial of Service (DoS) are beyond the scope of this chapter. In particular, we consider the two following types of attacks:

- **Collusion Attacks:** Such attacks occur when several malicious services collaborate together to give misleading judgments either to increase the trust score of some

services (i.e., a promoting attack) or to decrease the trust score of some other services (i.e., a slandering attack). Note that these types of attacks cannot occur in a non-collusive way in the sense that a single malicious service cannot submit multiple misleading judgements to conduct a promoting attack and/or slandering attack as judgments are given upon request in our trust framework.

- **Passive Attacks:** Such attacks occur when passive malicious services cheat about their available resources and/or QoS potential during communities' formation in order to increase their chances of being grouped into powerful communities. After communities are formed, these malicious services would renege on their agreements with both services and clients by benefiting from the other services' resources (e.g., physical computing infrastructure) and refraining from sharing their own resources to dedicate them for their own workload.

5.2 The DEBT Trust Framework

In this section, we present the details of our proposed trust framework.

5.2.1 Service Discovery

In order to establish trust between services in the social network (Fig. 5.1), judgments should be collected first. Therefore, we propose a discovery algorithm that allows services to inquire about each other from their direct neighbors (i.e., the services that had dealt with). The proposed algorithm capitalizes on the concept of *tagging* in online social networks (e.g., Facebook, LinkedIn) to explore the direct neighbors of a certain service. The basic idea is to keep tagging or nominating intermediate services until identifying all the reachable direct neighbors of the intended service. Let us consider the case of service s that asks service x about its judgment on service d. If x has a direct interaction (i.e., edge) with d, it reports its judgment directly to s. Otherwise, it tags its neighbors that had such an interaction (if any) or those that may introduce s to some other services that had such an interaction with d. The details are explained in Algorithm 3. The inputs of the algorithm are a source node src (inquiring service) and a destination node d (inquired about service) in a social network graph (lines 1-2). The output is the set of all reachable direct neighbors of d (line 3). The algorithm creates an empty tag instance (line 6) and loops over the direct neighbors of the source node one by one and marks them as explored (lines $7-8$). This has the advantage of avoiding the revisit of any already visited node. Each explored neighbor gets appended to the tag instance (lines 9-12). If the neighbor has a

Algorithm 3: Services Discovery Algorithm

```
1: Input: Source node src
2: Input: Destination node d
3: Output: Set of direct neighbors of d, N(d).
4: procedure SERVICEDISCOVERY
5:     s = src
6:     Create an empty tag instance t = ⟨⟩
7:     for each not explored node y ∈ N(s) do
8:         Mark y as explored
9:         if s ∉ t
10:            Append s to t
11:        end if
12:        Append y to t
13:        if (y,d) ∈ E then
14:            send t to src
15:            empty the tag instance
16:        else
17:            s = y
18:        end if
19:        if all y ∈ N(S) are explored then
20:            s = src
21:        end if
22:    end for
23:    Append the last element of the tag instance to N(d)
24:    return N(d)
25: end procedure
```

direct edge with the destination service d, then the tag instance is directly returned to the source node and the tag instance is emptied to start a new tagging process (lines 13-15). Otherwise, the algorithm loops recursively over the "neighbors of the neighbors" and adds them to the tag instance in case they have a direct edge with d. This is done by recursively assigning each neighbor with the role of the source node (line 17). This process stops at the level of a node that reports either a judgement to the source node or reports neither a judgement nor a tag instance. Thus, each tag instance can be considered as a sequence of services $\langle S_i, S_{i+1}, ..., S_n \rangle$ starting from a direct neighbor S_i of a certain requestor service S_0 and leading to one or more neighbor(s) (i.e., S_{i+1},..., S_n) of a destination service S_{n+1}.

As for the complexity of Algorithm 3, the algorithm is a variation of the Breadth-First Search (BFS) strategy [169] in graph theory. Therefore, the computational complexity of Algorithm 3 is $O(|S| + |E|)$ since the worst case would be when all the vertices and edges have to be visited exactly once, where $|S|$ is the number of services in the social network graph and $|E|$ denotes the number of interactions between them. The storage overhead of the algorithm is $O(|S|)$ since the worst case occurs when all the services would need to be

stored in the tag instance.

5.2.2 Trust Establishment

Having discussed the discovery algorithm, the next step is to establish the trust relationships between services. As mentioned earlier, trust is constructed by collecting judgments about services based on their previous interactions. The power of such a mechanism stems from its ability to produce meaningful judgements by considering multiple parties' opinions. Nonetheless, several challenges are encountered by such a mechanism in real-world scenarios [141]. Practically, services may be tempted to engage in some collusion scenarios and provide dishonest judgments, which leads to misleading trust results. Moreover, these services usually tend to refrain from revealing their opinions since they do not have incentives for doing so, which leads to meaningless or biased computations of the aggregate trust value. To tackle these problems, we propose (1) an aggregation model for the collected judgments that is able to overcome the collusion attacks even when attackers are the majority, and (2) an incentive model to motivate services to participate in the trust establishment process. That is, the aggregation technique should take into account the existence of colluding services. Therefore, simplistic combination techniques such as averaging and majority voting are unsuitable for the considered problem. To address this challenge, we propose an aggregation technique based on the Dempster-Shafer theory of evidence. Dempster-Shafer [150] is a mathematical theory that combines evidences from independent sources to come up with a degree of belief regarding a certain hypothesis. Dempster-Shafer is well-suited for the considered problem for two main reasons [29]: (1) Dempster-Shafer can represent uncertainty or lack of complete knowledge, and (2) it provides a powerful rule for combining observations from multiple (possibly unreliable) parties. The first property is important to guarantee the fairness in the trust aggregation process as some services may misbehave due to some out of control circumstances (e.g., problems in the physical infrastructure they are hosted on) and not as a result of some malicious behavior. For example, in Dempster-Shafer, if a service A confirms that service B is trustworthy with probability p, it does not follow that B is malicious with probability $1 - p$ as is the case in the Bayesian inference [48]. Hence, A would have p degree of belief in the trustworthiness of B and 0 degree of belief in B's maliciousness. The second property is important to prevent colluding services from misleading the final aggregate trust value. It is worth noting that Dempster-Shafer has been already used for trust establishment in multi-agent systems [165, 55]. The difference between these approaches and our approach is that the latter requires no threshold for deciding whether to trust another agent or not. Besides, we propose in our model a credibility update function and link the credibility scores of the services with the number of inquiries that they are allowed to make with the aim of

encouraging services to participate in the trust establishment process and provide truthful judgments. Moreover, in the referred approaches if no information about the newcomer agents may be obtained then these agents are deemed to have no reputation at all, which may end up overlooking these agents in future community formation processes in the presence of other well-reputable agents. To handle this issue, we propose in the next section a bootstrapping mechanism to assign initial trust values for such services.

The proposed aggregation technique works as follows. Let $\Omega = \{T, M, U\}$ denote a set consisting of three hypotheses. T denotes that a certain service x is trustworthy; M denotes that x is malicious; and U denotes that x is either trustworthy or untrustworthy to express the uncertainty or partial knowledge in the decisions. The basic probability assignment (bpa) of a service S in judging another service S', denoted by $m_S^{S'}$, defines a mapping function of the set Ω to a real-valued interval bounded by 0 and 1, i.e., $m_S^{S'} : \Omega \longmapsto [0,1]$. In our framework, the bpa for a certain hypothesis is equal to the credibility score believed on the service giving the judgement. In other words, suppose that service S is asked by service q to judge another service S', where q has a belief in S's credibility that is equal to α. Assume that S claims that S' is trustworthy, then the bpa's of S would be: $m_S^{S'}(T) = \alpha$, $m_S^{S'}(M) = 0$, and $m_S^{S'}(U) = 1 - \alpha$. On the other hand, if S reports that S' is malicious, then the bpa's of S would be: $m_S^{S'}(T) = 0$, $m_S^{S'}(M) = \alpha$, and $m_S^{S'}(U) = 1 - \alpha$. It is worth noting that, throughout this chapter, when S' is unique or understood from the context, we simplify by writing m_S and when both S and S' are understood from the context, we simply write m. To aggregate the different evidences (i.e., bpa's), a belief function is used. The belief function represents the total bpa's supporting a given hypothesis H and maps H to a real-valued number between 0 and 1. The belief function of service S in service S' regarding a certain hypothesis H (where $H = T, M$, and U respectively) after inquiring two other services 1 and 2 is given as follows:

$$bel_S^{S'}(T) = m_1(T) \oplus m_2(T) = \frac{1}{K}[m_1(T)m_2(T) + m_1(T)m_2(U) + m_1(U)m_2(T)] \tag{26}$$

$$bel_S^{S'}(M) = m_1(M) \oplus m_2(M) = \frac{1}{K}[m_1(M)m_2(M) + m_1(M)m_2(U) + m_1(U)m_2(M)] \tag{27}$$

$$bel_S^{S'}(U) = m_1(U) \oplus m_2(U) = \frac{1}{K}[m_1(U)m_2(U)] \tag{28}$$

where:

$$K = \sum_{h \cap h' = \emptyset} m_1(h)m_2(h') \tag{29}$$

Thus, the problem is turned into computing the beliefs in trustworthiness $bel_S^{S'}(T)$ and maliciousness $bel_S^{S'}(M)$ of service S in service S'.

Theorem 3. *The proposed aggregation technique overcomes collusion attacks even when*

attackers are the majority, if the credibility scores of truthful raters are higher than those of colluding raters.

Proof. For simplicity and without loss of generality, suppose that three services A, B, and C are asked by service S to judge another service D. Assume that D is trustworthy and that services A and C collude together to demote D by claiming the contrary, whereas B is truthful and reports that D is trustworthy. Assume as well that the credibility scores of A, B, and C believed by S are $Cr(S \to A) = \frac{\alpha}{2}$, $Cr(S \to B) = \alpha$, and $Cr(S \to C) = \frac{\alpha}{3}$ respectively such that $Cr(S \to B) > Cr(S \to A) > Cr(S \to C)$. As mentioned earlier, A and C claim unjustly that D is malicious whereas A reports that D is trustworthy. Thus, the bpa's of the three services are given as follows:

- **Service** A: $m_A(T) = 0$, $m_A(M) = \frac{\alpha}{2}$, and $m_A(U) = 1 - \frac{\alpha}{2}$.
- **Service** B: $m_B(T) = \alpha$, $m_B(M) = 0$, and $m_B(U) = 1 - \alpha$.
- **Service** C: $m_C(T) = 0$, $m_C(M) = \frac{\alpha}{3}$, and $m_C(U) = 1 - \frac{\alpha}{3}$.

The theorem may be proved by contradiction. Thus, assuming that the theorem does not hold, we should get that S's belief in D's maliciousness is higher than its belief in D's trustworthiness based on the judgments of the colluding services A and C that form the majority, i.e.,

$$bel_S^D(M) > bel_S^D(T) \qquad (*)$$

Let's start by combining the bpa's of A and B as per Table 5.1.

Table 5.1: Combination of the bpa's of services A and B

A \ B	$m_B(T) = \alpha$	$m_B(M) = 0$	$m_B(U) = 1 - \alpha$
$m_A(T) = 0$	0	0	0
$m_A(M) = \frac{\alpha}{2}$	$\frac{\alpha^2}{2}$	0	$\frac{\alpha - \alpha^2}{2}$
$m_A(U) = 1 - \frac{\alpha}{2}$	$\frac{2\alpha - \alpha^2}{2}$	0	$\frac{\alpha^2 - 3\alpha + 2}{2}$

Note that the cell values in Table 5.1 are obtained by multiplying the corresponding rows and columns. Now, let's compute the combined beliefs of services A and B.

- $K = m_A(T)m_B(T) + m_A(T)m_B(U) + m_A(U)m_B(T) + m_A(M)m_B(M) + m_A(M)m_B(U) + m_A(U)m_B(M) + m_A(U)m_B(U) = \frac{-\alpha^2+2}{2}$.

  We abuse the notation of the function m and define $m_{AB}(T)$, $m_{AB}(M)$, and $m_{AB}(U)$ as follows:

- $m_{AB}(T) = m_A(T) \oplus m_B(T) = 1/K[m_A(T)m_B(T) + m_A(T)m_B(U) + m_A(U)m_B(T)] = \frac{-2\alpha^2+4\alpha}{-2\alpha^2+4}$.
- $m_{AB}(M) = m_A(M) \oplus m_B(M) = 1/K[m_A(M)m_B(M) + m_A(M)m_B(U) + m_A(U)m_B(M)] = \frac{-2\alpha^2+2\alpha}{-2\alpha^2+4}$.
- $m_{AB}(U) = m_A(U) \oplus m_B(U) = 1/K[m_A(U)m_B(U)] = \frac{2\alpha^2-6\alpha+4}{-2\alpha^2+4}$.

Table 5.2: Combination of services A and B's belief with the bpa of C

AB \ C	$m_C(T) = 0$	$m_C(M) = \frac{\alpha}{3}$	$m_C(U) = 1 - \frac{\alpha}{3}$
$m_{AB}(T) = \frac{-2\alpha^2+4\alpha}{-2\alpha^2+4}$	0	$\frac{-2\alpha^3+4\alpha^2}{-6\alpha^2+12}$	$\frac{2\alpha^3-10\alpha^2+12\alpha}{-6\alpha^2+12}$
$m_{AB}(M) = \frac{-2\alpha^2+2\alpha}{-2\alpha^2+4}$	0	$\frac{-2\alpha^3+2\alpha^2}{-6\alpha^2+12}$	$\frac{2\alpha^3-8\alpha^2+6\alpha}{-6\alpha^2+12}$
$m_{AB}(U) = \frac{2\alpha^2-6\alpha+4}{-2\alpha^2+4}$	0	$\frac{2\alpha^3-6\alpha^2+4\alpha}{-6\alpha^2+12}$	$\frac{-2\alpha^3+12\alpha^2-22\alpha+12}{-6\alpha^2+12}$

Then, we combine in Table 5.2 the combined belief of services A and B with the bpa of service C. By computing the combined beliefs of services A, B, and C, we get:

- $K = \frac{2\alpha^3-10\alpha^2+12}{-6\alpha^2+12}$,
- $m_{AB}(T) \oplus m_C(T) = \frac{-12\alpha^5+60\alpha^4-48\alpha^3-120\alpha^2+144\alpha}{-12\alpha^5+60\alpha^4+24\alpha^3-192\alpha^2+144}$.
- $m_{AB}(M) \oplus m_C(M) = \frac{-12\alpha^5+72\alpha^4-36\alpha^3-144\alpha^2+120\alpha}{-12\alpha^5+60\alpha^4+24\alpha^3-192\alpha^2+144}$,
- $m_{AB}(U) \oplus m_C(U) = \frac{12\alpha^5-72\alpha^4+108\alpha^3+72\alpha^2-264\alpha+144}{-12\alpha^5+60\alpha^4+24\alpha^3-192\alpha^2+144}$.

We have that $\sum_{x\in\{T,M,U\}} m_{AB}(x) \oplus m_C(x) = 1$ and $m_{AB}(T) \oplus m_C(T) > m_{AB}(M) \oplus m_C(M)$ for every $0 < \alpha \leq 1$; meaning that $bel_S^D(T) > bel_S^D(M)$ for every $0 < \alpha \leq 1$, which contradicts with (*). Thus, S's belief in D's trustworthiness exceeds its belief in D's maliciousness although the majority of raters (i.e., A and C) colluded to state the contrary. Generally speaking, the proposed aggregation technique overcomes the collusion attacks even when attackers are the majority if and only if (1) the credibility values are between 0 and 1, and (2) the credibility scores of the trustworthy raters are higher ones. Formally, let:

- V_T: denote the set of truthful services participating in the judgement process.
- V_C: denote the set of colluding services participating in the judgement process.
- V: denote the set of services participating in the judgement process, i.e., $V = V_T \cup V_C$.

The proposed aggregation technique overcomes the collusion attacks even when attackers are the majority, i.e., $\mid V_C \mid \geq \mid V_T \mid$ if and only if:

$$\forall v \in V,\ 0 < Cr(v) \leq 1 \tag{30}$$

$$\forall\, t \in V_T \text{ and } c \in V_C,\ Cr(t) > Cr(c) \tag{31}$$

□

Obviously, the performance of the aggregation technique depends heavily on the credibility scores assigned to the services. Thus, maintaining healthy values of this metric is a building block for achieving truthful decisions. Therefore, the credibility metric should be updated continuously to ensure that truthful services always hold higher credibility scores than those of the colluding ones. Equation (32) depicts the credibility update function for each service x after having participated in judging service y in favor of service s.

$$Cr(s \to x) = \begin{cases} \min(1, Cr(s \to x) + |Z - Cr(s \to x)|), & \text{if C 1} \\ \left|Cr(s \to x) - \min(belief_s^y(T), belief_s^y(M))\right|, & \text{if C 2} \end{cases} \tag{32}$$

where $Z = \max(belief_s^y(T), belief_s^y(M))$ and $C1$ and $C2$ are two conditions such that:

C 1. $J(x,y) \in \{T\}\ \ \&\ \ belief_s^y(T) > belief_s^y(M)$ *or* $J(x,y) \in \{M\}\ \ \&\ \ belief_s^y(T) < belief_s^y(M)$

C 2. $J(x,y) \in \{T\}\ \ \&\ \ belief_s^y(T) < belief_s^y(M)$ *or* $J(x,y) \in \{M\}\ \ \&\ \ belief_s^y(T) > belief_s^y(M)$

The intuition behind Equation (32) is that truthful services whose judgments agree with the winner belief receive a reward that is equal to the difference between their current credibility scores and the value of that belief. For the untruthful services whose judgments disagree with the winner belief, they undergo a decrease in their credibility scores that is equal to the value of the loser belief. Finally, services should receive rewards for their participation in the services discovery and trust aggregation processes. This reward is important to motivate further participation from these services. To this end, we link the number of inquiries that a service is allowed to make about other services with its credibility score and the number of tags it has made. The reward function is given by Equation (33), where x is a given service being rewarded by service s for which it has provided judgement, $Inq(x \to s)$ denotes the total number of inquiry requests that x is allowed to make from s, $|Tags(x \to s)|$ denotes the number of neighbors tagged by x in favor of s, and 1 represents a fixed reward for x for providing its own judgment.

$$Inq(x \to s) = Inq(x \to s) + (|Tags(x \to s)| + \lceil |Tags(x \to s)| \times Cr(s \to x) \rceil + 1) \tag{33}$$

In this way, services would tend to contribute in the trust framework in order to increase their total number of inquiry requests that they can make and be able hence to participate in the coalition formation process. Moreover, by linking the number of possible inquiries with the credibility scores of the services, we are encouraging services to provide truthful

judgments in order to increase their credibility scores and increase hence their share of inquiry requests.

5.2.3 Trust Bootstrapping

Trust bootstrapping, i.e., assessing trust for newly deployed services, is a major issue that encounters our trust framework as no historical information about newcomers is available. For example, when a service is initially registered in a cloud center, no service has interacted with it and hence there is no record of its former behavior. As a result, its initial trust cannot be evaluated, which may lead to overlook this service in the future coalition formation processes. Therefore, a mechanism to allocate initial trust values for newly deployed services in the absence of historical information about their past behavior is needed.

The existing bootstrapping mechanisms in the Services-Oriented Computing (SOC) domain can be classified into three main categories: (1) Default-value-based, (2) punishment-based, and (3) adaptive mechanisms. The first approach assigns a default trust value for all of the new services. A self-evident drawback that encounters this approach is that it can arbitrarily favor either the already existing services or the newly deployed ones. Specifically, if the assigned default trust value is low, newly deployed services will be overlooked in the future coalition formation processes. On the other hand, if the assigned default trust value is high, newcomer services are favored over existing services that may have interacted and strived to achieve their trust values. This motivates malicious providers to continuously publish new identities to clear the past bad trust history of their services and gain high trust scores (a.k.a white-washing). As a remedy to white-washing, the punishment strategy proposes to assign low initial trust values for the newcomer services. In this way, however, the new services are disadvantaged since they will have no chance to make interactions and gain trust. In the adaptive strategy, the newcomer service is bootstrapped based on the rate of maliciousness in the community in which it registers. More specifically, this strategy assumes a community-based architecture in which a community groups a number of services sharing the same functionality to simplify the bootstrapping process. When a new service is registered in a certain community, it gets a trust value based on the rate of maliciousness in that community. However, the problem with this strategy is that it still allows malicious services to benefit from a low rate of maliciousness by leaving the network and rejoining again to obtain higher trust scores.

In this section, we propose a bootstrapping mechanism, based on the concept of endorsement in online social networks [113], that is resilient to white-washing. As mentioned earlier, each service maintains a dataset that records its previous interactions with several

services having different functional and non-functional specifications. Whenever a request from service i to bootstrap a new service j is received, the services that are interested in the request train a decision tree classifier on their datasets to predict an initial trust value for j. A decision tree [54] is a classification technique that recursively and repeatedly partitions the training set into subsets based on an attribute value test until all the samples at a given node belong to the same class or until splitting adds no more value to the classifications. Test attributes are selected based on some heuristic or statistical measure (e.g., information gain) that determines their relative importance in discriminating between classes being learned. Unlike some other classification techniques such as support vector machines and neural networks that entail high time and space complexity, the main advantages that make decision tree suitable for our bootstrapping problem are mainly its intuitive simplicity and computational time and space efficiency [41], which is important for resource-constrained nodes such as Web services.

The decision tree classifier analyzes the training dataset that contains properties and specifications for some existing services (e.g., provider's name, deployment country, etc.) and learns the patterns of the data by pairing each set of inputs with the expected output (e.g., the judgment on the services). To create the training and test sets, bootstrappers use the k-fold cross-validation with $k = \gamma$, where γ represents the number of folds. In this method, the dataset is split into k subsets, each of which is used each time as test set and the other $k-1$ subsets are merged together to compose the training set. The accuracy is then computed by averaging the error across all the k trials. This method has the advantage of reducing the bias of the classification results on the way based on which data is being divided due to the fact that each data point will be part of the test set exactly once and part of the training set $k-1$ times. Obviously, the computational complexity of this method grows as the size of k increases. Thus, the choice of γ would vary from one service to another depending on the available resources that each bootstrapper decides to dedicate to the bootstrapping process.

Bootstrappers use the learned classifier to predict an initial judgment for the services being bootstrapped. Based on the classification results, each service may endorse the underlying services either positively or negatively. Nonetheless, this does not constitute the final judgement. In fact, judgements from all bootstrappers are aggregated again by the bootstrapping requestor using Dempster-Shafer as described in the previous section to come up with a combined belief that is resilient to unreliable endorsements from colluding bootstrappers. Note that the results of the aggregation are not influenced by the number of bootstrappers even when this number is minimal since Dempster-Shafer is independent from the number of incoming observations. The extremely worst cases in this regard would be when only one service participates in the bootstrapping process or even no service is able to participate. In the former case, the requestor may rely on the opinion of the

bootstrapper without having to use Dempster-Shafer for aggregation. In the latter case, the requestor may bow to reality and use random guessing or default-value-based techniques. It's worth noting that the bootstrapping process is voluntary for bootstrappers in the sense that each service has the right to decide whether to train its classifier or not after each inquiry request received based on its available resources and whether to endorse services or not based on the underlying accuracy. This aspect is important to guarantee the fairness for both bootstrapper and bootstrapped services. In fact, after training the classifier and computing classifications' accuracy, some services may notice that they have no sufficient accuracy to make judgments due to the lack of similarity between the properties of the services that they had dealt with and the services being bootstrapped. Therefore, these services are better off refraining from submitting inaccurate endorsements.

Finally, the credibility scores of the bootstrappers are updated by the bootstrapping requestor according to Equation (32). The credibility update has two main advantages. On the one hand, it motivates the services having high levels of classification accuracy to participate in the bootstrapping mechanism to get their credibility scores increased. On the other hand, it demotivates the malicious services from submitting false endorsements to illegally promote some services or demote some other services and exclude them from future competitions.

5.2.4 Illustrative Example

In this section, we present an illustrative example to show how our proposed trust framework practically works. Consider the social network graph in Fig. 5.1 and assume that service S_1 wants to establish a trust relationship toward service S_5. To do so, it has first to discover all of S_5's reachable direct neighbors using Algorithm 3. Thus, it creates an empty tag instance t (Algorithm 3 - line 6), contacts its one-edge away neighbors S_2 and S_3 one by one, and marks them as explored for the current tag instance (Algorithm 3 - line 8). Starting with S_2, this latter gets appended directly to the tag instance (i.e., $t = \langle S_2 \rangle$) (Algorithm 3 - line 12) and checks whether it does have a direct edge with S_5 (Algorithm 3 - line 13). Since this is not the case, the algorithm recursively assigns the role of the source node to S_2 in lieu of S_1 (Algorithm 3 - line 17) to start inquiring its direct neighbor S_7 about S_5. S_7 gets marked as explored, gets appended to the tag instance (i.e., $t = \langle S_2, S_7 \rangle$), and checks whether it does have a direct edge with S_5. As this is the case, then the tag instance $t = \langle S_2, S_7 \rangle$ is returned to S_1 through the path $S_7 \rightarrow S_2 \rightarrow S_1$ (Algorithm 3 - line 14) and set to empty to restart the tagging process (Algorithm 3 - line 15). Since all of S_2's direct neighbors have been explored (Algorithm 3 - lines 19-21), the algorithm moves to S_1's second neighbor S_3. S_3 gets appended to the empty tag instance (i.e., $t = \langle S_3 \rangle$) and starts inquiring its direct neighbors S_4 and S_6 one by one. Starting with S_4, the node gets

appended to t (i.e., $t = \langle S_3, S_4 \rangle$) and gets marked as explored. It checks then whether it does have a direct edge with S_5. Since such an edge exists, the tag instance $t = \langle S_3, S_4 \rangle$ is returned to S_1 through the path $S_4 \rightarrow S_3 \rightarrow S_1$ and the tag instance is emptied again. Since not all of S_3's direct neighbors have been explored yet, the algorithm moves to S_3's second direct neighbor S_6 and repeats the same process that took place with S_4, where the tag instance $t = \langle S_3, S_6 \rangle$ gets returned to S_1 through the path $S_6 \rightarrow S_3 \rightarrow S_1$. Since all of S_3's direct neighbors have been explored now, the algorithm moves to checks for any additional neighbor of S_1 and stops after noticing that all the neighbors of S_1 have been explored. In this example, only 5 services (i.e., S_2, S_3, S_4, S_6, S_7) out of 7 and 8 edges (i.e., $S_1 \leftrightarrow S_2$, $S_1 \leftrightarrow S_3$, $S_2 \leftrightarrow S_7$, $S_7 \leftrightarrow S_5$, $S_3 \leftrightarrow S_4$, $S_3 \leftrightarrow S_6$, $S_4 \leftrightarrow S_5$, $S_6 \leftrightarrow S_5$) out of 9 have been explored. Therefore, the time complexity of the tagging process is linear in the number of services and edges in the social network graph due to the fact that each service and edge will be visited once in the worst case. In the example as well, 3 paths having each a size of 2 are returned in tagging instances; so in total $2 \times 3 = 6$ services are stored. This confirms that the space complexity of our services discovery algorithm, in its turn, is linear in the number of services.

As a second step, S_1 has to aggregate the judgments of S_5's direct neighbors, namely S_4, S_6, and S_7. Ostensibly, all of these three services would report that S_5 is trustworthy as depicted in Fig. 5.1. However, assume that S_4 and S_6 decide to collude and perform a slandering attack against S_5 by claiming both that this latter is malicious. As explained before, the judgement of each service is weighted based on its credibility score. Thus, the beliefs of the three services would be:

- **Service** S_4: $m_{S_4}(T) = 0$, $m_{S_4}(M) = 0.34$, and $m_{S_4}(U) = 0.66$.
- **Service** S_6: $m_{S_6}(T) = 0$, $m_{S_6}(M) = 0.23$, and $m_{S_6}(U) = 0.77$.
- **Service** S_7: $m_{S_7}(T) = 0.9$, $m_{S_7}(M) = 0$, and $m_{S_7}(U) = 0.1$.

First, let's combine the beliefs of the services S_4 and S_7. The details are explained in Table 5.3.

Table 5.3: Combination of S_4 and S_7's beliefs

S_4 \ S_7	$m_{S_7}(T) = 0.9$	$m_{S_7}(M) = 0$	$m_{S_7}(U) = 0.1$
$m_{S_4}(T) = 0$	0	0	0
$m_{S_4}(M) = 0.34$	0.306	0	0.034
$m_{S_4}(U) = 0.66$	0.594	0	0.066

- $K = m_{S_4}(T)m_{S_7}(T) + m_{S_4}(T)m_{S_7}(U) + m_{S_4}(U)m_{S_7}(T) + m_{S_4}(M)m_{S_7}(M) + m_{S_4}(M)m_{S_7}(U) + m_{S_4}(U)m_{S_7}(M) + m_{S_4}(U)m_{S_7}(U) = 0.694$.

- $m_{S_4}(T) \oplus m_{S_7}(T) = 1/K[m_{S_4}(T)m_{S_7}(T) + m_{S_4}(T)m_{S_7}(U) + m_{S_4}(U)m_{S_7}(T)] = \frac{0.594}{0.694} = 0.856.$
- $m_{S_4}(M) \oplus m_{S_7}(M) = 1/K[m_{S_4}(M)m_{S_7}(M) + m_{S_4}(M)m_{S_7}(U) + m_{S_4}(U)m_{S_7}(M)] = \frac{0.034}{0.694} = 0.049.$
- $m_{S_4}(U) \oplus m_{S_4}(U) = 1/K[m_{S_4}(U)m_{S_7}(U)] = \frac{0.066}{0.694} = 0.095.$

Then, we combine in Table 5.4 the combined beliefs of S_4 and S_7 with the beliefs of S_6.

Table 5.4: Combining S_4 and S_7's combined beliefs with the beliefs of S_6

S_6 \ S_4S_7	$m_{S_4S_7}(T) = 0.856$	$m_{S_4S_7}(M) = 0.049$	$m_{S_4S_7}(T) = 0.856 = 0.095$
$m_{S_6}(T) = 0$	0	0	0
$m_{S_6}(M) = 0.23$	0.19688	0.01127	0.02185
$m_{S_6}(T) = 0.77$	0.65912	0.03773	0.07315

- $K = 0.8031277.$
- $bel_{S_1}^{S_5}(T) = m_{S_4S_7}(T) \oplus m_{S_6}(T) = 0.821.$
- $bel_{S_1}^{S_5}(M) = m_{S_4S_7}(M) \oplus m_{S_6}(M) = 0.088.$
- $bel_{S_1}^{S_5}(U) = m_{S_4S_7}(U) \oplus m_{S_6}(U) = 0.091.$

Thus, S_1's belief in S_5's trustworthiness is still high (i.e., 0.821) although the majority of services colluded to claim the contrary.

Having computed the beliefs in S_5, S_1 should now update its credibility beliefs towards the services that have participated in the trust establishment process. Based on Equation (32), S_1's credibility belief towards S_4 would decrease to become: $Cr(S_1 \rightarrow S_4) = |0.34 - 0.088| = 0.252$ as its judgment does not agree with the computed belief. Similarly, S_1's credibility belief towards S_6 would decrease down to: $Cr(S_1 \rightarrow S_6) = |0.23 - 0.088| = 0.142$. On the other hand, the credibility towards the truthtelling service S_7 would increase up to: $Cr(S_1 \rightarrow S_7) = 0.9 + |0.821 - 0.9| = 0.979$. As a reward for tagging neighbors, S_2 (assuming that it was able to make one inquiry from S_1 before the tagging), that has tagged one of its neighbors to S_1, gets the number of inquiries it is able to make from S_1 increased up to $Inq(S_2 \rightarrow S_1) = 1 + (1 + \lceil 1 * 0.6 \rceil) + 1 = 4$ as per Equation (33). S_3 (assuming that it was able to make one inquiry from S_1 before the tagging), that has tagged two of its neighbors to S_1, gets the number of inquiries it is able to make from S_1 increased up to $Inq(S_3 \rightarrow S_1) = 1 + (2 + \lceil 2 * 0.7 \rceil) + 1 = 6$.

5.3 Trust-based Hedonic Coalitional Game

In this section, we model the problem of forming trusted multi-cloud communities as a hedonic coalitional game with non-transferable utility, propose the appropriate preference function, and analyse the properties of the game.

5.3.1 Game Formulation

A coalitional game is a game-theoretical model that analyzes the interactions among players when they gather into groups. The output of the coalitional game is a partition of the players' set into coalitions. For the proposed game, the players are the services that seek to form multi-cloud communities. The objective is to form trusted communities wherein the number of malicious members is minimal. Coalitional games may be either cohesive or non-cohesive games. In cohesive games, the motivation for coalescing is extreme [104]. That is, the formation of the single coalition that includes all the players referred to as *grand coalition* is the best option for all the players. In contrast, non-cohesive games are interested in studying situations wherein forming the grand coalition is costly for the players. Thus, the objective of non-cohesive games is to generate a set of disjoint coalitions. This type of games is often referred to as *coalition formation game*.

Property 1. *The proposed game is a coalition formation game.*

As mentioned earlier, the objective is to form trusted multi-cloud coalitions in which the number of malicious members is minimal. Obviously, the probability of encountering malicious services increases as the size of the coalitions increases. In other words, the grand coalition entails grouping all the trustworthy and malicious services together into a single coalition. Therefore, our objective of is to produce a set of disjoint coalitions instead of forming the grand coalition. Thus, the proposed game is a *coalition formation game*.

Coalitional games may be differentiated as well based on the utility that they assign to each coalition. Specifically, coalitional games may be either of Transferable Utility (TU) or Non-Transferable Utility (NTU). In TU games, the utility associated with each coalition of players worth the same for all the players who, as a result, can distribute and transfer this utility (e.g., money). In contrary, NTU games assume that the utility of the coalitions is non-distributable nor transferrable (e.g., happiness).

Property 2. *The proposed coalitional game is an NTU game.*

The utility of each service in a certain coalition is obtained by summing up the service's beliefs in trustworthiness in each of the coalition's members (Equation (25)). Apparently, the belief in trustworthiness is a social relationship in which an agent assigns a probability about another agent's future behavior [141]. That is, trust cannot be neither distributed nor transferred among services. Therefore, the proposed game is an NTU game.

A hedonic game is a special case of NTU games in which players have preferences over the coalitions that they may join and the utility of any player in a certain coalition depends *exclusively* on the *identity* of the members in that coalition regardless of how other services are structured. In simple words, the term *hedonic* comes from the idea that players seek to enjoy each other's partnership, apart from numeric considerations. Therefore, we believe that hedonic games are the best type of coalitional games that can model the trust relationships among services. More specifically, a hedonic game is a subclass of coalitional games that satisfies the two following requirements [26]:

1. The utility of any player in a given coalition depends only on the members of that coalition.
2. The players have preferences over the set of possible coalitions and coalitions form based on these preference relationships.

Property 3. *The proposed coalitional game is hedonic.*

In our game, the utility of the services in a certain coalition is obtained by summing up the service's beliefs in trustworthiness in each of the coalition's members (Equation (25)). Thus, the utility of services in a given coalition is solely dependent on the members of that coalition, which satisfies the first condition. For the second condition, we will formally define in the rest of this section the preference function that enables services to build preference relations between coalitions and compare them during the coalition formation process. Before discussing the preference function, let's define first the concept of *preference relation* [26].

Definition 8 (**Preference Relation).** *For every service $S_i \in \mathbb{N}$, a preference relation $(\geq_{S_i})_{S_i \in S \subseteq \mathbb{N}}$ is a complete, reflexive, and transitive binary relation over the set of all possible coalitions that S_i may join. $C_l \geq_{S_i} C'_l$ means that service S_i prefers to be a member of coalition C_l over being a member of C'_l or at least S_i prefers to be a member of both coalitions equally. $C_l >_{S_i} C'_l$ denotes that S_i strictly prefers to be part of C_l over being part of C'_l.*

Based on this definition, the preference function of the services can be defined as follows:

$$C_l \geq_{s_i} C'_l \Leftrightarrow P_{S_i}(C_l) \geq P_{S_i}(C'_l), \tag{34}$$

where $C_l \subseteq \mathbb{N}$ and $C'_l \subseteq \mathbb{N}$ are any two coalitions that service S_i is member of and $P_{S_i} : 2^{\mathbb{N}} \longmapsto \mathbb{R}$ is a preference function for any service S_i such that:

$$P_{S_i}(C) = \begin{cases} -\infty, & \text{if } a \in C \text{ \& } belief^a_{S_i}(T) < belief^a_{S_i}(M) \\ 0, & \text{if } C \in h_{S_i}(t) \\ U_{S_i}(C), & \text{otherwise,} \end{cases} \tag{35}$$

where $h_{S_i}(t)$ represents the history set of service S_i at time t. The history set $h_{S_i}(t)$ contains the coalitions that service S_i has already joined and left at any time $t' < t$ before the formation of the current coalition structure $\Pi(t)$. The main intuition behind the preference function P_{S_i} defined in Equation (35) is to allow each service S_i to choose the coalition that maximizes its belief in trustworthiness, while avoiding the coalitions that S_i believes contain malicious members. Particularly, the service S_i assigns a minimum preference (i.e., $-\infty$) to any coalition that contains a member that S_i believes is malicious (i.e., $a \in C$ & $belief^a_{S_i}(T) < belief^a_{S_i}(U)$). This condition is important to avoid being grouped with any malicious service in the same coalition. Moreover, the service avoids rejoining any previously visited coalition as long as the structure of the coalitions does not change. This may be considered as a basic learning process and is important to reduce the complexity of the coalition formation process since the already visited coalitions are excluded from the choice set of the services [119]. Otherwise, the service prefers the coalition that maximizes its utility that represents its belief in trustworthiness in the coalition's members (Equation (25)).

5.3.2 Hedonic Coalition Formation Algorithm

To achieve the solution of the game, we propose in this section a distributed hedonic coalition formation algorithm that enables services to make decisions about which coalitions to join in such a way to minimize the number of malicious services in the final coalition structure. The algorithm is depicted in Algorithm 4. The algorithm takes as input an initial partition of services at a certain time t (line 1) and outputs the final coalition structure obtained after applying the trust-based hedonic coalition formation algorithm (line 2). First, the time t is initialized along with the initial partition of services at that time $\Pi(t)$ and the history set of each service S_i belonging to that partition (line 4). The algorithm repeats the following steps. Each service S_i in the initial partition selects a given coalition C_l (line 8). For each member of C_l, if the member is newly deployed and having no past interactions,

Algorithm 4: Hedonic Coalition Formation Algorithm

1: **Input**: Initial partition of services $\Pi(t)$ at time t
2: **Output**: Final coalition structure $\Pi^*(t_f)$ at time t_f
3: **procedure** COALITIONFORMATION
4: Initialize $t = 0$, $\Pi(t) = \{C_1(t), ..., C_S(t)\}$, $h_{S_i}(t) = C_k^{S_i}(t)$
5: **repeat**
6: **repeat**
7: **for each** service $S_i \in \Pi(t)$ **do**
8: Select a coalition $C_l \in \Pi(t) \backslash C_k^{S_i}(t) \cup \{\emptyset\}$
9: **for each** service $S_j \in C_l$ **do**
10: **if** S_j has no previous interactions **then**
11: Request bootstrapping for S_j
12: **else**
13: Run Algorithm 3 to get $N(S_j)$
14: Compute $belief_{S_i}^{S_j}(T)$
15: **end if**
16: **end for**
17: Compute $belief_{S_i}^{C_l}(T)$ and $belief_{S_i}^{C_l}(M)$
18: Compare $P_{S_i}(C_l(t) \cup \{S_i\})$ and $P_{S_i}(C_k^{S_i}(t))$
19: **if** $C_l(t) \cup \{S_i\} >_{S_i} C_k^{S_i}(t)$
20: Leave C_k, i.e, $C_k(t) = C_k(t) \backslash \{S_i\}$
21: Join C_l, i.e., $C_l(t) = C_l(t) \cup \{S_i\}$
22: Update history, i.e., $h_{S_i}(t) = h_{S_i}(t) \cup \{C_l\}$
23: **else**
24: $\Pi(t+1) = \Pi(t)$
25: **end if**
26: **end for**
27: $t = t + 1$
28: **until** no change in the partition happens.
29: **until** ε elapses
30: $\Pi^*(t_f) = \Pi(t)$
31: **return** $\Pi^*(t_f)$
32: **end procedure**

S_i makes a request to bootstrap it (line 11); otherwise it runs the discovery algorithm described in Algorithm 3 to discover the member's direct neighbors (line 13). Thereafter, it computes the beliefs in trustworthiness and maliciousness for that member using Equation (26) and Equation (27) respectively (line 17). S_i uses then the preference function defined in Equation (35) to determine the preference order between its current coalition and the selected coalition (line 18). If the utility of S_i in the new coalition $C_l(t) \cup \{S_i\}$ exceeds its utility in the current coalition $C_k^{S_i}(t)$ (line 19), then it leaves the current coalition (line 20), joins the new coalition (line 21), and updates its history set by adding the newly joined coalition

to it (line 22). Otherwise, the partition of services remains unchanged (line 24). This process continues until converging to a Nash-stable coalition structure, i.e., the case where no service prefers to leave its current coalition and join another one (line 28). Note that the whole process is repeated periodically after a certain fixed period of time ε (line 29) to capture the changes that may occur in the partition; especially the dynamism in the services' trust values, arrival of new services, and leaving of existing services. For the computational complexity of Algorithm 4, the main complexity lies in the switch operations, i.e., the process of finding the next coalition to join (lines 7-18). The computational complexity of performing a switch operation is $O(\Pi)$, where Π is the coalition partition consisting of the disjoint coalitions of services. The worst case would be when each service acts alone in a singleton coalition as it implies that the number of coalitions in the coalition structure is exactly the number of services, i.e., $|\Pi| = |S|$.

5.3.3 Analysis of the Trust-based Hedonic Game

In this section, we analyze the properties of the proposed hedonic game. In particular, we analyze the convergence of the proposed coalition formation algorithm to a final solution and some stability concepts of the generated coalitions. Before starting the analysis, let's highlight some useful definitions and properties [26].

Definition 9 (**Nash Stability).** *A partition* Π *is Nash-stable if no player in* Π *has incentive to leave its current coalition and move to any other coalition (possibly empty) in such a way that makes the coalition structure change, assuming that the other coalitions remain unchanged, i.e.,* $\forall i \in N, C_k^{S_i} \geq_i C_l \cup \{i\}$ *for all* $C_l \in \Pi$.

In other words, a coalition structure Π is Nash-stable if (1) there exists no service S_i that prefers to leave its current coalition $C_k^{S_i} \in \Pi$ and act alone by forming its singleton coalition $\{S_i\}$, and (2) there exists no service S_i that has incentive leave its current coalition $C_k^{S_i} \in \Pi$ and join any other coalition $C_l \in \Pi$ in such a way that makes the coalition structure change.

Definition 10 (**Individual Stability).** *A partition* Π *is individually stable if no player in* Π *can benefit by moving from its current coalition to another coalition without making the members of the latter coalition worse off, i.e.,* $\nexists i \in N$ *and* $C_l \in \Pi \cup \{\emptyset\}$ *such that* $C_l \cup \{i\} >_i C_k^{S_i}$ *and* $C_l \cup \{i\} \geq_j C_l$, $\forall j \in C_l$.

In simple words, a coalitional structure Π is individually stable if there exists no coalition $C_l \in \Pi$ that a service S_i prefers over its current coalition $C_k^{S_i} \in \Pi$ without making its members

worse off.

Property 4. *The number of coalition structures for N services is finite and given by D_N , where D_N represents the N^{th} Bell number and is computed as follows:*

$$D_N = \sum_{i=0}^{N-1} \binom{N-1}{i} \cdot D_i \quad \textit{for} \quad N \geq 1 \quad \textit{and} \quad D_0 = 1 \tag{36}$$

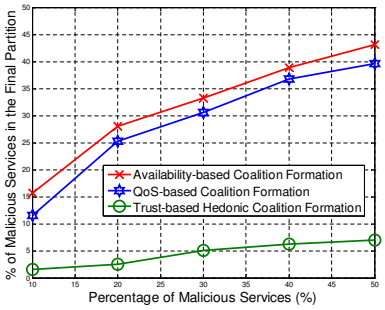
Now, let's move to the analysis of the proposed coalition formation algorithm's properties; particularly the three common properties of hedonic games: convergence to a final coalition structure, Nash-stability, and individual stability. It's worth noting that the methodology followed in the analysis is inspired by that presented in [119].

Theorem 4. *Algorithm 4 converges to a final coalition structure $\Pi^*(t_f)$ consisting of a number of disjoint coalitions.*

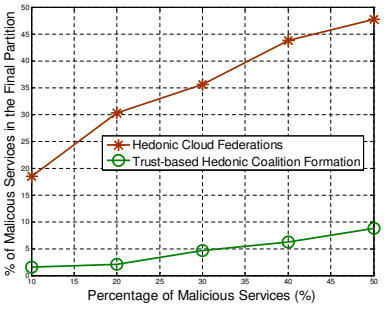
Proof. The proof involves showing that the algorithm leads to distinct coalitions from time t to time $t+1$ and that the number of coalition structures is finite. Given any initial partition $\Pi(t)$ of services at time t, Algorithm 4 switches the partition at hand $\Pi(t)$ into another partition $\Pi(t+1)$ at time $t+1 > t$ and so on until reaching the final partition $\Pi^*(t_f)$. Moreover, the preference function defined in Equation (35) states that services will not revisit any coalition that has been already visited and left. Thus, any switch operation done in Algorithm 4 leads to a new partition that has not been visited yet. Given this property and the fact that the number of coalition structures is finite as per Property 4, we can conclude that the number of switch operations is finite and that the switch operation always leads to a final partition $\Pi^*(t_f)$. Hence, Algorithm 4 always converges to a final coalition structure comprising a number of disjoint services coalitions. □

Theorem 5. *Algorithm 4 converges to a Nash-stable coalition structure $\Pi^*(t_f)$.*

Proof. The theorem may be proved by contradiction. Assume that the final coalition structure $\Pi^*(t_f)$ is not Nash-stable. Then, there exists a service S_i that prefers to leave its current coalition $C_k^{S_i}(t_f)$ and join another coalition $C_l(t_f)$ at time t_f (i.e., $C_l(t_f) \cup \{S_i\} >_{S_i} C_k^{S_i}(t_f)$). Consequently, the coalition structure $\Pi^*(t_f)$ changes to a new coalition structure $\Pi^{**}(t_f)$ such that $\Pi^{**}(t_f) \neq \Pi^*(t_f)$, which contradicts with Theorem 4. Hence, we can conclude that Algorithm 4 always converges to a Nash-stable coalition structure $\Pi^*(t_f)$. □

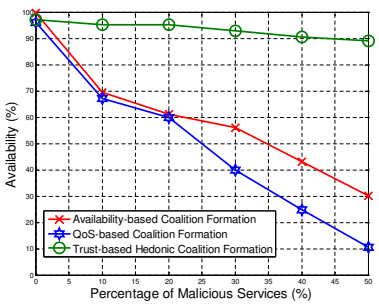


(a) Trust-based vs. Availability and QoS models

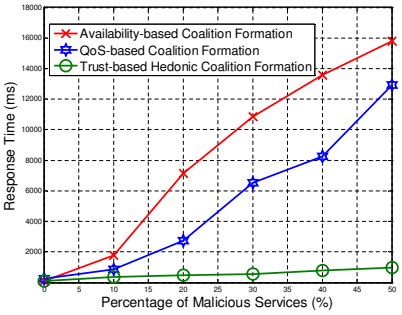


(b) Trust-based vs. Hedonic cloud federations models

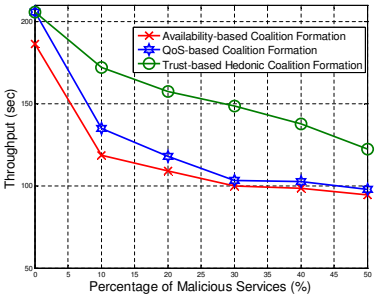
Figure 5.3: Percentage of malicious services: Our trust-based model minimizes the number of malicious services



(a) Availability



(b) Response time



(c) Throughput

Figure 5.4: Performance metrics: Our model improves availability, response time, and throughput compared to the Availability and QoS models

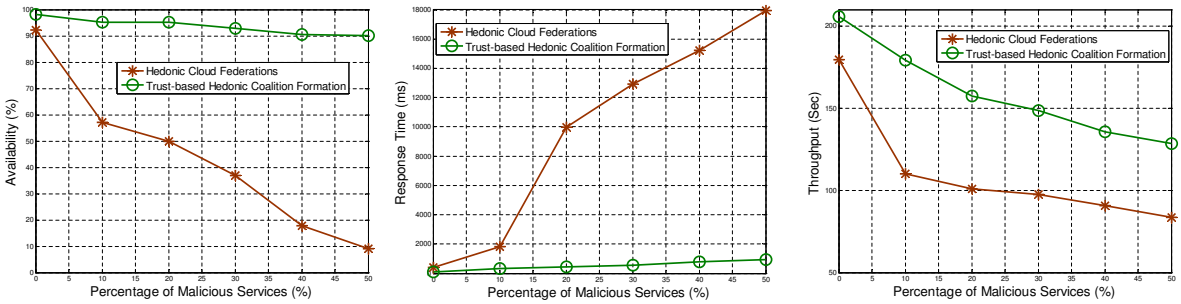
Theorem 6. *Algorithm 4 converges to an individually stable coalition structure* $\Pi^*(t_f)$.

Proof. It has been proven that every Nash-stable coalition structure is also individually stable [26]. Since Algorithm 4 converges to a Nash-stable coalition structure as per Theorem 5, Algorithm 4 converges also to an individually stable coalition structure. □

5.4 Experimental Results and Analysis

5.4.1 Experimental Setup

We implement our solution in a 64-bit Windows 7 environment on a machine equipped with an Intel Core i7-4790 CPU 3.60 GHz Processor and 16 GB RAM. Throughout simulations, we vary the percentage of malicious services from 0% to 50% and compare our



(a) Availability *(b) Response time* *(c) Throughput*

Figure 5.5: Performance metrics: Our model improves availability, response time, and throughput compared to the Hedonic Federations model

model with three other models: (1) Availability-based Coalition Formation [73], (2) QoS-based Coalition Formation [63], and (3) Hedonic Cloud Federations [88]. The Availability-based Coalition Formation considers the availability of the services as a building block in the community formation process. The QoS-based Coalition Formation considers, in addition to availability, several QoS metrics such as throughput and response time in the community formation process. The Hedonic Cloud Federations considers the prices and costs of the services (i.e., VMs) to formulate the utility function. The comparison is possible since all these approaches are based on a coalition formation algorithm. MATLAB has been used as a simulation tool to implement the different algorithms, where service instances have been modeled as objects; each of which having a set of QoS parameters. The QoS values such as promised and monitored availability are obtained from the CloudHarmony dataset[3], which contains information about services owned by well-known providers such as Amazon Web Services and Agile Cloud. The dataset comprises 53 different services operating in different parts of the world and 187 different activities for these services. The availability of the services has been studied during a period of a whole month and the average availability is recorded. During coalitions formation, the malicious services are considered as those that deviate from the SLA clauses. In particular, the initial decision on whether a certain service is trustworthy or not (i.e., before applying the proposed trust aggregation technique) is obtained by comparing the promised availability with the monitored availability. After coalitions are formed, the malicious services are considered those that refuse to share their needed resources with the coalition colleagues. To make our experiments fair with the Hedonic Cloud Federations model [88], which uses different parameters from ours (i.e., price and cost of VMs) and a small number of providers (i.e., eight providers), we have selected a subset of eight AmazonEC2 services (the same type of services used in [88]) from the used dataset, assigned them the same prices and costs

[3]http://cloudharmony.com/

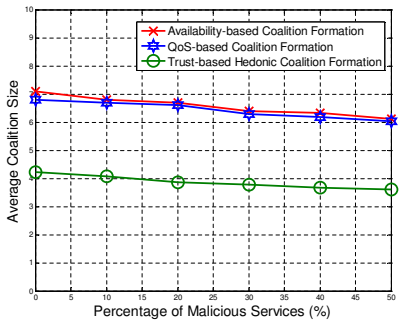
used in [88] which are publicly available[4], and run independent simulations from the ones used to compare with the other Availability-based and QoS-based models.

5.4.2 Experimental Results

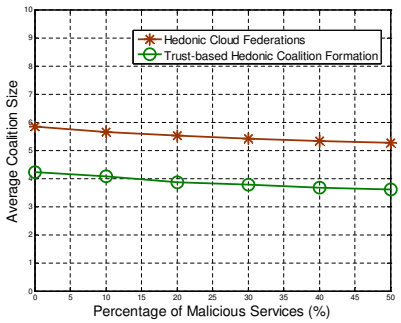
First, we study in Fig. 5.3 the percentage of malicious services that exist in the final coalition structure w.r.t the percentage of malicious services that existed in the initial partition of services. In other words, the aim is to study how effective is each of the compared models in avoiding the malicious services during communities formation. Fig. 5.3a shows that the percentage of malicious services in the final partition keeps increasing in the availability-based, QoS-based, and our trust-based hedonic coalition formation models with the increase in their percentage in the initial partition. However, the trust-based coalition formation model is more resilient to that increase and is able to reduce the percentage of malicious services up to 30% compared to the other models. The reason is that our model takes into account the trust relationships among services in the preference function (Equation (34)) of the hedonic game used during the coalition formation process and is able as well to overcome the collusion attacks that may affect the trust establishment results as per Theorem 3. On the other hand, the percentage of malicious services in the two other models turns out to be high. The reason is that although these models take into account some QoS metrics in the community formation, the declared metrics may not be consistent with the actual metrics in case of passive malicious misbehavior. Moreover, Fig. 5.3b reveals that our trust-based model outperforms the hedonic cloud federations model in terms of minimizing the percentage of malicious members. It is worth noticing that the hedonic cloud federations model entails a higher percentage of malicious members than both the availability-based and QoS-based models. The reason is that the former, contrary to the other two models, focuses solely on the prices and costs of services and disregards totally both the performance and security perspectives. Overall, we can conclude that if we allow services to rationally select their coalitions without considering their trust relationships, these services may have incentives to structure themselves into coalitions consisting of a large number of malicious services.

Next, we test the performance of the generated communities for a period spanning over more than 3 days (i.e., $260,000$ iterations) and compute the average availability, response time, and throughput; where each single iteration represents a second. At each iteration, we assign 1000 requests for every community; meaning that each community receives 1000 requests per second, which is realistic to a large degree. The malicious services at this stage are those that benefit from the resources of the other community colleagues but

[4] http://aws.amazon.com/ec2/pricing/



(a) Trust-based vs. Availability and QoS models

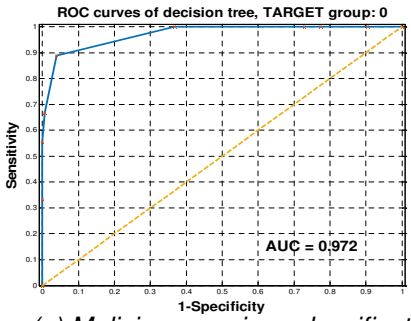


(b) Trust-based vs. Hedonic cloud federations models

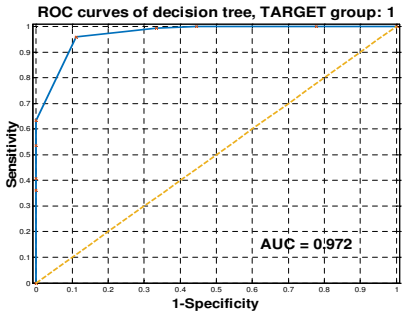
Figure 5.6: Average coalition size: Our trust-based model generates coalitions of smaller size

refuse to share their needed resources with them. Figs. 5.4 and 5.5 study how effective are the formed coalitions in terms of availability, response time, and throughput. Availability depicts the time period in which a community of services is ready for use. Fig. 5.4a shows that in the absence of malicious services in the initial partition, the availability-based coalition formation model outperforms the other two model by achieving an availability percentage of $\approx 100\%$. This result is expected since this model takes the availability as a sole factor for forming communities. However, starting from $\approx 5\%$ of malicious services, our model outperforms both the availability-based and QoS-based models whose performance begins to decrease drastically. This is due to the fact that our trust-based model minimizes the percentage of malicious services in the final partition as per Fig. 5.3a. Practically, the increase in the number of malicious services that refrain from sharing their resources when these resources are needed leads to an increase in the number of unfulfilled requests. Similarly, Fig. 5.5a reveals that our model outperforms the hedonic cloud federations in terms of availability for the same above-discussed arguments.

Figs. 5.4b and 5.5b studies how effective are the formed coalitions in terms of response time. Response time represents the time between the submission of the request and the receipt of the response, which incudes both service time and wait time. Fig. 5.4b reveals that our trust-based model yields a much lower response time compared to the availability-based and QoS-based models in the presence of malicious services. Similarly, Fig. 5.5b shows that our model outperforms the hedonic cloud federations model in terms of response time. This is also due to the fact that our trust-based model minimizes the percentage of malicious services in the final partition as per Fig. 5.3. Practically, the increase in the number of malicious services that refuse to share their needed resources entails additional wait time to find alternative non-malicious services and offering the same type of resources, which augments consequently the whole response time. Figs. 5.4c and 5.5c study the performance of the produced coalitions in terms of throughput. Throughput



(a) Malicious services classification



(b) Trusted services classification

Figure 5.7: Bootstrapping accuracy: Our bootstrapping mechanism achieves high accuracy rate

describes the number of requests that coalitions can handle in a given time. For our simulations, we measure the throughput per second. Fig. 5.4c reveals that our trust-based model yields much higher throughput compared to the other two models in the presence of malicious services. Similarly, Fig. 5.5c shows that our model outperforms the hedonic cloud federations model in terms of throughput. Obviously, the improvement of throughput is a natural result of the improvement in terms of response time.

Fig. 5.6 measures the average coalitions size w.r.t. the increase in the percentage of malicious services in the initial partition. Fig. 5.6a reveals that our trust-based model generates coalitions of smaller size than those generated by the availability-based and QoS-based models. Moreover, Fig. 5.6b shows that the size of the coalitions produced by our trust-based model is smaller than that of the coalitions produced by the hedonic cloud federations model. The intuition behind this result is that coalitions of smaller sizes are able to reduce the number of malicious services. In other words, the size difference between our model and the other models may be thought of as the malicious services that our model excludes from the coalition structure. Thus, we can conclude that our model produces a network of large number of small disjoint coalitions.

Finally, we study the effectiveness of our trust bootstrapping technique in providing accurate initial trust values. To this end, we train a decision tree classifier on our dataset using the 10-fold cross-validation technique. The dataset consists of four attributes: $Service\_Provider$, $Operation\_Country$, $Promised\_Availability$, and $Status$. The first attribute denotes the service provider's name, the second attribute denotes the country in which the service was used, $Promised\_Availability$ denotes the availability promised for this service in the SLA contract, and $Status$ denotes the status of the service which can be either trusted or not (based on the compliance of the promised QoS metrics with the actual ones). Note that we have excluded the $Monitored\_Availability$ attribute from the dataset since this metric cannot be known a priori for the services being bootstrapped. The *relative importance* of

the $Promised\_Availability$ attribute in the decision tree classifier is reported to be 0% since usually all providers tend to promise optimal availability (i.e., 100%), compared to 100% for the $Service\_Provider$ attribute and 64.1% for the $Operation\_Country$ attribute. Fig. 5.7a and Fig. 5.7b represent the Receiver Operating Characteristic (ROC) curves [39] generated by the classifier, where sensitivity measures the proportion of positives that are correctly classified as such (a.k.a true positive rate) and specificity measures the proportion of negatives that are correctly classified as such (a.k.a true negative rate). Thus, $1-$ specificity means the proportion of positives that are misclassified as negatives (i.e., false positive rate). Fig. 5.7a measures the accuracy of the bootstrapping mechanism in classifying the malicious services as such. Thus, sensitivity means in this figure the percentage of malicious services that are correctly classified as malicious and $1-$specificity means the percentage of malicious services that are misclassified as trustworthy. On the contrary, Fig. 5.7b measures the accuracy of the bootstrapping mechanism in classifying the trustworthy services as such. Thus, sensitivity means in this figure the percentage of trustworthy services correctly classified as trustworthy, whereas $1-$specificity means the percentage of trustworthy services that are misclassified as malicious. The best possible classification model would yield 100% sensitivity (no false negatives) and 100% specificity (no false positives); thus a point whose coordinates are $(0,\ 1)$. The dashed diagonal line represents a fully random guess (Fig. 5.7). By carefully inspecting both Fig. 5.7a and Fig. 5.7b, we can notice that the sensitivity and specificity measures are nearly optimal in our bootstrapping mechanism. The overall classifier's accuracy is quantified in terms of Area Under the Curve (AUC) [39], where a value of 1 represents a perfect test and a value of 0.5 represents a worthless test. Figs. 5.7a and 5.7b reveal that our bootstrapping mechanism yields high AUC values up to 0.972 in both Figs.

5.5 Conclusion

In this chapter, we discussed a comprehensive trust framework that allows services to establish credible trust relationships in the presence of collusion attacks in which attackers collude to mislead the trust results. We introduced as well a trust bootstrapping mechanism that capitalizes on the concept of endorsement in online social networks to assign initial trust values for the newly deployed services. Thereafter, we designed a trust-based hedonic coalitional game that is able to form trusted multi-cloud services' communities and proposed a relevant algorithm that converges to a stable coalition structure. Experiments conducted on a real cloud services dataset revealed that our proposed solution minimizes the number of malicious services in the final coalition structure up to 30% compared to three state-of-the-art cloud federation and service communities formation models.

Moreover, our solution improves the performance of the formed communities in terms of availability, response time, and throughput. Besides, the proposed bootstrapping mechanism yields high accuracy levels up to 97.2%. Thus, we have achieved our second research objective (**Objective** 2) discussed in Chapter 1, which aimed at enabling the formation of trustworthy communities of multi-cloud services wherein the number of passive malicious services is minimal.

Despite the effectiveness of the proposed security-oriented community formation model in minimizing the number of malicious members, some services can still change their behavior after joining communities or even get compromised to perform some malicious activities. To cope with such a challenge, the next chapter is dedicated to tackling the problem of detecting the malicious services that launch active attacks against the formed communities and/or other services, with focus on DDoS attacks as a case study. To expand the applicability of our solution, the proposed detection approach discussed in the next chapter is presented in a generalized manner so as to fit both community-based and non-community-based applications.

Chapter 6

Optimal Load Distribution for the Detection of VM-based DDoS Attacks in the Cloud

Distributed Denial of Service (DDoS) constitutes a major threat against cloud systems owing to the large financial losses it incurs. This motivated the security research community to investigate numerous detection techniques [76, 134, 101, 156, 71, 77] to limit such attack's effects. Yet, the existing solutions are still not mature enough to satisfy a cloud-dedicated detection system's requirements since they overlook the attacker's wily strategies that exploit the cloud's elastic and multi-tenant properties, and ignore the cloud system's resource constraints. Motivated by this fact, the objective of this chapter is to develop an offline detection load distribution strategy that enables the hypervisor, based on the trust relationships it builds toward VMs, to learn about the optimal detection load percentage that should be allocated to each of its guest VMs in real-time. The purpose is to maximize the detection of distributed active attacks under a limited amount of resources (e.g., CPU, memory, and network bandwidth), with focus on DDoS attacks. To attain this objective, we propose first a trust framework that enables the hypervisor to construct trust relationships toward guest VMs. To ensure building credible relationships, we combine both subjective and objective sources of trust and aggregate them using the Bayesian inference theory [48]. On top of the proposed trust framework, we design a resource-aware trust-based maxmin game between the hypervisor and DDoS attackers whose solution guides the hypervisor to determine the optimal detection load distribution among VMs in real-time that maximizes DDoS attacks' detection[1].

[1] The content of this chapter is published in [146]

6.1 System Model and Assumptions

We formulate in this section the studied problem formally and then explain the attack model considered in this chapter.

6.1.1 System Model and Strategies

Let $H = \{h_1, h_2, \ldots, h_n\}$ be a finite set of hypervisors, where each hypervisor $h_i \in H$ hosts a set of virtual machines $V_i = \{v_1, v_2, \ldots, v_l\}$. Note that when i is not important or can be induced from the context, we simply use V instead of V_i. Each virtual machine $v_j \in V$ residing on h_i is owned by a client from the set $C = \{c_1, c_2, \ldots, c_m\}$. A hypervisor $h_i \in H$ is a software agent that stays between the cloud system's hardware and the VMs and whose role is to emulate a set of hardware resources $I = \{I_1, I_2, ..., I_n\}$ and to schedule the access of the VMs to it in order to enable the synchronous running of multiple VMs on a shared cloud infrastructure. A virtual machine $v \in V$ (to simplify the notation, we omit the index whenever possible) is a pair $\langle O; A \rangle$, where O represents the underlying operating system (OS) and A denotes the set of applications running inside v.

As a first stage, the hypervisor seeks to establish trust relationships toward its guest VMs. To do so, it first monitors and analyzes the CPU, memory, and network bandwidth utilization of each $v \in V$ to determine any abnormal consumption of those resources (i.e., over-utilization). This allows the hypervisor h to build an initial belief in each v's trustworthiness denoted as $InitialBelief_h^v$. The hypervisor then collects recommendations from other VMs/hypervisors on the behavior of the underlying VMs. In the rest of this section, we abstract away the identity of recommenders and refer to both VMs and hypervisors as *source*. Each recommendation $R_s^v \in [0,1]$ denotes a certain source s's recommendation on the behavior of a VM v based on their previous interactions (e.g., compositions, hosting). Note that each source s enjoys a fixed number of inquiries it is allowed to make from every (other) hypervisor h' and is denoted by $Inq(s \rightarrow h')$. The motivation behind this assumption is to encourage services to participate in the trust establishment process through linking the number of inquiries that they are allowed to make with the degree of their participation in the trust framework. This assumption is realistic since we are considering a *selfish* environment in which a service will respond to a certain inquiry coming from another service only if the latter has previously responded the former's inquiries. Initially, all sources enjoy an equal amount of inquiries, where this amount is updated later during the trust establishment process (See Section 6.2). Now, the hypervisor h aggregates the results of the monitoring phase with the results of the recommendations phase using the *Bayesian inference* technique to come up with a final belief $Belief_h^v$ in each v's trustworthiness (See Section 6.2.3).

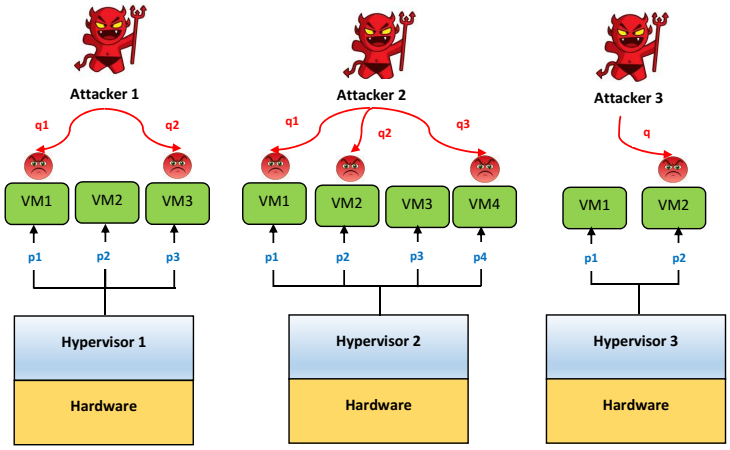


Figure 6.1: Attack scenario: Attackers distribute their attacks over a set of VMs to minimize the detection probability, while hypervisors distribute the detection load over the set of guest VMs to maximize this minimization

Having computed the final trust scores for the VMs, the hypervisor integrates these scores into its utility function (see Section 6.3). The objective is to benefit from the computed trust scores to find the optimal distribution of the detection load among its set of guest VMs that maximizes the attacks detection probability, knowing that DDoS attackers are distributing their attacks over a set of VMs to minimize this maximization.

6.1.2 Attacker Strategy

In order to complicate the detection process and rip off the hypervisor, attackers can use a mixed strategy by distributing a single DoS attack over multiple VMs running on top of the same hypervisor. To this end, the attacker splits its attack code into several malicious fragments and assigns a set of fragments to each VM. Each malicious VM aims at sending k malicious fragments to the hypervisor at different time intervals. Let $Q_V = (q(v_1), ..., q(v_l))$ denote the probability distribution vector of the attacks over the set V deployed on hypervisor h such that $\sum_{v \in V} q(v) = 1$. The attack succeeds if one or many malicious fragments attain the hypervisor without being detected.

Definition 11 (**Distributed Attack).** *A distributed attack is a set of k malicious fragments $\{f_1, ..., f_k\}$ distributed over V with a probability of $q(v)$ for each $v \in V$ such that $\sum_{v \in V} q(v) = 1$.*

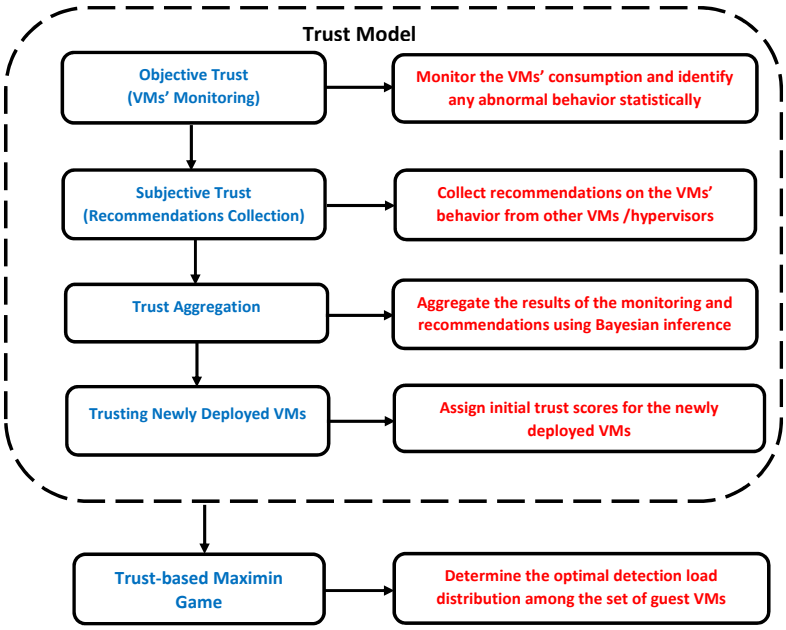


Figure 6.2: Solution methodology of the optimal detection load distribution model

6.1.3 Hypervisor Strategy

Knowing this fact, the hypervisor, having a limited amount of resources to be dedicated for detection, has to choose a mixed strategy consisting of the optimal detection load probability distribution vector $P_V = (p(v_1), ..., p(v_l))$ over the set V such that $\sum_{v \in V} p(v) = 1$.

For the readers' convenience, a graphical formulation of the above-mentioned problem is given in Fig. 6.1 and the methodology followed to perform our solution is schematized in Fig. 6.2. Moreover, Table 6.1 defines and summarizes the different notations that are used throughout this chapter.

6.1.4 Attack Model

We consider in this work a DDoS attack scenario wherein attackers are a group of VMs targeting a particular cloud system. Although DDoS attacks in a cloud environment may take many forms and can be seen in different contexts (e.g., application, web services, network, etc.), we focus in this work on the DDoS attacks that occur at the virtualization layer between the hypervisor and its guest VMs (i.e., VM-based DDoS). Particularly, we study the case in which attacking VMs try to flood the victim cloud system in such a way that makes it unavailable to support further VMs. These attacking VMs may be either

Table 6.1: Notations

Symbol		Significance
H	:	Set of hypervisors.
V_i (or simply V)	:	Set of virtual machines hosted on top of hypervisor h.
$InitialBelief_h^v$	:	Initial belief of hypervisor h in virtual machine v's trustworthiness.
$Belief_h^v$	:	Final belief of hypervisor h in virtual machine v's trustworthiness.
R_s^v	:	Recommendation given by a certain source s on the behavior of a virtual machine v.
$Inq(s \rightarrow h')$	:	Number of inquiries that a source s is allowed to make from hypervisor h'.
Q_V	:	The attack probability distribution vector over the set of virtual machines V_h deployed on hypervisor h.
$q(v)$	:	Attack distribution probability on virtual machine v.
$q_x(v)$	:	Attack distribution probability on virtual machine v at time x.
P_{V_h}	:	The detection load probability distribution vector over the set of virtual machines V_h deployed on hypervisor h.
$p(v)$	:	Probability of detection load allocated to virtual machine v.
$p_x(v)$	:	Probability of detection load allocated to virtual machine v at time x.
$W(v)$	:	Worth of virtual machine v.
$[t_1, t_2]$	:	Window of time starting at time t_1 and ending at time t_2.
$t_2 + 1$	:	Current system time.
$\beta_{[t_1,t_2]}$	:	Average detection rate of the IDS agent running on hypervisor h during the window of time $[t_1, t_2]$.
$U_{t_2+1}(h)$	:	Utility function of hypervisor h at time $t_2 + 1$.
$U_{t_2+1}(a)$	:	Utility function of attacker a at time $t_2 + 1$.

malicious or compromised by a malicious attacker to serve as bots in the DDoS attack process[2]. To perform their attacks, DDoS attackers benefit particularly from the *auto-scaling* (a.k.a *elasticity*) property that is provided by the virtualization technology as an appealing added-value feature to the cloud computing systems. Specifically, auto-scaling enables the cloud to keep assigning extra resources to the VMs that are in need of additional resources. Fortunately, such a property leads to enhance the cloud system's performance due to the fact that a VM will never undergo resource outages as long as the VM's owner agrees to keep paying bills versus receiving additional resources. Unfortunately, this property is being appealingly manipulated by DDoS attackers who compromise VMs and keep sending, through these compromised VMs, fake resources scaling requests. This allows the attacker to achieve the two following malicious objectives: (1) draining the resources of the cloud system to make it unable to support further VMs; and (2) increasing the bill of the VMs' real owners by obliging them to pay for (supposedly) unrequested resources (a.k.a Economic Denial of Sustainability (EDoS)) [128].

[2]In the rest of this chapter, we abstract on the type of attacking VMs and refer to them as *attackers*.

6.2 Building Trust on Virtual Machines

We describe in this section the details of the proposed trust model consisting of four main phases: virtual machines monitoring, recommendations collection, trust aggregation, and trusting newly deployed VMs.

6.2.1 Objective Trust: Virtual Machines Monitoring

In this phase, the hypervisor monitors the VMs' CPU, memory, and network bandwidth consumption directly from the hosting infrastructure and applies the Interquartile Range (IQR) statistical measure [32] to identify any abnormal usage. This constitutes the objective source of trust which is of prime importance in the field of trust and reputation to avoid biased and/or subjective judgements [141]. The IQR is a measure of variability whose basic idea is to split a given set of data into disjoint quartiles (i.e., Q_1, Q_2, and Q_3).

The first quartile Q_1 corresponds to the value in the data set that 25% of the values are smaller than it. The second quartile Q_2 stands for the data set's median value. The third quartile Q_3 represents the value in the data set that 25% of the values are higher than it. The IQR is obtained then by subtracting the first quartile from the third quartile. The reasons for choosing the IQR measure for the considered problem lie in its (1) robustness to messy data and outliers, and (2) simple and lightweight nature that imposes no heavy computation efforts on the hypervisor [32].

The algorithm of this phase that is executed by the hypervisor is depicted in Algorithm 5. Having monitored and recorded the CPU, memory, and network bandwidth consumption of the VM in question at time t (i.e., for the time window $[t-\mu,t]$), the hypervisor computes, for each of these metrics (e.g., CPU), the median usage of the VM (step 20). It finds then, based on the computed median, the first and third quartiles Q_1 and Q_3 for each metric respectively (steps 21-22). Using these quartiles, the IQR is computed by subtracting Q_3 from Q_1 and multiplying the obtained value by 1.5 (step 23). Intuitively, this means that any value lying more than one and a half times beyond the upper quartile is considered to be an outlier according to Tukey analysis [133]. By adding the IQR to the third quartile, the hypervisor computes the upper consumption limit for each underlying metric (step 24). Intuitively, this limit represents the pattern of maximal habitual utilization of the VM at a certain time period; where any future utilization above this limit would be considered unusual. The hypervisor checks then for any future consumption of the VM at time $t+\mu$ whether there exists any consumption that exceeds the computed upper limit (lines 25-26). If so, this event is appended to a table that stores the VM's unusual consumption (line 27)

Algorithm 5: Virtual Machines' Monitoring

Initialization:
μ: size of time window after which the algorithm is to be repeated
v: a VM being monitored by the hypervisor
$U = \{CPU,\ memory,\ and\ bandwidth\}$: the set of v's metrics to be analyzed by the hypervisor
$U_v^x(t)$: a table recording the amount of each metric $x \in U$ consumed by v during the time interval $[t-\mu, t]$
$M_v^x(t)$: the median consumption of $x \in U$ by v during the time interval $[t-\mu, t]$
$Q1_v^x(t)$: the 1st quartile consumption of $x \in U$ during the time interval $[t-\mu, t]$
$Q3_v^x(t)$: the 3rd quartile consumption of $x \in U$ during the time interval $[t-\mu, t]$
$IQR_v^x(t)$: the IQR consumption of $x \in U$ by v during the time interval $[t-\mu, t]$
$L^x(t)$: the upper consumption limit of $x \in U$ during the time interval $[t-\mu, t]$
$OverUse_v^x$: sum of v's unusual consumption of $x \in U$ (initialized to 0)
$CountOverUse_v^x$: a counter enumerating the occurrence of unusual consumption of $x \in U$ by v (initialized to 0)
$AvgOverUse_v^x$: v's average unusual consumption of $x \in U$
$PropOverUse_v^x$: v's unusual consumption of $x \in U$ proportionally to the upper consumption limit of this x
$|OverusedMetrics|$: the number of metrics that v overconsumed such that $|OverusedMetrics| \leq |U|$
$InitialBelief_h^v$: the initial belief of hypervisor h in v's trustworthiness

procedure VMMONITORING
  repeat
    for each metric $x \in U$ **do**
      Compute the median $M_v^x(t)$ of $U_v^x(t)$
      Find $Q1_v^x(t)$ as the median of $U_v^x(t)$'s lower half
      Find $Q3_v^x(t)$ as the median of $U_v^x(t)$'s upper half
      Compute $IQR_v^x(t) = (Q3_v^x(t) - Q1_v^x(t)) \times 1.5$
      Compute $L_v^x(t) = IQR_v^x(t) + Q3_v^x(t)$
      for each data point $y \in U_v^x(t+\mu)$ **do**
        if $y > L_v^x(t)$ **then**
          $OverUse_v^x = OverUse_v^x + y$
          $CountOverUse_v^x = CountOverUse_v^x + 1$
        end if
      end for
      if $CountOverUse_v^x > 0$ **then**
        $AvgOverUse_v^x = OverUse_v^x / CountOverUse_v^x$
        $PropOverUse_v^x = L_v^x(t) / AvgOverUse_v^x$
        $|OverusedMetrics| = |OverusedMetrics| + 1$
      end
    end for
    if $|OverusedMetrics| = 0$ **then**
      $InitialBelief_h^v = 1$
    else
      $InitialBelief_h^v = \frac{\sum_{x \in U} PropOverUse_v^x}{|OverusedMetrics|}$
    end
  until μ **elapses**
end procedure

and the average of unusual consumption for each metric is computed (line 32). The hypervisor computes finally its initial belief in the VM's trustworthiness by dividing the sum of average unusual consumptions over all the metrics by the number of metrics that the VM has overconsumed, if any (line 40). If no metric has been overconsumed, the initial belief in the VM's trustworthiness would be set to 1 (line 38), which represents a full initial trust in the VM. Note finally that the whole process is repeated periodically after a certain period of time μ to continuously capture the dynamism in the VMs' performance and behavior.

For example, suppose that the CPU, memory, and bandwidth upper consumption limits for a VM v were 60%, 70%, and 50% respectively. Suppose as well that v has been overconsuming the CPU with an average of 73%, the memory with an average of 73%, but has not been overconsuming the bandwidth metric. Then, the proportional overuse of CPU and memory would be calculated respectively as follows (Algorithm 5 - line 33): $PropOverUse_v^{CPU} = 60/73 = 0.822$ and $PropOverUse_v^{Memory} = 70/73 = 0.959$. The initial hypervisor's belief in v's trustworthiness would then amount to: $InitialBelief_h^v = \frac{0.822+0.959}{2} = 0.8905$. Note that we have divided by 2 since only two metrics, namely the CPU and memory have been overconsumed by v. Consider now another case wherein v was overconsuming the CPU with an average of 95% and the memory with an average of 98%. Then, the proportional overuse of CPU and memory would be calculated respectively as follows: $PropOverUse_v^{CPU} = 60/95 = 0.631$ and $PropOverUse_v^{Memory} = 70/98 = 0.714$. The initial hypervisor's belief in v's trustworthiness would then amount to: $InitialBelief_h^v = \frac{0.631+0.714}{2} = 0.6725$. We notice from the two examples that as the overconsumption keeps going far from the upper consumption limit, the initial trust score keeps decreasing (i.e., $0.6725 < 0.8905$).

6.2.2 Subjective Trust: Recommendations Collection

In order to enhance the quality of the trust scores, the hypervisor collects recommendations $R_{s_1}^v, \ldots, R_{s_n}^v$ (where $R_{s_i}^v \in [0,1]$) on the former behavior of the VMs. This constitutes the subjective source of trust and is widely used in the context of trust and reputation due to the fact that it consults different parties' opinions to improve the quality of the judgements [141]. The source of the recommendations may be either VM(s) having dealt with the VM in question or other hypervisor(s) having previously hosted that VM. The recommendations obtained from the former source (i.e., VMs) are important in order to learn about the performance of the VMs in the cases of services cooperation or composition in the cloud. The recommendations obtained from the latter source (i.e., hypervisors) allow us to capture the dynamism in the VMs' performance on different cloud infrastructures, which alleviates the risk of misjudging VMs because of bad performance of the host cloud system. This contributes in enhancing the detection accuracy under a changing cloud infrastructure environment. The recommendations are derived based on the overall behavior of the

VMs, not only based on the metrics related to DDoS attacks. For example, if a certain VM is not launching a DDoS attack but launching a side-channel attack [117], then it should receive low recommendation scores. We argue that obtaining such recommendations is becoming easier with the emerging cooperation architectures that are being proposed and adopted for cloud-based services such as services' communities and cloud federations [152, 144]. Practically, such architectures allow services, coming either from one cloud or deployed even in different cloud centers, to cooperate with one another in order to improve the performance and security of the underlying system. In this way, services (in the form of VMs) are allowed to easily migrate from one cloud infrastructure to another as a result of the community/federation agreement contract. In the same context, the VMs that are grouped in the same community/federation are likely to cooperate with one another to better respond to customers' requests. Therefore, obtaining recommendations from both hypervisors and VMs is becoming simpler and more realistic.

6.2.3 Trust Aggregation

Having obtained both the objective and subjective sources of trust, the next step is to aggregate these sources and come up with final aggregate trust scores for the VMs. For this purpose, we employ the Bayesian inference from the *subjective probability theory* [48], which has shown to be effective in aggregating trust sources in many application domains [69, 153]. Bayesian inference is a theory that describes uncertainty using a probability distribution. In simple words, assume that a person has an uncertainty about an issue. This uncertainty may be depicted using a probability distribution known as that person's *prior distribution*. Assume now that this person has been able to gain some information pertinent to that issue. The information evolves his uncertainty, which may be then represented as a new probability distribution called *posterior distribution*. This posterior distribution reflects the knowledge obtained from both the prior distribution and new information. The main function of Bayesian inference lies in the process of moving from prior to posterior distribution. In our case, the prior distribution represents the hypervisor's initial beliefs about VMs obtained from the monitoring process described in Algorithm 5 prior to collecting recommendations. This allows us to overcome a substantial problem of Bayesian inference caused by the arbitrary choices of the initial prior beliefs, where such uninformative prior beliefs have a great negative impact on the precision of the posterior final beliefs [129]. Once the recommendations are gathered, the prior distribution is converted into a posterior distribution that reflects the updated hypervisor's beliefs after analyzing the received recommendations. This is done by employing the *conditional probability* laws often referred to as *Bayes theorem* [136].

By applying the *Bayes theorem* for aggregating the objective and subjective trust sources,

we get that the Bayesian estimation of the trust belief in a VM v with n recommendations $R^v_{s_1},\ldots,R^v_{s_n}$ (where $R^v_{s_i}\in[0,1]$) is given by:

$$Belief^v_h=\sum_{i=1}^{n}\frac{R^v_{s_i}+n\gamma}{2n}, \tag{37}$$

where $\gamma=InitialBelief^v_h$ represents the hypervisor h's prior belief in v's trustworthiness, $R^v_{s_i}$ denotes a recommendation given on v by a source s_i, and n is the total number of collected recommendations. Note that a similar aggregation function has been used in [69, 153] to compute trust values for Web services willing to participate in composition processes.

As a reward for submitting recommendations, the VMs and hypervisors should receive some payment. The payment is given in the form of inquiry requests that they can make from (other) hypervisors on the behavior of (other) VMs. Specifically, a source s receives an increase in the number of inquiries $Inq(s\rightarrow h')$ it can make from hypervisor h' for which it has recommended a VM v' proportionally to the difference between the trust recommendation $R^{v'}_s$ submitted by s and the final hypervisor h''s belief in v''s trustworthiness. Formally, let $diff=\left|R^{v'}_s-Belief^{v'}_{h'}\right|$, the source s would receive the following payment:

$$Inq(s\rightarrow h')=\begin{cases}\lceil\frac{Inq(s\rightarrow h')}{diff\times\alpha}\rceil, & \text{if } Inq(s\rightarrow h')>0\\ \lceil\frac{1}{diff\times\alpha}\rceil, & \text{otherwise}\end{cases} \tag{38}$$

The purpose of this payment mechanism is two-fold. On the one hand, it stimulates the participation of both VMs and hypervisors in the trust establishment process. Practically, the hypervisors and VMs that refuse to participate would end up being unable to make further inquiries about the behavior of (other) VMs since the number of inquiries that they are able to make would be drained over the time without receiving any additional reward, which deprives them from constructing further trust beliefs. On the other hand, the proposed payment mechanism motivates these VMs and hypervisors to give honest recommendations through making the amount of payment proportional to the difference between the given recommendations and the final hypervisor's belief. In this way, hypervisors/VMs whose recommendations diverge from the final belief will get their payment decreased; whereas those whose submitted recommendations which are convergent to the final belief would receive a larger amount of payment. Note finally that α is a smoothing factor chosen by the designer and whose main role is to avoid the saturation of VMs/hypervisors in terms of number of inquiries and motivate thus their further participation in the trust establishment process. In this way, the larger α is, the lower the number of additional inquiries rewarded to VMs/hypervisors would be.

6.2.4 Trusting Newly Deployed VMs

Building trust relationships toward the VMs that are newly deployed in cloud centers constitutes evidently a serious obstacle against our proposed trust mechanism. Indeed, the absence of any historical and actual information that corroborates the performance of such VMs makes it quite difficult to compute trust scores for them. This raises the need for a mechanism enabling the hypervisor to assign initial trust values for the newly deployed VMs in the absence of any historical and current data. Such a problem is referred to as a trust bootstrapping problem [152]. To handle this issue, we capitalize on the trust bootstrapping mechanism proposed in Chapter 5 (Section 5.2.3) to form trustworthy multi-cloud services communities and that combines the concept of endorsement in online social networks (e.g., LinkedIn) with the decision tree classification technique to solve the problem. We borrow the overall logic used in that mechanism, while performing some technical updates to adapt it to our studied problem. The idea is explained in the following.

Whenever a hypervisor is willing to build trust toward a certain VM that is newly created, it sends a bootstrapping requests to other VMs and hypervisor asking to endorse the VM in question. Interested VMs/hypervisors (e.g., those having enough resources to participate in such a process) train a decision tree classifier on the dataset containing the details of their interactions (e.g. provider's name, operation country, etc.) with several VMs having various functional and non-functional properties. The classifier learns the patterns of the data by pairing each set of inputs (e.g, provider's name, operation country) with the corresponding output (i.e., trust score). For this sake, boostrappers use the k-fold cross-validation technique to create training and testing sets. In this way, the dataset gets split into k subsets, each used everytime as test set and the other $k-1$ subsets are combined altogether to form up the training set. The accuracy of the training process is then assessed by bootstrappers to decide on wether to submit endorsements or not. Particularly, if the underlying accuracy is high, this means that there exists a worthy similarity between the VM being bootstrapped and (some of) the VMs that bootstrappers have dealt with. In this case, bootstrappers are better off submitting their endorsements to the requesting hypervisor. On the other hand, if the accuracy is low, bootstrappers are better off refraining from submitting false endorsements (thanks to the payment mechanism described in the following). This voluntary aspect of the bootstrapping process is necessary to guarantee the fairness for both bootstrappers (in terms of payments and/or resource availabilities) and bootstrapped (in terms of endorsements' precision) parties.

The endorsements from the different bootstrappers are then aggregated using the Bayesian inference equation (Equation (37)) in order to avoid biased endorsements. In this case, the prior beliefs in all the newly deployed VMs would be all set to $\frac{1}{2}$ (i.e., $\gamma = \frac{1}{2}$),

where this expresses a neutral belief between trust and distrust. Finally, bootstrappers receive payments from the boostrapping requestor for having helped it construct trust beliefs. The payment for bootstrappers is given again in terms of additional inquiry requests they can make from the bootstrapping requestor as per Equation (38). This payment mechanism is important to (1) stimulate the hypervisors/VMs that enjoy high classification accuracy rates to get involved in the bootsrapping process so as to receive payments; and (2) discourage the malicious hypervisors/VMs from offering bogus endorsements to illegally promote/demote some VMs.

6.3 Determining the Optimal Detection Load Distribution Strategy: Trust-based Maxmin Game

Having computed the trust scores, we can now proceed with designing the utility functions of both the hypervisor and DDoS attackers and modelling the trust-based maxmin game. The utility of a hypervisor h quantifies its success in protecting the monitored virtual machines V, of worth $W(v)$ each, inversely proportional to h's belief in each v's trustworthiness. The utility function of h at time t_2+1 that comes after the considered window of time $[t_1,t_2]$ is computed as follows:

$$U_{t_2+1}(h) = \sum_{v \in V} \frac{W(v) \times \beta_{[t_1,t_2]}}{Belief_h^v}, \tag{39}$$

where $W(v)$ represents the worth of each virtual machine v (e.g., price, criticality of the applications running inside it), $Belief_h^v$ denotes the belief of h in v's trustworthiness, and $\beta_{[t_1,t_2]}$ is the average detection rate of the IDS agent running on h during the time window $[t_1,t_2]$ and is computed as per Equation (40).

$$\beta_{[t_1,t_2]} = 1 - \sum_{x=t_1}^{t_2} \sum_{v \in V} \frac{(q_x(v) - p_x(v))}{t_2 - t_1} \text{ for each } q_x(v) > p_x(v), \tag{40}$$

where $p_x(v)$ (respectively $q_x(v)$) is the value of $p(v)$ ($q(v)$) at time x.

The idea behind dividing by the trustworthiness belief in the utility function is to make the utility of the hypervisor increase when the belief in a certain VM's trustworthiness decreases and vice versa. In this way, the hypervisor would pay more attention to those VMs that it believes are less trusted when deciding about the optimal detection load distribution strategy. This adds a learning component to the game and aids the hypervisor hence to optimize its detection load distribution strategy. It is worth mentioning that all the calculations in the rest of this chapter are done at time t_2+1 (i.e., the current time for the hypervisor).

Thus, we simplify the notation and use $U(h)$ instead of $U_{t_2+1}(h)$ when referring to hypervisor's h utility at time t_2+1. The payoff of the attacker a represents the loss incurred to the hypervisor as a result of a successful attack. Therefore, the payoff of the attacker is the negation of the hypervisor's payoff, i.e.,

$$U(a) = -U(h) \tag{41}$$

This forms a hypervisor-attacker (two-player) zero-sum game wherein one player's gain is equivalent to the other player's loss.

Definition 12 (**Hypervisor-attacker Zero-sum Game).** *A hypervisor-attacker zero-sum game is a tuple $G = \langle h, a, P_V, Q_V, U(h) \rangle$, where:*

- *h: denotes the hypervisor (i.e., the first player).*
- *a: denotes the attacker (i.e., the second player).*
- *P_V: denotes the probability distribution vector of the detection load over the set V of VMs hosted on top of h (i.e., the mixed strategy of h).*
- *Q_V: denotes the probability distribution vector of the attack over the set V of VMs hosted on top of h (i.e., the mixed strategy of a).*
- *$U(h)$: the utility function of the hypervisor h.*

The objective of the attacker is to choose its probability distribution Q_V for distributing the DoS attack over the VMs' set with the aim of minimizing the hypervisor's detection probability and hence minimizing the latter's payoff, i.e.,

$$\arg\min_{Q_V} U(h) \tag{42}$$

Knowing this fact, the hypervisor would choose a probability distribution P_V over the set of VMs in such a way to maximize the attacker's minimization, i.e.,

$$\arg\max_{P_V} \min_{Q_V} U(h) \tag{43}$$

This forms a maxmin game wherein the attacker tries to minimize the hypervisor's probability of detecting his attacks by distributing each attack over multiple VMs, whereas the

hypervisor tries to maximize this minimization by choosing the optimal distribution of detection load over the VMs.

Definition 13 (**Hypervisor's Maxmin Strategy).** *The maxmin strategy for the hypervisor* h *is* $\arg\max_{P_V} \min_{Q_V} U(h)$ *and the maxmin value for* h *is* $\max_{P_V} \min_{Q_V} U(h)$.

The solution of the game can be devised using Linear Programming (LP), referred to as the problem of determining the values of some real variables for the purpose of minimizing or maximizing a linear function (the objective function) subject to linear constraints on these variables. To this end, let us consider the problem first from the point of view of the hypervisor trying to maximize the minimum of the attacker and let us rewrite Equation (43) as follows:

$$\begin{aligned} \text{maximize} \quad & \min_{Q_V} \sum_{v \in V} p(v) \times U(h) \\ \text{subject to} \quad & \sum_{v \in V} p(v) = 1, \\ & p(v) \ \geq \ 0, \ for\ all\ v \in V. \end{aligned} \tag{44}$$

By inspecting Equation (44), we can notice that the objective function is not linear in the p's owing to the presence of the *min* operator. Therefore, the problem in its current form cannot be solved using LP. To linearize it, we define a variable f such that $f \leq \min_{Q_V} \sum_{v \in V} p(v) \times U(h)$ and try to make f as large as possible subject to this new constraint. Thus, the problem is turned into choosing f and $\sum_{v \in V} p(v)$ to:

$$\begin{aligned} \text{maximize} \quad & f \\ \text{subject to} \quad & f \leq \sum_{v \in V} p(v) \times U(h), \\ & p(v_1) + \cdots + p(v_l) = 1, \\ & p(v) \ \geq \ 0, \ for\ all\ v \in V. \end{aligned} \tag{45}$$

Intuitively, this means that the hypervisor, by choosing its mixed strategy $p(v) \in P_V$, is trying to make as large as possible the minimum that the attacker is attempting to inflict by playing his mixed strategy $q(v) \in Q_V$. To ease the computations, we transform the LP presented in Equation (45) into a simpler form. Assume that $f > 0$ and let $x(v) = \frac{p(v)}{f}$. The constraint $p(v_1) + \cdots + p(v_l) = 1$ becomes then $x(v_1) + \cdots + x(v_l) = 1/f$. Since maximizing f is equivalent to minimizing f's reciprocal $1/f$, we can get rid of f in our problem by rather minimizing $x(v_1) + ... + x(v_l)$. Thus, the problem becomes: choose $x(v_1) + \cdots + x(v_l)$ to:

$$\begin{aligned} \text{minimize} \quad & x(v_1) + \cdots + x(v_l) \\ \text{subject to} \quad & 1 \leq \sum_{v \in V} x(v) \times U(h), \\ & x(v) \ \geq \ 0, \ for\ all\ v \in V. \end{aligned} \tag{46}$$

The above problem may be solved in polynomial time using the simplex method for solving Linear Programming, which is known for its fast performance [40]. Having solved the problem, the hypervisor's optimal strategy would be $p(v) = f \times x(v)$ for each $v \in V$.

If we consider the problem from the attacker's point of view, the latter's objective is to minimize the hypervisor's maximal probability of detection.

Definition 14 (**Attacker's Minimax Strategy).** *The minimax strategy for the attacker* a *is* $\arg\min_{Q_V} \max_{P_V} U(h)$ *and the minimax value for* a *is* $\min_{Q_V} \max_{P_V} U(h)$.

The problem can be written as follows:

$$\begin{aligned} \text{minimize} \quad & \max_{P_V} \sum_{v \in V} q(v) \times U(h) \\ \text{subject to} \quad & \sum_{v \in V} q(v) = 1, \\ & q(v) \ \geq \ 0, \ for\ all\ v \in V. \end{aligned} \tag{47}$$

Using the same logic of transformation followed for the hypervisor's maximization problem, the problem in Equation (47) can be rewritten as:

$$\begin{aligned} \text{minimize} \quad & g \\ \text{subject to} \quad & g \geq \sum_{v \in V} q(v) \times U(h), \\ & q(v_1) + ... + q(v_l) = 1, \\ & q(v) \ \geq \ 0, \ for\ all\ v \in V. \end{aligned} \tag{48}$$

By carefully examining Equation (45) and Equation (48), we notice that the two programs are dual. Following the duality theorem [79], the maximum that the hypervisor can realize in Equation (48) is equivalent to the minimum that the attacker can achieve in Equation (45).

6.4 Numerical Example

Consider a hypervisor h hosting, at time t, three VMs v_1, v_2, and v_3. The prices of these VMs are \$5.33, \$5.24, and \$6.86 respectively. Suppose as well that the hypervisor's detection probability at time t is 0.85, which means that the hypervisor was able to detect up to 85% of the attacks that targeted the cloud system during the time interval $[t_1,t]$. The first step would be to build trust relationships between the hypervisor and its three guest VMs. Assume that the hypervisor performs a monitoring process for the VMs' performance using Algorithm 5 and that the monitoring process results in the following initial trust beliefs: $InitialBelief_h^{v_1} = 0.66$, $InitialBelief_h^{v_2} = 0.32$, and $InitialBelief_h^{v_3} = 0.68$. Note that v_2 is (largely) suspected initially to be launching DDoS attacks. To alleviate the uncertainty and come up with final beliefs, the hypervisor asks three other sources (i.e., VMs and hypervisors), say s_1, s_2, and s_3, about each VM's past behavior.

For v_1, suppose that the trust recommendations from the three sources are: $R_{s_1}^{v_1} = 0.66$, $R_{s_2}^{v_1} = 0.43$, and $R_{s_3}^{v_1} = 0.78$. By applying Equation (37) to compute the final belief in v_1's trustworthiness, we get: $Belief_h^{v_1} = \frac{1.87+3\times0.66}{2\times3} = 0.642$. For v_2, suppose that v_2 is actually malicious and that s_1 and s_2 colluded to give v_2 high (good) recommendation scores, while s_3 gives a honest recommendation. Let the trust recommendations from the three sources regarding v_2 be: $R_{s_1}^{v_2} = 0.77$, $R_{s_2}^{v_2} = 0.61$, and $R_{s_3}^{v_2} = 0.40$. By applying Equation (37) to compute the final belief in v_2's trustworthiness, we get: $Belief_h^{v_2} = \frac{1.78+3\times0.32}{2\times3} = 0.456$. Note that although both s_1 and s_3 colluded to give v_2 high recommendation scores, the final belief of h in v_2's trustworthiness is still low (i.e., 0.456), which reveals that our trust model is quite resilient to the collusion attacks even when attackers form the majority. This is the case because our model combines objective and subjective sources to maximize the accuracy of the final trust results. For v_3, suppose that the trust recommendations from the three sources are: $R_{s_1}^{v_3} = 0.66$, $R_{s_2}^{v_3} = 0.59$, and $R_{s_3}^{v_3} = 0.63$. By applying Equation (37) to compute the final belief in v_3's trustworthiness, we get: $Belief_h^{v_3} = \frac{1.88+3\times0.68}{2\times3} = 0.653$. Suppose now that the saturation factor α is set to 1.5 and that all of s_1, s_2 and s_3 were allowed initially to make 2 inquiries from h (i.e., $Inq(s_1 \rightarrow h) = 2$, $Inq(s_2 \rightarrow h) = 2$, and $Inq(s_3 \rightarrow h) = 2$). As rewards for recommending VMs to h, these three sources receive payments in the form of additional inquiries that they can make from h. For recommending v_1 and as per Equation (38), the number of additional inquiries s_1 can make from h increases up to $\frac{2}{|0.66-0.642|\times1.5} = 74$, the number of additional inquiries s_2 can make from h increases up to $\frac{2}{|0.43-0.642|\times1.5} = 6$, and the number of additional inquiries s_3 can make from h increases up to $\frac{2}{|0.78-0.642|\times1.5} = 10$. The same logic of payment calculation applies also as to recommending v_2 and v_3. It is worth noticing that s_1 whose recommendation score nearly agrees with h's final belief receives a large amount of inquiries compared to s_2 and s_3 whose recommendation scores diverge somewhat from that belief.

By employing Equation (39) and Equation (41) for deriving the utility values of the hypervisor and attacker respectively, we obtain the following game matrix:

$$U = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 7.06 & -9.77 & -8.929 \\ -7.06 & 9.77 & -8.929 \\ -7.06 & -9.77 & 8.929 \end{pmatrix} \end{array}$$

In this matrix, the row represents the hypervisor and the column represents the attacker. Given that we consider a zero-sum game wherein the attacker's gain is equal to the hypervisor's loss and vice versa, we present only the hypervisor's utility in order to simplify notations. Thus, $U(i,j)$ would denote the hypervisor's utility when it is monitoring v_i while the attacker is launching its attack through v_j, whereas $-U(i,j)$ would denote the attacker's utility in that case. For instance, when the hypervisor monitors v_1 while the attacker is attacking through v_1, the hypervisor would gain $U(1,1) = \frac{5.33 \times 0.85}{0.642} = 7.06$ for having been successful in protecting v_1 and the attacker would lose 7.06 for its unsuccessful attack. Contrariwise, when the hypervisor monitors v_1 while the attacker is attacking through v_2, then the former would lose $U(1,2) = \frac{5.24 \times 0.85}{0.456} = 9.77$ for being unsuccessful in protecting v_2, while the latter gains 9.77 for having his attack successful.

Having represented the problem, the next step is to determine the optimal detection load distribution using the simplex technique. This is done by following the subsequent steps:

Step 1**:** *Add a constant to all the matrix's entries, if necessary, to make sure that all the entries are non-negative*.

In order to make all U's elements non-negative, we need to add 9.77 to each entry. This makes the game matrix become:

$$U' = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 16.8300 & 0 & 0.8410 \\ 2.7100 & 19.5400 & 0.8410 \\ 2.7100 & 0 & 18.6990 \end{pmatrix} \end{array}$$

Step 2**:** *Create a tableau* T *by (1) extending the matrix with a border of* $+1'$*s along the right edge,* $-1'$*s along the lower edge, and zero in the lower right corner and (2) labelling the hypervisor's strategies on the left from* x_1 *to* x_m *and those of the attacker on the top from* y_1 *to* y_n.

After applying step 2, we obtain:

	y_1	y_2	y_3	
x_1	16.8300	0	0.8410	1
x_2	2.7100	19.5400	0.8410	1
x_3	2.7100	0	18.6990	1
	-1	-1	-1	0

Step 3**:** *Select the pivot (an element in the Simplex Tableau that is central in the pivot operation which is performed to change the basic solution) belonging to row "a" and column "b" subject to the following properties:*

1. *The border number in the lower edge of the pivot's column "b" must be negative.*
2. *The pivot $T(a,b)$ must be positive.*
3. *The pivot should belong to the row giving the smallest ratio (of the border number in right edge to the pivot) among all the positive entries in the pivot column.*

Since there exists negative elements in all of T's three columns, we can choose any of these columns to be the pivot column. Let's select column 1. Given that the pivot has to be positive, then the selection space is restricted to the first three rows. To determine the pivot, we compute the ratio (of the border number in right edge to the pivot) for the elements in the first three rows to learn about the element that gives the smallest ratio. The ratios for the three elements are 0.0594, 0.3690, 0.3690 respectively. Since $0.0594 < 0.3690$, then the first element is selected to be the pivot, i.e., $T(1,1) = 16.8300$.

Step 4**:** *Perform the pivoting steps as follows:*

1. *Substitute the pivot value with its reciprocal.*
2. *Substitute each element in the pivot row, except for the pivot, with its value divided by the value of the pivot.*
3. *Substitute each element in the pivot column, except for the pivot, with the negative of its value divided by the value of the pivot.*
4. *Substitute each element $T(i,j)$ not belonging neither to the pivot row nor to the pivot column with $T(i,j) - T(a,j) \times T(i,b)/T(a,b)$.*

Step 5**:** *Substitute the label of the pivot row with that of the pivot column and vice versa.*

After applying steps 4 and 5, the tableau would become:

Step 6**:** *Check whether there exists any negative number remaining in the lower border row. If so, return to step* 3*; otherwise, jump to step* 7.

	x_1	y_2	y_3	
y_1	0.0594	0	0.05	0.0594
x_2	−0.1610	19.54	0.7056	0.8390
x_3	0.1610	0	18.5636	0.839
	0.0594	−1	−0.95	0.0594

Since the lower border row still contains two negative entries, we return back to step 3 and execute the pivoting process again. This process gets repeated until having all the elements in the lower border row non-negative. Once this condition is fulfilled, the tableau becomes:

	x_1	x_2	x_3	
y_1	0.0599	0	−0.0027	0.0572
y_2	−0.0079	0.0512	−0.0019	0.0413
y_3	−0.008	0	0.0539	0.0452
	0.0432	0.0512	0.0492	0.1437

We can now go forward with step 7 since all the entries in the lower border row are at this stage non-negative.

Step 7**:** *The solution is determined as follows:*

1. *The optimal strategy of the hypervisor is (1) zero for the hypervisor's variables that end up on the left side, and (2) the value of the bottom edge in the same column divided by the lower right corner for those that end up on the top.*
2. *The attacker's optimal strategy is (1) zero for the attacker's variables that end up on the top, and (2) the value of the right edge in the same row divided by the lower right corner for those that end up on the left.*

In our case, the optimal detection load probability distribution of the hypervisor over its guest VMs would be:

- $p(v_1) = 0.0432/0.1437 = 0.3011$,
- $p(v_2) = 0.0512/0.1437 = 0.3562$, and
- $p(v_3) = 0.0492/0.1437 = 0.3427$.

On the other hand, the optimal attack probability distribution of the attacker over the VMs would be:

- $q(v_1) = 0.0572/0.1437 = 0.3979$,

- $q(v_2) = 0.0413/0.1437 = 0.2875$, and
- $q(v_3) = 0.0452/0.1437 = 0.3146$.

Following these calculations, the hypervisor's optimal strategy is to assign (in real-time) 30.11% of the detection load to v_1, 35.62% to v_2, and 34.27% to v_3. On the other hand, the attacker's optimal strategy is to distribute the DoS attacks over VMs as follows: 39.79% for v_1, 28.75% for v_2, and 31.46% for v_3.

6.5 Experimental Results and Analysis

In this section, we describe the experimental setup and present experimental results by comparing our solution with a benchmark consisting of the *price-based maxmin* [143] and the *fair allocation* [154, 159] detection load distribution strategies.

6.5.1 Experimental Setup

We provide experimental results to test the performance of our model and validate the theoretical and numerical results obtained in the previous sections. The objective of these experiments is three-fold. First, we aim to study how effective the proposed model is in terms of augmenting the attack detection and minimizing both the false positives and negatives. Second, we verify that applying our solution under DDoS attack environments contributes in minimizing the CPU, memory, and network bandwidth wastage. Third, we aim to test the efficiency of our solution in terms of execution time. To these ends, we conduct our experiments using CloudSim [28] in a 64-bit Windows 7 environment on a machine equipped with an Intel Core i7-4790 CPU 3.60 GHz Processor and 16 GB RAM. CloudSim is a cloud simulation tool that has witnessed in the past few years a growing recognition among both academic and industrial milieux. It provides several features that help mimic realistic cloud environments by enabling the simulation of (1) large-scale cloud environments including co-hosted virtualized services; (2) service provisioning and resources allocation policies; (3) network connections among the different cloud components; and (4) federated cloud environments that inter-network resources from both private and public domains [28]. We decided to simulate our own cloud instead of using rented resources from one of the existing cloud providers for the two following main reasons. First, most of the cloud providers such as Amazon EC2 have restriction rules regarding any security testing on their resources and systems, where all the large cloud providers list DoS testing as a non-permissible activity [124]. Second, no cloud provider offers its users direct access

to the VMs' host, which makes inspecting performance information at the host level quite difficult to perform [124].

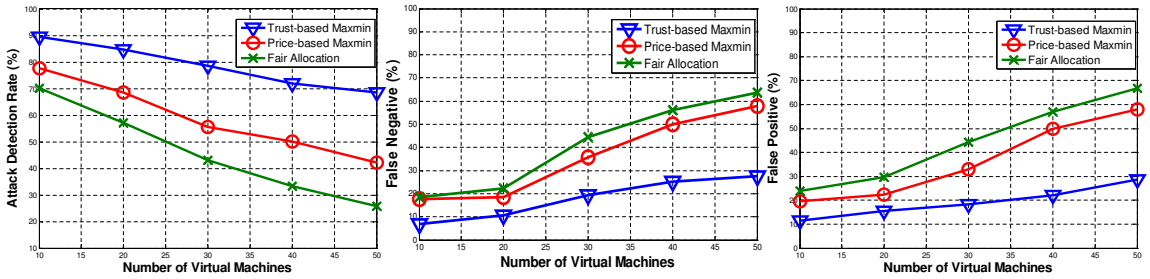
To build our cloud, we create a datacenter whose VMs' configuration is inspired by Amazon EC2 X-large instances[3]. Practically, the created datacenter hosts five physical machines each of which is assigned with a number of VMs varying from 10 to 50 of image size amounting to 10000 MB each. Every VM is equipped with 5-core CPU of 1000 Millions of Instructions Per Second (MIPS) each. Each VM has a memory RAM capacity of 16 GB, hard drive storage of 976.5625 GB, and network bandwidth share of 50000 Kbit/s. In the created datacenter, x86 has been used as a system architecture, Linux as an operating system, and Xen as a Virtual Machine Monitor (VMM). The properties of the datacenter and VMs are summarized in Table 6.2. The VMs are given a set of CPU-intensive tasks (i.e., cryptographic operations and scientific computations). The properties of the tasks have been populated from SPECjvm2008 [9], a standard benchmark suite for Java virtual machines.

Table 6.2: Datacenter properties

Parameter	Value
Number of physical hosts	5
System architecture	x86
Operating system	Linux
Virtual Machine Monitor	Xen
Number of VMs	10, 20, 30, 40, and 50
Number of CPU cores per VM	5
CPU speed per VM	1000 MIPS
RAM memory per VM	16 GB
Hard drive storage per VM	976.5625 GB
Network bandwidth share per VM	50000 Kbit/s

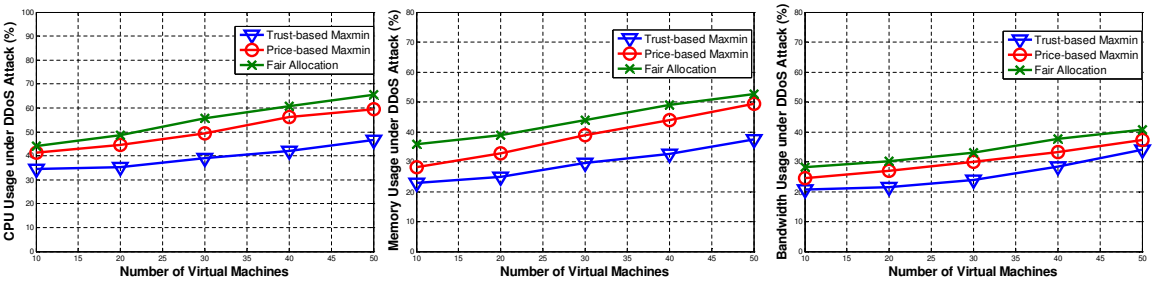
To populate the trust recommendations regarding VMs, we resort to the use of the *Epinions* data set[4] that has been long used in cloud computing and many other domains for representing trust [103, 33]. The data set comprises 664,824 ratings given by 49,290 agents on 139,738 items. The prices of the VMs that are used along with the trust scores to compute the utility functions of both the hypervisor and attackers have been populated from the Amazon EC2 pricing dataset[5]. We compare our model against a benchmark consisting of two other models, namely the *Price-based Maxmin* [143] and *Fair Allocation* [154, 159]. Similar to our model, the *Price-based Maxmin* employs a maxmin game to derive the optimal detection load distribution. However, unlike our solution, this model

[3]https://aws.amazon.com/ec2/details/
[4]https://snap.stanford.edu/data/soc-Epinions1.html
[5]http://aws.amazon.com/ec2/pricing/



(a) Attack Detection *(b) False Negative* *(c) False Positive*

Figure 6.3: Detection performance: Our model increases the percentage of detected attacks and decreases the percentages of false negatives and resources wastage compared to the price-based maxmin and the fair allocation strategy



(a) CPU Usage *(b) Memory Usage* *(c) Bandwidth Usage*

Figure 6.4: Resources usage: Our model minimizes the CPU, memory, and network bandwidth usage under DDoS attack compared to the price-based maxmin and the fair allocation strategy

considers the worth of the VMs (concretized as the VMs' prices) solely in the formulation of the problem. Our model considers, in addition to the worth, the trust scores of the VMs believed by the hypervisor. The fair allocation model, on the other hand, distributes the detection load in an equal fair manner among all VMs. Note that we have selected the fair allocation model to compare with since it is the commonly used allocation strategy for cloud resources in the domain of cloud computing [154, 159]. We have adapted the resources fair allocation model to the detection load distribution problem as we were not able to find, through our extensive literature review investigation, any detection load distribution strategy in the domain of cloud computing other than our previous *Price-based Maxmin* strategy [143].

6.5.2 Experimental Results

In Fig. 6.3, we study how effective the proposed model is in improving the detection performance metrics, namely attack detection, false positive and false negative percentages compared to the price-based maxmin and fair allocation models. Fig. 6.3a reveals that our solution along with the price-based maxmin outperforms the fair allocation model in terms of attack detection (this is the case as well for the rest of the parameters as will be shown later). The reason is that the two former models consider the attacker's strategy when deciding about the detection load distribution strategy among VMs contrary to the fair allocation model wherein the distribution is done by considering the IDS's perspective only. Moreover, Fig. 6.3a shows that the trust-based maxmin performs better than the price-based maxmin model in increasing the detection of the attacks. This is because our trust-based solution allows the hypervisor to learn about the behavior of the VMs over the time, which enables it to adjust the detection load distribution in such a way that assigns more load to the VMs that have a large number of misbehavior during their past history. Besides the incremental learning property that our solution offers to the hypervisor, the fact that trust is a private relationship between the hypervisor and VMs reduces the possibility of the attacker to predict the potential detection load distribution strategy that will be adopted by the hypervisor (i.e., the attacker is unable to know which VMs are the most trusted by the hypervisor to attack through). Per contra, in the price-based model, the attacker may anticipate, for example, that the hypervisor would dedicate more detection load to the more valuable VMs (knowing that the VMs' prices are publicly available for users) and adjust hence its attack distribution strategy accordingly.

In Fig. 6.3b, we measure the false negative percentage that quantifies the percentage of attacks that the system was not able to capture during the detection process. This percentage is computed by subtracting the probability distributions of the attacker from those of the hypervisor when the values of the former are greater. The average false negative rate $\alpha_{[t_1,t_2]}$ during the time window $[t_1,t_2]$ is computed as follows:

$$\alpha_{[t_1,t_2]} = \sum_{x=t_1}^{t_2} \sum_{v \in V} \frac{(q_x(v) - p_x(v))}{t_2 - t_1} \text{ for each } q_x(v) > p_x(v). \tag{49}$$

The false negative rate in the example given in Section 6.4 would be: $\alpha = [(0.3979 - 0.3011)] = 0.0968$. Thus, the percentage of false negatives entailed by our model in the given example is: $0.0968 \times 100 = 9.68\%$. Fig. 6.3b shows that our model diminishes the false negative of the IDS for the same arguments explained in the context of attack detection. Intuitively, decreasing false negatives is an automatic result of increasing attacks detection.

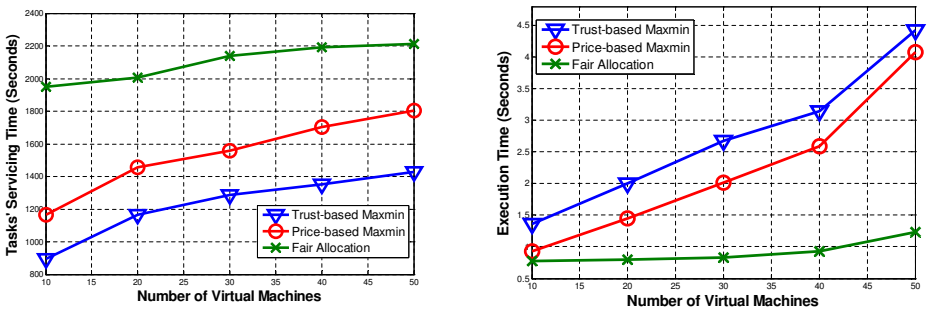
In Fig. 6.3c, we measure the percentage of false positive incurred by the studied detection models. This metric is of prime importance in our case since it can tell us the percentage of resources wasted by the system during the detection process. Intuitively, false positive measures in our case the percentage of resources spent by the hypervisor in monitoring the VMs while these VMs are not sending any attack fragment. It is obtained by subtracting the probability distributions of the hypervisor from those of the attacker when the values of the former are greater. The average rate of resources wasted $\gamma_{[t_1,t_2]}$ during the time window $[t_1,t_2]$ is computed as follows:

$$\gamma_{[t_1,t_2]} = \sum_{x=t_1}^{t_2} \sum_{v \in V} \frac{(p_x(v) - q_x(v))}{t_2 - t_1} \text{ for each } p_x(v) > q_x(v), \tag{50}$$

The false positive rate in the example given in Section 6.4 would be: $\gamma = [(0.3562 - 0.2875) + (0.3427 - 0.3146)] = 0.0968$. Thus, the percentage of false positives entailed by our model in the given example is: $0.0968 \times 100 = 9.68\%$. Fig. 6.3c shows that our model is able to considerably reduce the percentage of false positives compared to the other two models. This is justified by the fact that our model guides the hypervisor on the optimal distribution of detection load that best synchronises with the attacker's probability distributions of the DDoS attacks.

Fig. 6.4 is introduced to study the effectiveness of the proposed model in minimizing the cloud system's resources consumption when this system faces DDoS attack scenarios. Fig. 6.4a measures the CPU consumption of the cloud system under DDoS attacks. Practically, we measure the percentage of CPU that is being consumed by the VMs in the presence of both legitimate requests coming from well-behaving VMs and fake (malicious) requests coming from malicious or compromised VMs. Fig. 6.4a reveals that using our proposed model minimizes the cloud system's CPU consumption under DDoS. Indeed, the fact that our model augments the attack detection and decreases the false positive and negative percentages enables the hypervisor to optimize its resources allocation strategy by limiting the CPU portion assigned to the VMs that are detected to be generating malicious requests, which helps thus save the system's resources and restrict the wastage. Similarly, Fig. 6.4b demonstrates that applying our solution minimizes the cloud system's memory consumption compared to the price-based maxmin and the fair allocation models. In fact, by enhancing the hypervisor's ability to recognize the malicious requests, our model mitigates the load put on the system's memory by these requests. Fig. 6.4c shows that our solution reduces the network bandwidth's consumption compared to the other two models. This is due to the effectiveness of our model in identifying attacks, which reduces the flux of the malicious traffic on the datacenter's network.

Moreover, it is worth noticing from Fig. 6.3 and Fig. 6.4 that the performance of the different models either in terms of detection performance metrics (attack detection, false



(a) Tasks' Servicing Time *(b) Execution Time*

Figure 6.5: Servicing and execution times: Our model reduces the tasks' servicing time compared to the price-based maxmin and the fair allocation strategy and is efficient in terms of execution time

positive, and false negative) or in terms of resources consumption (CPU, memory, and bandwidth) decreases with the increase in the number of VMs. This fact is expected since the more the VMs deployed in a cloud infrastructure, the higher is the freedom given to the attacker to divide its attacks over a greater number of VMs and the less is the detection load that the hypervisor might be able to dedicate for each single VM. Though, our model is still far more resilient to a larger number of deployed VMs due to the advantages explained earlier that our solution brings, which supports the scalability of our model.

We study in Fig. 6.5a the average time taken by the VMs to respond to the assigned tasks. This metric is obtained by subtracting the tasks' end time from the tasks' start time for each VM and averaging these times over all the cloud system's VMs. The results demonstrate that our model aids in reducing the servicing time compared to the price-based maxmin and fair allocation models. The reason is that our model is able to maximize the detection of the DDoS attacks, which helps reduce the congestion on the cloud system's resources and assists hence the legitimate tasks in being accomplished in a more efficient fashion.

Finally, Fig. 6.5b measures the efficiency of the three studied models in terms of their execution time with the variation in the number of VMs. The results reveal that the fair allocation model performs faster than both our model and the price-based model. This result is expected since the fair allocation model divides simply the detection load equally among VMs, which alleviates the time spent on computing the optimal detection load distribution. Moreover, the price-based model performs a bit faster than our model. This time difference may be thought of as the time needed by our model to collect and compute the trust scores for the VMs and adding these scores to the utility functions. In addition, we can notice from the results that our model takes around 4.4 seconds to compute the optimal detection load distribution for a cloud system consisting of 50 VMs, which boosts the

efficiency of our solution. Moreover, by inspecting Fig. 6.5b, we can notice that the time complexity of the model grows polynomially with the increase in the number of VMs, which upholds the feasibility of our model in large-scale cloud datacenters.

6.6 Conclusion

In this chapter, we tackled the problem of maximizing the detection of VM-based DDoS attacks in cloud systems. For this purpose, we proposed first a trust model that combines objective (monitoring) and subjective (recommendations) trust sources and employs the Bayesian inference to aggregate them so as to build credible trust relationships between the hypervisor and guest VMs. On top of this model, we introduced and solved a trust-based hypervisor-attacker maxmin game wherein the hypervisor seeks to maximize the detection probability under a limited budget of resources, knowing that the attacker is trying to minimize this maximization by intelligently distributing the DoS attacks over several VMs. By solving the game, the hypervisor learns about the optimal distribution strategy of detection load among VMs that maximizes the detection of DDoS attacks. Promisingly, a series of experimental comparisons with a benchmark consisting of the price-based maxmin and fair allocation detection load distribution strategies reveal that our solution maximizes the detection of DDoS attacks up to $\approx 26\%$ and minimizes the false positives and negatives by $\approx 20\%$. Moreover, our solution proves to be able to minimize the cloud system's CPU consumption by $\approx 15\%$, memory consumption by $\approx 11\%$, and network bandwidth consumption by $\approx 5\%$ under DDoS scenarios. Lastly, the proposed solution performs efficiently in large-scale data centers, where it takes $\approx 4.4s$ to run in a cloud system consisting of 50 co-hosted VMs. Thus, we have partially achieved our third research objective (**Objective** 3) discussed in Chapter 1, which aimed at maximizing the detection of active malicious attacks using a limited budget of resources.

In the next chapter, we complement this work by putting forward an intelligent detection approach that is able to detect not only DDoS attacks but also multi-type simultaneous attacks targeting the cloud system. The proposed approach copes as well with the challenge of encountering smart attackers that continuously monitor the detection strategies of the cloud system to adjust their attack plans accordingly. Moreover, an attack prevention mechanism that is able to protect cloud-based services from being successful targets for attackers is also discussed in the next chapter.

Chapter 7

Resource-Aware Detection and Defense System Against Multi-Type Attacks in the Cloud

Cloud-based systems are subject to various attack types launched by Virtual Machines (VMs) manipulated by attackers having different goals and skills. The existing detection and defense mechanisms might be suitable for simple attack environments but become ineffective when the system faces advanced attack scenarios wherein simultaneous attacks of different types are involved. This is because these mechanisms overlook the attackers' strategies in the detection system's design, ignore the cloud system's resource constraints, and lack sufficient knowledge about the attackers' types and abilities. To address these shortcomings, the goal of this chapter is to develop a comprehensive detection and defense mechanism against multi-type attacks in the cloud. The proposed solution is presented in the form of a repeated Bayesian Stackelberg game that consists of four phases executed repeatedly to provide the cloud system with incremental and continuous learning about the attackers' strategies and objectives and the VMs' actual security status [1].

7.1 Problem Formulation

We illustrate in this section the problem formulation and explain the attack model considered in this chapter.

[1] The content of this chapter is submitted to the *IEEE Transactions on Cloud Computing* journal (Under Second Round of Reviews)

7.1.1 System Model

The system model consists of a set of virtual machines $V = \{v_1, v_2, ..., v_k\}$ hosted on a shared hypervisor. Note that when $i \in \{1, \ldots, k\}$ can be understood from the context, we simply use v instead of v_i. These VMs might be either well-behaving or attacking. Well-behaving VMs are those that aim at doing their jobs smoothly without having the intention to harm neither the cloud system nor other VMs. On the other hand, attacking VMs seek to harm the cloud system and/or other co-hosted VMs by continuously and collaboratively sending malicious code fragments to form distributed malicious attacks. Such VMs might be either (1) malicious in case their real owners create the attacks or (2) compromised in case the source of attacks is a third party that manipulates VMs and injects its malicious code through them[2]. Each attacking VM is of type $y \in Y$, where Y denotes the set of all attackers' types (e.g., privilege escalation attackers, DoS attackers, etc). Knowing this fact, the cloud system has to find the optimal detection strategy that maximizes the detection of such attacks. To do so, the hypervisor, acting on behalf of the cloud system, has a specific amount of resources R that comprises both the amount R_c of resources to be dedicated to serving clients and the amount R_d of resources to be dedicated for intrusion detection such that $R = R_c + R_d$. Thus, the objective of the hypervisor becomes finding the optimal detection load distribution strategy that maximizes the detection of distributed attacks, while respecting the budget R_d of resources. We model this situation as a repeated Bayesian Stackelberg security game of two players, i.e., hypervisor and attackers. The game is played sequentially in the sense that the hypervisor representing the *leader* of the game commits first to a certain detection load distribution strategy and then attackers (*followers* of the game) choose their attack distribution strategy after having observed the hypervisor's move.

Formally, the hypervisor's set of pure strategies consists of a (sub)set of VMs to put the detection load on. To rip off and confuse attackers, the hypervisor would select a mixed strategy (i.e., probability distributions) at each time moment x belonging to the fixed interval of discrete time $[t_1, t_2]$. The mixed strategy consists of the vector $H_x(V) = (h_x(v_1), \ldots, h_x(v_k))$ over the set V of VMs hosted on top of the hypervisor such that $\sum_{v_i \in V} h_x(v_i) = 1$. In this way, the hypervisor would assign a different detection load probability to each of its VMs. The attackers observe the hypervisor's detection load distribution strategy and then decide about their own strategies consisting also of a (sub)set of VMs through which the attack is to be launched. In their turn, attackers might choose a mixed strategy at time moment $x \in [t_1, t_2]$ consisting of the vector $A_x(V) = (a_x(v_1), \ldots, a_x(v_k))$ over the set V of VMs to attack through such that $\sum_{v_i \in V} a_x(v_i) = 1$. In this way, each attacker would pick a VM $v \in V$ at time x with a probability of $a_x(v) \in A_x(V)$ to attack through so as to confuse the cloud system [106].

[2] In the rest of this chapter, the term *attacker* is used to refer to both cases.

Note that attackers may decide not to attack through any VM at a certain time moment.

Based on the strategies adopted by both the hypervisor and attackers, a payoff is assigned to each of these parties. Particularly, when the hypervisor selects the strategy i and the attacker selects the strategy j, the hypervisor receives a payoff of U_{ij} and the attacker receives a payoff of Q_{ij}. The utility of the hypervisor for each VM v facing an attack of type y at time t_2+1 (current system time based on the interval $[t_1,t_2]$) is given by Equation (51).

$$U_{ij}^{v,y}(t_2+1) = \frac{\beta([t_1,t_2]) \times w(v) - mon(v)}{\kappa^y(v)} \tag{51}$$

This payoff represents the success of the hypervisor in protecting its virtual machine v of worth $w(v)$ (the worth of a VM depends mainly on its price as well as its hardware, network, and storage configuration) minus the cost $mon(v)$ of monitoring v; inversely proportional to the degree of damage $\kappa^y(v)$ caused by an attacker of type y assaulting through v. Since the success of detection depends heavily on the IDS's detection probability, the utility of the hypervisor at time t_2+1 is weighed based on its average detection rate $\beta([t_1,t_2])$ during the time window $[t_1,t_2]$, which is computed as per Equation (52).

$$\beta([t_1,t_2]) = 1 - \sum_{x=t_1}^{t_2} \sum_{v \in V} \frac{\lambda}{t_2 - t_1}, \tag{52}$$

$$\lambda = \begin{cases} a_x(v) - h_x(v), & \text{if} \quad a_x(v) > h_x(v) \\ 0, & otherwise \end{cases}$$

In Equation (52), $\sum_{x=t_1}^{t_2} \sum_{v \in V} a_x(v)$ represent the attack distribution probabilities adopted by attackers on every $v \in V$ at each time moment $x < t_2+1$, which are known by the hypervisor using the backward induction reasoning (See Section 7.2). Similarly, $\sum_{x=t_1}^{t_2} \sum_{v \in V} h_x(v)$ represent the detection load probability distributions adopted by the hypervisor on every $v \in V$ at each time moment $x < t_2+1$. Note that all the calculations in the rest of this chapter are done at time t_2+1 (i.e., the current time for the hypervisor and attackers). Thus, we simplify the notation and use $U_{ij}^{v,y}$ (respectively $Q_{ij}^{v,y}$) instead of $U_{ij}^{v,y}(t_2+1)$ ($Q_{ij}^{v,y}(t_2+1)$) when referring to hypervisor's (attacker) utility at time t_2+1.

On the other hand, the attacker's payoff would be:

$$Q_{ij}^{v,y} = w(v) \times \kappa^y(v) - att(v) \tag{53}$$

This payoff quantifies the gain of the attacker from exploiting the VM of worth $w(v)$ proportionally to the damage $\kappa^y(v)$ caused by the attacker of type y assaulting through v

Table 7.1: List of attacks w.r.t the associated vulnerabilities

Attacks	**Vulnerabilities**
Hypervisor's Memory Content Disclosure	Virtual CPUs
Co-hosted VMs' Memory Modification	Soft Memory Management Unit (MMU)
Denial of Service	I/O and networking Interrupt and timer Paravirtualized I/O VM exits
Virtual Machine Destruction	VM Management
Virtual Machine Crash	Hypercalls
Privileges Escalation	Symmetric Multiprocessing (SMP) Remote Management Software Hypervisor Add-ons

minus the cost $att(v)$ incurred by attacking v.

7.1.2 Attack Model

We consider in this work the attacks that occur at the cloud system's virtualization surface which offers attackers with a new appealing security attack vector. Roughly speaking, each functionality provided by the hypervisor (e.g., CPU virtualization, VM management, etc.) can include some vulnerabilities that attackers might exploit to carry out their malicious activities. We discuss in the following some of the attacks that might be exerted against the cloud system w.r.t the corresponding vulnerabilities that might be exploited to carry out such attacks. These attacks have been utilized when performing our experiments as will be explained in Section 7.4. The list of attacks along with their corresponding vulnerabilities are summarized in Table 7.1. These attacks have been inspired by the list of cloud-specific vulnerabilities identified in [112] as a recapitulation of some real vulnerabilities collected from the National Institute of Standards and Technology (NIST)'s National Vulnerability Database (NVD) [5], SecurityFocus [8], Red Hat's Bugzilla [7] and Code Vulnerabilities and Exposures (CVEs) [4]. The attacks include hypervisor's memory content disclosure, co-hosted VMs' memory modification, DoS, virtual machine destruction, virtual machine crash, and privilege escalation. The *hypervisor's memory content disclosure* attackers might exploit vulnerabilities emerging from the virtualization of the CPU and memory operations to reveal some data contained in the hypervisor's physical memory. *Co-hosted VMs' memory modification* attackers take advantage of the memory virtualization process to gain access to the physical host and be able hence to alter the memory kernel-space of some co-resident VMs. In their turn, *DoS* attackers benefit from some vulnerabilities present in the Input/output (I/O) and networking, Interrupt and timer, Paravirtualized I/O, and VM exits to inject some malicious instructions that lead to huge increase in the resources allocated to some VMs. *Virtual machine destruction* attackers

exploit the VMs' management operations (i.e., starting, stopping, and pausing VMs) to inject some malicious code into the management domain leading to destroy some co-hosted VMs. As for the *virtual machine crash* attackers, such attackers strive to manipulate VMs and start embedding random memory physical addresses into their memory management (hyper)calls, which leads to crash the VMs in question. Finally, *privilege escalation* attackers take advantage of some of the vulnerabilities that may be present in the Symmetric Multiprocessing (SMP), Remote Management Software (RMS), and Hypervisor Add-ons to get some unprivileged (Ring 3) malicious processes executed. More details about these attacks are given in the following:

- **Hypervisor's Memory Content Disclosure**: Each guest VM is assigned with a set of virtual CPUs (vCPUs) whose role is to mirror the physical CPU's actions. Poor initialization of the vCPU's data structures might result in very dangerous threats including the disclosure of the hypervisor's memory content. As a real-world attack, a padding field that has not been zeroed-out ended up carrying some information from data structures formerly used by the hypervisor. This enabled attackers to learn about some data contained in the hypervisor's memory.

- **Co-hosted VMs' Memory Modification**: Since guest VMs are not permitted to get direct access to the host's Memory Management Unit (MMU) [112], the hypervisor has to run a soft version of the MMU in order to maintain a shadow page table (data structure used to store mappings of virtual addresses to physical addresses) for each guest VM. In this way, such a soft MMU will intercept each page mapping (from the virtual address provided by the VM to the physical address of the actual memory where needed data is stored) issued by the VM with the aim of adapting shadow page tables appropriately. Unfortunately, vulnerabilities in the underlying soft MMU can be maliciously and dangerously exploited by attackers to unveil and modify data contained in co-hosted VMs' memory segments. As a real-world attack example, the fact that Kernel-based Virtual Machine (KVM)'s emulator used to run in Ring 0 (most privileged) mode when accessing a guest VM's memory, allowed unprivileged (Ring 3) application running inside a VM to gain access to a Memory Mapped I/O (MMIO) region through emulated MMIO instructions. This was exploited to force KVM to execute a malicious instruction that alters the memory kernel-space of the VM in question.

- **Denial-of-Service (DoS)**: To carry out DoS attacks, attackers might take advantage of some vulnerabilities present in the Input/output (I/O) and networking, Interrupt and timer, Paravirtualized I/O, and VM exits. We explain in the following how I/O and networking vulnerabilities can be practically exploited to perform DoS attacks. In

virtualized environments, the hypervisor takes the role of emulating networking and I/O operations. This is usually done through a separation of roles, where emulated drivers are either front-end or back-end. While front-end drivers stay in the guest VMs and have no immediate access to the physical hardware, back-end drivers reside in the hypervisor and have direct access to the host's resources. Thus, a communication between front-end and back-end drivers is required to accomplish the desired operations. The fact that such a process is usually coded in high-level expressive programming languages such as C and $C++$ [112] makes it prone to elaborate attacks such as DoS. As a real-world example, a vulnerability that existed in Xen hypervisors gave paravirtualized front-end drivers the chance to forward a malicious shared framebuffer descriptor that deceived Xen into assigning them large internal buffer inside Dom0, which resulted in a DoS situation.

- **Virtual Machine Destruction**: The hypervisor is responsible for performing the guest VMs' management responsibilities that include starting, stopping, and pausing these VMs. When starting a VM, at each boot up operation, the VM's kernel images have to be decompressed into memory and interpreted by the hypervisor's management domain. This can be exploited by attackers to inject some malicious code into the management domain leading to break down some co-hosted VMs. As a real-world attack example, some attackers took advantage of the fact that Xen's bootloader employed Python's exec() statements to handle kernel's configuration files and were able to change the configuration file inside Dom0 in such a way that destroys another co-hosted VM using the $xm\ destroy$ command.

- **Virtual Machine Crash**: Hypercalls, in their turn, can be manipulated to execute malicious attacks. In fact, hypercalls offer guest VMs with the ability to demand privileged actions from the hypervisor such as CPU and Hard Disk partitions managements. Thus, attackers who take control of some guest VMs can manipulate hypercalls to obtain premium privileges over the host's resources. As a real-world example, KVM used to allow unprivileged (Ring 3) guest VMs to make memory management hypercalls, which was exploited by attackers to inject random memory physical addresses into the memory management hypercall which resulted in crashing the underlying VM.

- **Privilege Escalation**: Privilege escalation attacks might be executed by exploiting some vulnerabilities that may be present in the Symmetric Multiprocessing (SMP), Remote Management Software (e.g., Web applications providing user interfaces for virtualization management), and Hypervisor Add-ons. We illustrate in what follows how vulnerabilities in the SMP property can be practically exploited to launch privileges escalation attacks. Specifically, the SMP property that comes with some guest

VMs allows two or more vCPUs pertaining to one single VM to be scheduled simultaneously to the physical CPU. The vulnerabilities in such a case would emerge from a hypervisor's code making presumptions that are only valid on single-threaded processes but do not hold any more in multi-threaded environments. As a real-world attack example, a bug in the KVM hypervisor allowed malicious unprivileged (Ring 3) processes to exploit a race condition scenario to execute privileged instructions. Particularly, attackers have invoked a legitimate I/O instruction on one thread and tried to replace it with a privileged instruction from another thread right after KVM did the validity check of the first one but before it was executed. This means that the (second) privileged instruction is the one that has been actually executed although (only) the first instruction underwent the validity check.

7.2 Adaptive Detection Load Distribution Strategy: Bayesian Stackelberg Game

We formulate in this section the intrusion detection problem as a Stackelberg security game between the cloud system and attackers. Practically, the hypervisor (acting on behalf of the cloud system) plays the role of the game leader and makes the first move by choosing its detection load distribution strategy over VMs, whereas attackers are the followers that observe the leader's strategy and choose their best responses to it in terms of attack distribution strategies. The game is modeled first as a Mixed-Integer Quadratic Program (MIQP) and then converted into a Mixed-Integer Linear Program (MILP) to be solved using a linear programming solver tool. The backward induction reasoning is employed to determine the optimal strategies of both the cloud system and attackers. This is done by first deriving the best response of the attackers to a (fixed) observed strategy of the cloud system and then integrating this best response to the cloud system's optimization problem to help it select the optimal detection load distribution strategies. Intuitively, this means that the cloud system anticipates that attackers will play their best responses to its (observed) detection load distribution strategy and embeds this knowledge into its optimization problem to select the optimal detection load distribution strategy using this information.

Let L and F denote the index sets of the hypervisor (leader) and attacker's (follower) pure strategies, respectively. Let l represent a vector of the hypervisor's pure strategies (a.k.a hypevisor's policy) and f represent a vector of the attacker's pure strategies (a.k.a attacker's policy). Thus, the value l_i would represent the proportion of times in which the hypervisor plays the pure strategy i from its policy set. Similarly, the value f_j represents the proportion of times in which the attacker plays the pure strategy j from its policy set. Based on the strategies chosen by the hypervisor and attacker and as explained in Section 7.1,

U and Q represent the payoff matrices such that U_{ij} is the gain of the hypervisor and Q_{ij} is the gain of the attacker when the hypervisor selects the pure strategy i and the attacker selects the pure strategy j.

Let us fix first the hypervisor's policy to a certain policy l. After observing l, the attacker needs to solve the following linear programming optimization problem in order to determine its optimal response to l:

$$\begin{aligned} \text{maximize} \quad & \sum_{j \in F} \sum_{i \in L} Q_{ij} \times f_j \times l_i \\ \text{subject to} \quad & \sum_{j \in F} f_j = 1, \\ & f_j \in [0,1], \qquad \forall j \in F \end{aligned} \tag{54}$$

Knowing the fixed strategy l of the leader, the best response $f_j(l)$ of the attacker should yield a non-negative utility to the attacker, which means that Problem (54) has to satisfy the following constraint:

$$f_j \times \sum_{i \in L} Q_{ij} \times l_i \geq 0, \quad \forall j \in F \tag{55}$$

Moreover, given that $f_j(l)$ is the attacker's best response strategy, any deviation from this strategy (i.e., $1 - f_j$) would lead the attacker to undergo a loss in terms of utility. Thus, Problem (54) has to satisfy the following constraint as well:

$$(1 - f_j) \times \sum_{i \in L} Q_{ij} \times l_i \leq 0, \quad \forall j \in F \tag{56}$$

Let's move now to the cloud system's side. The hypervisor, knowing that the attacker will play its best response $f_j(l)$ to every hypervisor's strategy l, incorporates this knowledge into its optimization problem to determine the solution l that maximizes its own payoff. Thus, the hypervisor has to solve the following problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in L} \sum_{j \in F} U_{ij} \times f_j(l) \times l_i \\ \text{subject to} \quad & \sum_{i \in L} l_i = 1, \\ & l_i \in [0,1], \qquad \forall i \in L \end{aligned} \tag{57}$$

Problem (57) can be completed by incorporating the characterization of $f_j(l)$ depicted in Problem (54) and Equations. (55) and (56). Taking into account the fact that given any optimal mixed strategy $f_j(l)$, then all the pure strategies in its support are also optimal [108], we can consider only the optimal pure strategies of the attacker (which exist always)

and symbolize the optimal pure strategies using binary variables. Thus, the hypervisor's problem becomes:

$$
\begin{aligned}
\underset{l,f}{\text{maximize}} \quad & \sum_{i\in L}\sum_{j\in F} U_{ij}\times l_i \times f_j \\
\text{subject to} \quad & \sum_{i\in L} l_i = 1, \\
& \sum_{j\in F} f_j = 1, \\
& (1-f_j)\times \sum_{i\in L} Q_{ij}\times l_i \leq 0, \quad \forall j \in F \\
& f_j \times \sum_{i\in L} Q_{ij}\times l_i \geq 0, \qquad \forall j \in F \\
& l_i \in [0,1], \qquad \forall i \in L \\
& f_j \in \{0,1\}, \qquad \forall j \in F
\end{aligned}
\tag{58}
$$

In order to enhance the decisions of the hypervisor, we incorporate the probability distribution p^y of facing each type $y \in Y$ of attackers into the hypervisor's optimization problem. For example, if the hypervisor learns that the majority of attackers targeting the cloud system at a certain time moment are DoS attackers, then it would adjust its detection load strategy towards assigning more load to the VMs that are suspected to be vulnerable to such attacks. Section 7.3.3 explains how the hypervisor would be able to practically compute p^y. Having embedded the probability distribution p^y, the hypervisor's problem becomes:

$$
\begin{aligned}
\underset{l,f}{\text{maximize}} \quad & \sum_{y\in Y}\sum_{i\in L}\sum_{j\in F} p^y \times U^y_{ij}\times l_i \times f^y_j \\
\text{subject to} \quad & \sum_{i\in L} l_i = 1, \\
& \sum_{j\in F} f^y_j = 1, \qquad \forall y \in Y \\
& (1-f^y_j)\times \sum_{i\in L} Q_{ij}\times l_i \leq 0, \quad \forall j \in F,\ y \in Y \\
& f^y_j \times \sum_{i\in L} Q_{ij}\times l_i \geq 0, \qquad \forall j \in F,\ y \in Y \\
& l_i \in [0,1], \qquad \forall i \in L \\
& f^y_j \in \{0,1\}, \qquad \forall j \in F,\ y \in Y
\end{aligned}
\tag{59}
$$

In Problem (59), the first and fifth constraints compel a feasible mixed policy for the hypervisor, whereas the second and sixth constraints compel a feasible pure strategy for the attacker. The sixth constraint restricts as well the actions' vector of the attacker to be a pure distribution over F. The third and fourth constraints force the best response $f_j(l)$ to be

optimal for the attacker in terms of gained utility. Problem (59) is an integer program with a non-convex quadratic objective [108]. Thus, the final step would be converting the Mixed-Integer Quadratic Programming (MIQP) at hand into a Mixed-Integer Linear Programming (MILP) by removing the non-linearity of the objective function. This can be achieved by assigning the value of $l_i \times f_j^y$ to a new variable z_{ij}^y. Thus, the problem becomes:

$$
\begin{aligned}
\underset{l,f}{\text{maximize}} \quad & \sum_{y\in Y}\sum_{i\in L}\sum_{j\in F} p^y \times U_{ij}^y \times z_{ij}^y \\
\text{subject to} \quad & \sum_{i\in L}\sum_{j\in F} z_{ij}^y = 1, && \forall y \in Y \\
& f_j^y \leq \sum_{i\in L} z_{ij}^y \leq 1, && \forall j \in F,\ y \in Y \\
& \sum_{j\in F} f_j^y = 1, && \forall y \in Y \\
& (1-f_j^y) \times \sum_{i\in L} Q_{ij} \times z_{ij}^y \leq 0 && \forall j \in F,\ y \in Y \\
& f_j^y \times \sum_{i\in L} Q_{ij} \times z_{ij}^y \geq 0 && \forall j \in F,\ y \in Y \\
& z_{ij}^y \in [0,1], && \forall i \in L,\ j \in F,\ y \in Y \\
& f_j^y \in \{0,1\}, && \forall j \in F,\ y \in Y
\end{aligned}
\tag{60}
$$

Having linearized the problem, the MILP in Problem (60) can be now solved using a linear programming solver tool to derive the optimal mixed strategies of the cloud system and attackers [40]. For example, a possible optimal mixed strategy for the cloud system over the set $V = \{v_1, v_2, v_3\}$ of VMs could be to assign 35% of the detection load to v_1, 25% to v_2, and 40% to v_3, when attackers are attacking 30% of times over v_1, 30% of times over v_2, and 40% over v_3.

7.3 Learning-based Detection and Defense System: Repeated Bayesian Stackelberg Game

As depicted in Fig. 7.1, the repeated Stackelberg game consists of four main phases: Bayesian Stackelberg game, virtual machines' risk assessment, services deployments and defense mechanism, and attackers' types recognition technique. These phases run repeatedly at each time unit x of the discrete time window $[t_1, t_2]$. The Bayesian Stackelberg game (described in Section 7.2) computes the optimal probability distributions of the hypervisor's detection load over the guest VMs. To evaluate the effectiveness of the detection strategy, the risk assessment phase enables the hypervisor to conduct an in-depth study on the vulnerabilities and threats that might be present on VMs and to analyze the past

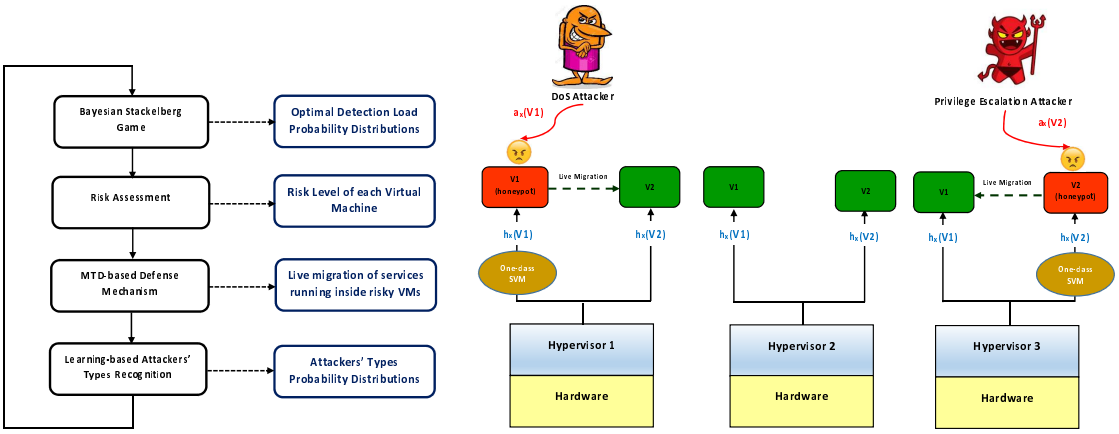


Figure 7.1: Repeated Bayesian Stackelberg Game: The repeated Bayesian Stackelberg game is composed of four main phases: Bayesian Stackelberg game, Risk Assessment, Services Deployment, and Honeypots Deployment

attack history of those VMs to derive the appropriate risk level of each guest VM. Having identified the risky VMs, the goal of the services deployment phase is to advance a defense mechanism that protects services from being successful target for attackers. This is done by offering a live-migration-based [30] decision making framework that allows the hypervisor to migrate services hosted on VMs classified as risky to other safer VMs. Finally, the honeypots deployment phase exploits the idle VMs (running no active services) by deploying honeypots inside them to collect malicious data with the aim of studying and learning the behavior and objectives of the attackers. The collected data is analyzed using a one-class SVM classifier to predict the types of attackers and learn about their probability distributions. This information is used finally to feed the Bayesian Stackelberg game of the next time moment $x+1$ with the probability distributions over the attackers' types to adjust and optimize the hypervisor's detection load distribution strategies. Note that only the first two phases (Bayesian Stackelberg game and risk assessment) have to be continuously repeated (i.e., every time unit of the discrete time window). Specifically, the execution of the rest of the phases is dependent on the output of the risk assessment phase. In other words, if no VMs are suspected to be risky at a certain time unit, there will be no need to proceed with the other subsequent steps at that time moment. In what follows, we explain each phase of the repeated Stackelberg game in detail and provide numerical examples to illustrate how our solutions can be practically applied.

7.3.1 Virtual Machines Risk Assessment

Having computed the optimal detection load distribution strategy using the one-stage Bayesian Stackelberg game described in Section 7.2, the hypervisor assesses in this

phase the risk level of each VM. The methodology used for risk assessment is inspired mainly by that of the NIST [47], which provides a comprehensive guide on how to evaluate the security risk levels of Information Technology (IT) resources. Specifically, the risk level of a VM v is estimated in terms of the **likelihood** of exploiting a specific **vulnerability** that is present on v to exert some **attack** along with the consequent **impact** of that malicious act on v. Formally, the risk level assessment function of VM v at the time moment $x \in [t_1, t_2]$ is calculated as follows:

$$Risk^x(v) = w^x(v) \times \mho^x(v) \times \vartheta^x(v) \tag{61}$$

where $w^x(v)$ is the worth of v at time moment x, $\mho^x(v)$ is the magnitude of impact resulting from the exploit of the vulnerabilities present on v at time moment x, and $\vartheta^x(v)$ is the threat likelihood on v at time moment x. Thus, the first step in assessing the risk levels would be estimating the worth of each virtual machine. The worth is an indicator of the degree of damage that could be entailed by the exercise of a certain attack on the VM. Obviously, the worth of a certain VM is decided on the basis of its current hardware, storage, and networking capabilities (e.g., memory, CPU, bandwidth, etc.). Table 7.2 shows a list of possible worth levels, values, and descriptions that can be used to assess the worths of the VMs.

Table 7.2: Virtual machine worth scale and description

Worth Level	Value	Description
Important	6	The VM has sophisticated hardware, networking, and storage capabilities.
Medium	3	The VM has intermediate hardware, networking, and storage capabilities.
Moderate	1	The VM has simple hardware, networking, and storage capabilities.

The second step in the risk assessment process involves identifying and listing the VM's potential vulnerabilities that attackers might take advantage of to carry out their malicious attacks. In our case, we use the list of vulnerabilities identified in Table 7.1 and that consists of: vCPUs, soft MMU, I/O and networking, interrupt and timer, paravirtualized I/O, VM exits, VM management, hypercalls, SMP, remote management software, and hypervisor add-ons. Table 7.3 shows a list of possible vulnerability levels, values, and descriptions that can be used to assess the impacts of vulnerability exploitations on the VMs. Note that this list can be updated regularly at each time moment x to cover new types of discovered vulnerabilities, where security administrators/experts can periodically feed the hypervisor with an updated list of potential vulnerabilities for each corresponding VM.

Table 7.3: Vulnerability scale and description

Vulnerability Level	Value	Description
High	6	The exploit of the vulnerability results in extremely painful losses for the VM and cloud system as a whole. Such vulnerabilities can include vCPUs, VM management, SMP, paravirtualized I/O and remote management software.
Medium	3	The exploit of the vulnerability results in painful losses for the VM and cloud system. Such vulnerabilities can include soft MMU.
Low	1	The exploit of the vulnerability results in manageable losses for the VM. Such vulnerabilities can include hypercalls.

Having characterized the potential vulnerability exploitation impacts, the third step is to determine the corresponding threats that exploit the identified vulnerabilities to launch attacks against VMs. For our risk assessment process, we restrict the analysis to the list of attacks identified in Table 7.1 and that consists of: hypervisor's memory content disclosure, co-hosted VMs' memory modification, DoS, virtual machine destruction, virtual machine crash, and privilege escalation. Table 7.4 shows a list of some possible threats levels, values, and descriptions that can be used to assess the threat likelihood on the VMs.

Table 7.4: Threat scale and description

Threat Level	Value	Description
High	6	The threat is extremely strong and performed by an expert attacker. Such threats can include hypervisor's memory disclosure, DoS, and privilege escalation.
Medium	3	The threat is strong and performed by a motivated attacker. Such threats can include co-hosted VMs' memory modification and VM destruction.
Low	1	The threat is weak and performed by a non-professional attacker. Such threats can include virtual machines crash.

Now that we have defined the worth, vulnerability, and threat levels, we need to proceed with identifying the risk levels scale to be used as a reference when deciding about the VMs' risk levels. The risk levels scales and descriptions are presented in Table 7.5.

Table 7.5: Risk Scale and description

Risk Level	Risk Scale	Description
High	5-6	There is an urgent need to implement corrective measures (e.g., live migration) to resume the normal operation of the cloud system.
Medium	3-4	There is a need to implement corrective measures within a reasonable period of time to resume the normal operation of the cloud system.
Low	1-2	The risk does not constitute an obstacle to the normal operation of the cloud system.

We are now well-equipped to move forward with the risk levels determination step, where the risk level of each VM is computed using Equation (61) after normalization. We give in Table 7.6 a numerical example that clarifies how to compute and determine the risk levels of three VMs based on the worth, vulnerability, threat, and risk scales defined in Tables 7.2, 7.3, 7.4, and 7.5 respectively.

Table 7.6: Risk levels determination example

VM	Worth	Vulnerability Impact	Threat Likelihood	Risk
v_1	6	1	1	$\frac{6\times1\times1}{216}\times6=0.17$
v_2	6	3	6	$\frac{6\times3\times6}{216}\times6=3$
v_3	6	6	6	$\frac{6\times6\times6}{216}\times6=6$

In Table 7.6, v_1 has to be classified as being low risk (according to Table 7.5), v_2 as moderately risky, and v_3 as highly risky. Note that we multiply by 6 and divide by 216 (i.e., $6\times6\times6$) in Table 7.6 to normalize the computed risk level values [47].

Nonetheless, our risk assessment process is not yet complete. In fact, despite its importance and effectiveness, the above presented risk assessment is generic for all VMs and does not take into account the past history of each VM. Practically, to make the risk analysis more realistic and thorough, we have to consider the past attack history of the VMs in our analysis. Therefore, we propose to integrate the attack growth/decay (growth in case the number of attacks is increasing and decay otherwise) factor $e^{x.k(v)}$ of each VM v at time moment x into our risk level determination formula. Thus, the risk assessment formula presented initially in Equation (61) becomes:

$$Risk^x(v) = w^x(v)\times\mathcal{v}^x(v)\times\vartheta^x(v)\times e^{[(x-1)-(x-2)].k^x(v)}, \tag{62}$$

$$k^x(v) = \frac{\ln(\frac{N^{x-2}(v)}{N^{x-1}(v)})}{[(x-2)-(x-1)]} \tag{63}$$

where $k^x(v)$ is the attack growth/decay rate on v and $N^x(v)$ is the number of times v has been attacked at time moment x. As depicted in Equation (63), the attack growth/decay rate $k^x(v)$ is computed based on the difference between the number of attacks that existed at the two past consecutive time moments $x-1$ and $x-2$. Assume that the risk calculations presented in Table 7.6 were derived at time moment $x=3$ and that v_1 got attacked three times at time moment $x=1$ and five times at time moment $x=2$. We explain in the following how Equation (63) has been derived and how to practically compute the attack growth factor $k^3(v_1)$ of v_1 at time moment $x=3$. Specifically, we have: $5=3\times e^{(2-1).k^3(v_1)} \Rightarrow 5/3 = e^{k^3(v_1)} \Rightarrow \ln(5/3) = \ln(e^{k^3(v_1)}) \Rightarrow \ln(5/3) = k^3(v_1) \Rightarrow k^3(v_1) = \ln(5/3) = 0.511$. Thus, the risk level of v_1 would be updated to become $R^3(v_1) = 0.17 \times e^{0.511\times 1} = 0.283$, where v_1 remains a lowly risky VM.

7.3.2 Services Deployment and Defense Mechanism

In the light of the results obtained from the risk assessment phase, we discuss in this section a Moving Target Defense (MTD)-based [111] services' deployment strategy whose goal is to provide a defense mechanism to protect the services hosted in the cloud system from being successful targets for attackers. Practically, we propose a security-oriented live migration strategy that allows the hypervisor to migrate the services running inside VMs classified as risky to be hosted in other more secure VMs. To do so, the hypervisor has to identify first the set of VMs that might serve as replacements for the risky ones. Apart from security considerations, determining such a set of VMs involves some technical constraints, where the migration process should maintain some technical compatibilities between the migration source and destination VMs. For example, the Operating systems (OSs) of the source and destination VMs have to be consistent since migration between distinct OSs (e.g., Windows and Linux) might entail some technical complications and unanticipated technological roadblocks. Moving to the security perspective, the set of VMs that are eligible to serve as replacements should be evidently selected to be non-risky based on the risk assessment's results.

Formally, let $E^x(v_i)$ denote the set of VMs that are eligible to replace a VM v_i at time moment x. These VMs satisfy thus aforementioned technical constraints and are classified as low risk in the risk assessment phase. Also, let $p^x(v_i \longrightarrow v_j)$ denote the percentage of worth increase between v_i and v_j, which is calculated as per Equation (64).

$$p^x(v_i \longrightarrow v_j) = \begin{cases} \frac{w^x(v_j)-w^x(v_i)}{w^x(v_i)}, & \text{if } \frac{w^x(v_j)-w^x(v_i)}{w^x(v_i)} \geq 0 \\ +\infty, & \text{otherwise} \end{cases} \tag{64}$$

Let v_j^* be the VM that gives the minimum worth increase percentage w.r.t v_i, i.e., $p^x(v_i \longrightarrow$

$v_j^*) = \min(p^x(v_i \longrightarrow v_m)),\ \forall v_m \in E^x(v_i)$. The decision of the hypervisor to migrate a service running inside v_i to another VM $v_j \in E^x(v_i)$ is taken as follows:

- if $E^x(v_i) \neq \emptyset$ and $p^x(v_i \longrightarrow v_j^*) \neq +\infty$, the hypervisor selects the VM v_j^* that gives the least percentage increase in the worth value $p^x(v_i \longrightarrow v_j^*)$ compared to v_i.
- if $E^x(v_i) = \emptyset$ or $p^x(v_i \longrightarrow v_j^*) = +\infty$, then the hypervisor creates a new VM v_z to be the migration destination for the services running in v_i.

The idea behind selecting the VM giving the least worth increase percentage to serve as a replacement is to guarantee that the migrated service will be running in a new environment that is very similar to that it was running inside (before migration) in terms of VM's storage, hardware, and networking configuration. This is because the worth is an indicator of the current hardware, storage, and networking capabilities of the VM. Such a migration decision would help maintain the performance of the service after the migration process. Along the same lines, we exclude the VMs that give a negative worth percentage increase (by assigning them $+\infty$ in Equation (64) so that they will never be selected as minimum) because we do not want the migrated service to run in an environment that does not satisfy its actual performance needs. On the other hand, if no VMs satisfying the technical and security constraints of the migration source v_i are available (i.e., $E^x(v_i) = \emptyset$) or no VMs having non-negative worth increase percentage compared to v_i exist (i.e., $p^x(v_i \longrightarrow v_j^*) = +\infty$), then the hypervisor creates a new VM and migrates the services running inside the risky VM to it. In the following, we give an illustrative example that shows how the migration decision would be practically taken. Assume that we have two VMs v_1 and v_2 that are classified as highly risky and whose worth is of 6 for v_1 and 2 for v_2. The set of possible replacements for v_1 and v_2 that are technically compatible and classified as low risk are v_3, v_4, and v_5. The worths of v_3, v_4, and v_5 are 6, 3, and 1 respectively. The worth increase percentage from v_1 to v_3 is of $p^x(v_1 \longrightarrow v_3) = \frac{6-6}{6} = 0\%$, from v_1 to v_4 is of $p^x(v_1 \longrightarrow v_4) = +\infty$ (since $\frac{3-6}{6} = -0.5\% < 0$), and from v_1 to v_5 is of $p^x(v_1 \longrightarrow v_5) = +\infty$ (since $\frac{1-6}{6} = -0.83\% < 0$). Thus, v_3 that gives the minimum worth increase percentage w.r.t v_1 will be selected as the best candidate to replace v_1. On the other hand, the worth increase percentage from v_2 to v_3 is of $p^x(v_2 \longrightarrow v_3) = \frac{6-2}{2} = 2\%$, from v_2 to v_4 is of $p^x(v_2 \longrightarrow v_4) = \frac{3-2}{2} = 0.5\%$, and from v_2 to v_5 is of $p^x(v_2 \longrightarrow v_5) = +\infty$ (since $\frac{1-2}{2} = -0.5\% < 0$). Again, v_4 that gives the minimum worth increase percentage w.r.t v_2 is selected as the best candidate to host v_2's migrated services.

After applying this migration strategy, we will have two types of VMs present in the cloud system: *working* VMs and *idle* VMs. Working VMs are those containing real services running inside them, whereas idle VMs are those VMs either running no services or running

fake services not pertaining to any user. Note finally that we adopt the *snapshot-and-restore* migration approach [30] in which a snapshot of the service to be migrated is taken and kept in the VM. Consequently, if the service is migrated back to the same VM, the snapshot is restored and the service resumes from that point. Thus, unlike the *refreshing* migration approach [30] which destroys the execution context of the migrated service, the *snapshot-and-restore* model maintains the service's execution context in the VMs, which minimizes the time and overhead of migrating back the services across VMs.

7.3.3 Honeypot Deployment and Attackers' Types Recognition

In order to learn the probability distributions over the attackers' types and inspect their objectives, the hypervisor can exploit the *idle* VMs by deploying honeypots inside them to serve as traps for attackers. A honeypot in our case is a deception VM that is configured by the hypervisor to serve as a purposed target for attacks. The objective is to give attackers the impression that they are interacting with a real system and encourage them hence to freely launch their attacks in order to gather massive and valuable information. In this way, any connection with the honeypot would be deemed to be an attack and all the traffic circulating to the honeypot is roughly entirely unauthorized. Honeypot systems can be either of low-interaction or high-interaction [116]. Low-interaction honeypots (e.g., Honeyd) function by emulating services designed to catch some specific malicious activities (e.g., FTP login), which makes them limited to a confined level of interaction with attackers. The main advantages of low-interaction honeypots lie in their simplicity to deploy and maintain, and in the minimal risk that they entail to the system. Practically, low-interaction honeypots do not allow attackers to have access to the OS, which protects the cloud system and co-hosted VMs from potential attacks. Nonetheless, the main disadvantages of low-interaction honeypots are the limited amount of information that they can capture and their simple configuration that increases the capability of skillful attackers to detect their presence.

On the other hand, high-interaction honeypots (e.g., Honeynets) consist of real applications and OSs that are designed for advanced research purposes. Simply speaking, high-interaction honeypots involve providing a real execution environment (e.g., a real Windows honeypot system running a real FTP server) in which nothing is being emulated. The main advantage of such honeypots is the ability to gather large amounts of information that enable analyzing and understanding the complete extent of the attackers' malicious behavior. Moreover, the fact that high-interaction honeypots rely on real systems makes them appealing to attackers and hard to be recognized as being traps. However, the main self-evident disadvantage of such a type of honeypots lies in the risks that they might impose on the real system. Therefore, a thoughtful implementation and configuration of high-interaction honeypots is required to block attackers from exploiting these honeypots

to hurt other non-honeypot systems. Such a thoughtful implementation might include, for example, isolating the CPU assigned to honeypots from that assigned to non-honeypot VMs to prevent scheduling tasks coming from honeypots on the same physical CPU as other non-honeypot VMs.

Because the aim of our honeypots deployment process is to study the behavior of the attackers to be able to determine the probability distributions over their types, we choose to employ high-interaction honeypots for our problem [14]. The fact that high-level interaction honeypots make no predefined assumptions on how attackers shall misbehave makes them able to capture all types of malicious activities including unexpected misbehavior. Thus, they are suitable to study and analyze different types of attacks including unknown ones. Furthermore, in order to make honeypots even more appealing for attackers, our honeypot deployment approach consists of keeping a copy of the (migrated) services running inside honeypot VMs, while using fake data to populate them. For example, a banking system that migrates to another safer VM will keep running inside the honeypot VM, while using dummy accounts numbers, clients' names, etc. Along the same line, the services running inside honeypot VMs are changed and updated on a regular basis to minimize the chances of being discovered by attackers as traps.

The use of honeypots helps us achieve many benefits regarding the data set's size, false negatives and positives, resources consumption, and detection effectiveness. Specifically, since the data collected by honeypots is restricted to attack and unauthorized events, the size of the data to be stored and analyzed tends to be greatly reduced compared to the data generated by the traditional IDSs. This is because the use of honeypots removes the load of the normal activity data whose size is usually much larger than the one of attack events. This will allow the attackers' types classification system to function with a minimal amount of resources compared to other intrusion detection techniques. Moreover, the employment of honeypots aids in diminishing the false negatives and positives of the classifier. This is realised by characterizing new attacks for which traditional IDSs do not have existing signatures. Besides, honeypots are effective in catching advanced attacks including encrypted ones because they run in kernel mode and are able hence to see all data prior to encryption.

Having collected the necessary data from honeypots, we need a classification technique to analyze this data and learn the probability distributions over the attackers' types. To this end, we choose to employ the one-class Support Vector Machine (SVM) [158] which has been proposed as an extension of the traditional SVM binary classifier. One-class SVMs try to find the decision boundary (i.e., hyperplane) which separates the majority of the data points from the origin. In this way, the data points that lie on the other side of the decision boundary will be deemed to be *outliers* or *abnormal activity*. This enables the

decision function to classify any new data as being analogous or different from a certain pattern of data fed in the training phase (i.e., novelty detection). The selection of one-class SVM to be used in our problem stems from three main observations. First, one-class SVM is an unsupervised classification technique which requires no extensive prior information nor predefined class labels for the analyzed data. Second, one-class SVM supports multi-class data classification, which makes it appropriate for our problem in which we deal with attackers of multiple types. Third, the fact that one-class SVM is dedicated to novelty detection makes it well-suited to identify new types of (yet) undetected attacks. This might be achieved by considering each type of already identified attacks as a *normal activity* and determining the degree of similarity/dissimilarity of each set of new data w.r.t that normal activity data. Suppose, for example, that the classification system has already identified DoS and privilege escalation attacks. If new features that do not match neither DoS nor privileges escalation attacks' features are found on the honeypot system, then this would be considered as a new attack type targeting the cloud system.

Formally, let $x = (x_1, x_2, \ldots, x_n)$ denote the feature vector which contains all the attack features (e.g., source and destination IP addresses, host names, protocol used, geographical information of the attack sources, etc.) collected by the honeypot system. The one-class SVM classification problem is mapped into solving the following objective function's minimization problem:

$$\begin{cases} \min\limits_{\omega,\xi_i,\rho} \ \frac{1}{2}\|w\|^2 + \frac{1}{\upsilon n}\sum_{i=1}^{n}\xi_n - \rho \\ \text{subject to:} \\ (w.\phi(x_i)) \geq \rho - \xi_i & \forall i = 1,\ldots,n \\ \xi_i \geq 0 & \forall i = 1,\ldots,n \end{cases} \tag{65}$$

where n is the size of the training set, w represents the normal vector to the hyperplane, and ρ is the bias term. Moreover, $\phi(.)$ denotes a transformation function concretized by the kernel function to project the data into a higher dimensional space and $\xi_i \in \xi_n$ are slack variables used to allow some data points to lie within the margin so as to prevent the SVM classifier from over-fitting with noisy data. Yet more importantly, υ is the regularisation parameter that determines the shape of the solution by specifying (1) an upper bound on the fraction of outliers; and (2) a lower bound on the number of training tuples employed as support vectors. Thus, an increased value of υ widens the soft margin and augments the probability that the training data will fall outside the normal borders. Problem (65) can be solved using the Lagrange multipliers method so that the decision function $f(x)$ becomes:

$$f(x) = sgn((w.\phi(x_i)) - \rho) = sgn(\sum_{i=1}^{N}\alpha_i k(x, x_i) - \rho) \tag{66}$$

where $k(x, x_i)$ is the kernel function, which might be either linear, polynomial, gaussian, or sigmoid, i.e.,

$$K(x_j, x_i) = \begin{cases} x_i.x_j, linear \\ (\gamma.x_i.x_j + c)^d, polynomial \\ \exp(-\gamma.|\ x_i - x_j\ |^2), gaussian\ radial\ basis \\ \tanh(\gamma.x_i.x_j + c), sigmoid \end{cases} \tag{67}$$

Based on the results obtained from the classification process, the hypervisor computes the probability p^y for each attacker's type $y \in Y$ using Equation (68).

$$p^y = \frac{\text{Number of observations classified as "}y\text{"}}{\text{Total number of observations}} \tag{68}$$

Finally, this information is used back to feed the Bayesian Stackelberg game (Section 7.2) with the attackers' types probability distributions to help it continuously adjust and optimize the detection load probability distributions over the set of guest VMs.

7.4 Experimental Results and Analysis

We explain in this section the environment employed to perform our experiments and present and analyze the experimental results.

7.4.1 Experimental Setup

To carry out the experiments, we build our own cloud datacenter using CloudSim [28], a cloud simulator that provides realistic cloud features such as co-hosted VMs, network connections among cloud components, resource allocation policies, and services migration and cloud federation support. The decision to create our own cloud rather than using rented resources from existing providers stems from two observations [146]. In the first place, most of the cloud providers (e.g., Amazon EC2) have strict restrictions concerning any security testing on their resources and infrastructure [124]. In the second place, cloud providers forbid any direct access of the users to the VMs' host system, thus making the acquisition of performance data and the implementation of new algorithms at the host's level far difficult to achieve. The characteristics of the created cloud are populated from the Amazon EC2 X-large instances [1] in terms of VMs configurations and pricing scheme. Specifically, the cloud datacenter is equipped with 100 physical machines, each hosting a number of VMs varying from 10 to 50. The image size of the VMs is of 10000 MB, the memory RAM capacity is of 16 GB, and the hard drive storage is of 976.5625 GB. Each VM

is supplied with a 5-core CPU of 1000 Millions of Instructions Per Second (MIPS) each. The network bandwidth share of each VM is 50000 Kbit/s. Moreover, Linux has been adopted as an OS in the datacenter, x86 as a system architecture, and Xen as a Virtual Machine Monitor (VMM). The prices of the VMs, used to compute the utility functions, have been selected according to Amazon EC2 pricing scheme [2].

To analyze the performance of the attackers' types recognition phase, we use a dataset [3] from the Data Driven Security (DDS) datasets collection. The dataset is collected from Amazon Web Services (AWS) honeypots deployed on several instances across the world for a period covering March to September 2013 [52]. The dataset records attack data including source and destination IP addresses, host names, protocol used (e.g., TCP, UDP, etc.), source and destination ports, and geographical information of the attack sources (i.e., country, postal code, longitude, and latitude). To create the training and test sets, we use the k-fold cross-validation technique (with $k = 10$) whereby the dataset is split into k subsets, each used every time as test set and the remaining $k-1$ subsets are combined together to form the training set. The principal advantage of the k-fold cross-validation lies in its ability to diminish the bias of the classification results on the way based on which data is being divided since each data tuple will be part of the test set exactly once and part of the training set $k-1$ times. As for the defense mechanism, in addition to implementing it in CloudSim as part of the repeated Stackelberg game, we were able to study the performance of our live-migration-based defense strategy independently in a real environment using OpenStack [6], an open-source software platform for cloud computing that is based on interrelated components controlling diverse, multi-vendor hardware pools of processing, storage, and networking resources. We used OpenStack to create VMs and assign real jobs to them including on-line gaming server and Hadoop MapReduce tasks. The purpose is to assess the time needed to perform the live migration of services having different sizes and complexities. Finally, to populate the probability distributions over the attackers' types (used to achieve the Bayesian property of the Stackelberg game), we capitalize on the findings presented in [112], which surveys the attacks/vulnerabilities distributions on Xen hypervisors (used in our simulations) based on real data collected from NVD [5], SecurityFocus [8], Red Hat's Bugzilla [7] and CVEs [4]. These probability distributions are summarized in Table 7.7. Note that all the experiments have been conducted in a 64-bit Windows 7 environment on a machine equipped with an Intel Core i7-4790 CPU 3.60 GHz Processor and 16 GB RAM.

To show the improvements brought by our solution compared to the state-of-the-art, we compare our work experimentally with three other load distribution strategies, namely the one-stage Stackelberg [145], maxmin [146, 143], and fair allocation [154]. In the fair allocation model, the detection load is distributed in an equal manner among VMs so as to guarantee the fairness of the detection process. On the other hand, the maxmin-based

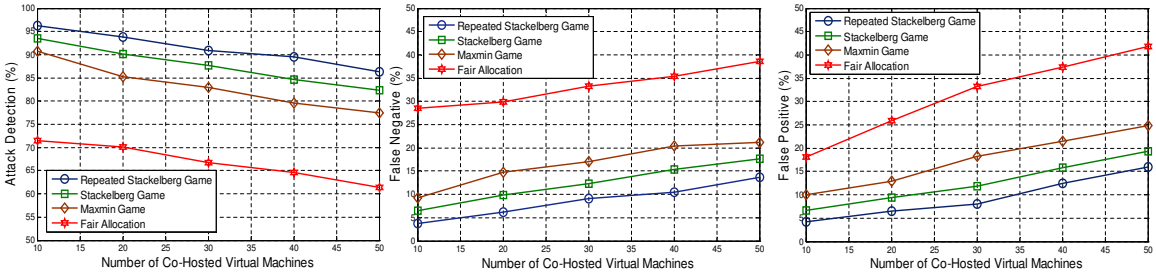
Table 7.7: Attacks occurrence distributions on Xen hypervisors

Attack	Percentage of Occurrence on Xen Hypervisors
Hypervisor's Memory Content Disclosure	8.5%
Co-hosted VMs' Memory Modification	6.8%
Denial of Service	44.1%
Virtual Machine Destruction	11.9%
Virtual Machine Crash	3.4%
Privileges Escalation	25.3%

detection load distribution strategy leverages a maxmin game whose utility functions are mainly fed by the trust scores computed by the hypervisor toward its guest VMs. Although the maxmin-based strategy accounts for the attackers' strategies and resources constraints in the design of the problem (as is the case in our solution), it does not account, however, for the fact that attackers have the ability to monitor the cloud system's strategies and adjust their own strategies. Similar to our solution, the one-stage Stackelberg accounts for this challenge by computing the best responses of the attackers to the hypervisor's detection load distribution strategies and incorporating this knowledge into the hypervisor's optimization problem. Different from our solution, the one-stage Stackelberg model abstracts on the types of attackers and is hence not able to provide the hypervisor with real-time learning about the actual types and objectives of the attackers. Our work overcomes this limitation by collecting and analyzing malicious data to learn the probability distributions over the types of attackers targeting the cloud system and incorporating this knowledge into the hypervisor's optimization problem to optimize its decisions. Moreover, our solution offers a proactive defense mechanism (Section 7.3.2) that protects services from being successful targets for attackers and works in a repeated fashion to provide incremental and continuous learning for the cloud system.

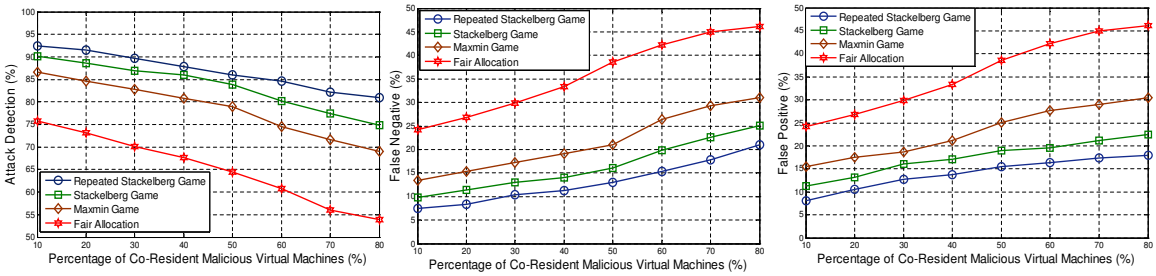
7.4.2 Experimental Results

Fig. 7.2 illustrates the detection performance metrics (attack detection, false negative, and false positive percentages) of the four studied solutions. Attack detection represents the percentage of attacks that the IDS was able to identify as such and is calculated as per Equation (52). On the other hand, false negative depicts the percentage of attacks identified by the IDS to be non-attacks. This percentage is computed by subtracting the detection load probability distributions of the cloud system from the attack probability distributions of the attacker when the values of the former are smaller. Equation (69) shows how to compute the false negative rate $\alpha([t_1,t_2])$ during the time window $[t_1,t_2]$:



(a) Attack Detection *(b) False Negative* *(c) False Positive*

Figure 7.2: Detection performance: Our solution improves the detection performance and is scalable to the increase in the number of co-hosted VMs compared to the one-stage Stackelberg, maxmin, and fair allocation strategies



(a) Attack Detection *(b) False Negative* *(c) False Positive*

Figure 7.3: Detection performance: Our solution improves the detection performance and is scalable to the increase in the percentage of co-resident malicious VMs compared to the one-stage Stackelberg, maxmin, and fair allocation strategies

$$\alpha([t_1,t_2]) = \sum_{x=t_1}^{t_2} \sum_{v \in V} \frac{\mu}{t_2 - t_1}, \text{where} \tag{69}$$

$$\mu = \begin{cases} a_x(v) - h_x(v), & \text{if} \quad a_x(v) > h_x(v) \\ 0, & otherwise \end{cases}$$

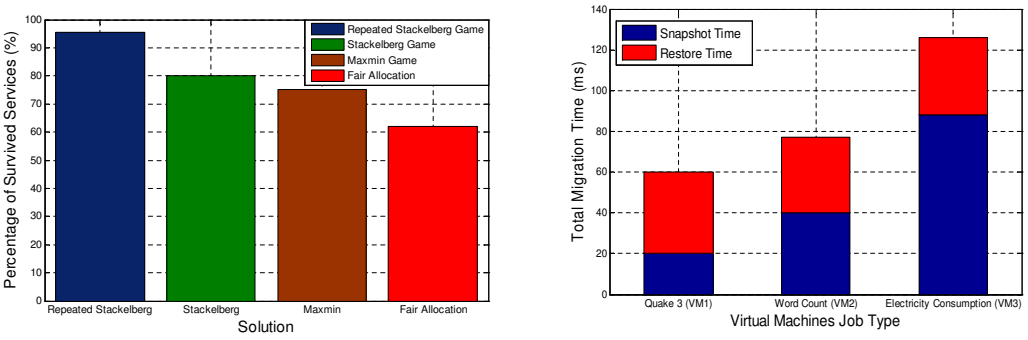
False positive describes the percentage of non-attack activities identified by the IDS as attacks. This metric is of special importance since it gives a hint on the amount of security resources wasted during the detection process. Simply speaking, false positive tells us the percentage of resources spent by the hypervisor in monitoring some VMs, while these VMs were not selected by attackers to launch attacks. As explained in Equation (70), the false positive rate $\gamma([t_1,t_2])$ during the time window $[t_1,t_2]$ is computed by subtracting

the detection load probability distributions of the cloud system from the attack probability distributions of the attacker when the values of the former are larger.

$$\gamma([t_1,t_2]) = \sum_{x=t_1}^{t_2} \sum_{v \in V} \frac{\chi}{t_2 - t_1}, \text{where} \tag{70}$$

$$\chi = \begin{cases} h_x(v) - a_x(v), & \text{if} \quad h_x(v) > a_x(v) \\ 0, & otherwise \end{cases}$$

By examining Fig. 7.2, we can notice that the performance of all the studied solutions begins to decrease with the increase in the number of co-hosted VMs. The reason is that increasing the number of VMs on a single physical machine increases the attack space for attackers by giving them an increased number of VMs to distribute their attacks over. Moreover, the increase in the number of co-hosted VMs would lead to reduce the effectiveness of the security budget, since the same budget would need to be distributed across a larger number of VMs. Thus, the share of security resources for each single VM is naturally reduced as the number of VMs grows up. However, we can notice from Fig. 7.2 that our solution remains far more resilient to an increased number of co-hosted VMs than the other solutions. The second observation that can be made from Fig. 7.2 is that our repeated Bayesian Stackelberg, the maxmin, and one-stage Stackelberg models achieve better detection performance (in terms of attack detection, false negative, and false positive) compared to the fair allocation strategy. The reason is that the repeated Bayesian Stackelberg, one-stage Stackelberg, and maxmin models consider the attackers' strategies in the formulation of the game, which enables them to compute the optimal detection load distributions that best synchronize with the attackers' strategies. On the other hand, the fair allocation model seeks to achieve the fairness in the detection process by distributing the detection load in an equal manner among VMs, thus overlooking how attackers' are distributing their attacks. For example, a fair allocation model which distributes the detection load amongst three VMs v_1, v_2, and v_3 so that each one receives 33.33% might end up assigning a big part of the security resources (i.e., 33%) monitoring a VM that will not be selected by attackers to contribute in the attacks. Moreover, the Stackelberg-based solutions (i.e., our solution and the one-stage Stackelberg) outperform the maxmin-based solution since the former models account for the fact that attackers have the ability to monitor the hypervisor's detection load distribution strategies and they integrate this knowledge into the hypervisor's optimization problem to optimize its detection strategies. Finally, our repeated Bayesian Stackelberg solution performs better than the one-stage Stackelberg



(a) Percentage of Survived Services *(b) Total Live Migration Time*

Figure 7.4: Defense mechanism: Our defense mechanism maximizes the percentage of survived services and takes less than one second to run

because it includes a learning component that learns the types and objectives of the attackers and incorporates this knowledge into the hypervisor's optimization problem. This increases the awareness of the hypervisor about the nature and gravity of the attacks that are expected to be launched on every VM and aids it hence to adjust the detection load distributions accordingly.

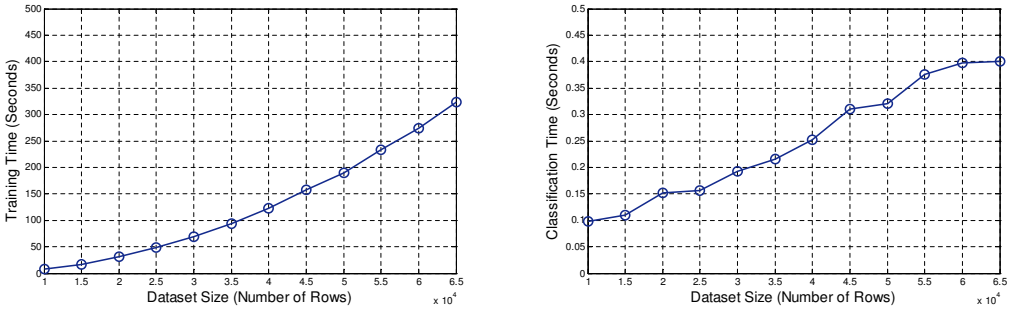
In Fig. 7.3, we study the scalability of our solution with respect to the variation in the percentage of co-resident malicious VMs. To do so, we vary the percentage of attacking VMs co-residing on a single cloud system from 10% up to 80% to explore the effects of this variation on the performance of the studied solutions. As shown in Fig. 7.3, the performance of all the solutions begins to decrease with the increase in the percentage of attacking VMs. This unsurprising result is due to the fact that the bigger the number of VMs attacking the system, the less is the ability of the cloud system to capture attacks under the limited budget of security resources. Thus, a possible choice for the cloud system's administrators would be to increase the security budget to face an increased number of attacks. Specifically, our attackers' types recognition phase can aid the cloud's administrators in deciding whether there is a need to increase the security resources budget or not by giving them detailed information about the volume and nature of attacks targeting the cloud system. Fortunately, our solution shows a better scalability to an increased percentage of attacking VMs compared to the other models even in extreme cases (i.e., 80% of co-resident malicious VMs) thanks to the previously discussed advantages brought by our solution.

In Fig. 7.4, we study the effectiveness and efficiency of our MTD-based defense mechanism by measuring the percentage of survived services and the total live migration time. The percentage of survived services represents the percentage of services that remained unattacked during their whole lifetime. We can notice from Fig. 7.4a that our solution is able to increase the number of survived services compared to the other solutions. This is

thanks to the proactive defense mechanism that our solution advances and that migrates the services running inside risky VMs to other more secure VMs to protect them from being successful targets for attacks. The absence of such a mechanism in the other solutions limits their effectiveness to some reactive measures (i.e., detection) and leads hence to an increased number of attacked services.

Moving to the efficiency evaluation, Fig. 7.4b illustrates the average time taken to migrate services of different complexities. To run this experiment, we use OpenStack to create three VM instances connected using an internal network and accessible through the internet (using a public IP address). Each VM instance is assigned a different job type with the aim of studying the live migration performance on VMs having varying loads. For each VM, we execute the migration process ten times and compute the average time of all experiments. Note that the *snapshot-and-restore* migration approach described in Section 7.3.2 has been applied during the experiments. The first VM (i.e., VM1) is given Quake 3, a multiplayer on-line game server, which is considered as a low-overhead application. The migration occurred at the point where four players joined the game and began to play. We can observe from Fig. 7.4b that the total migration time of such an application amounts to $60ms$ (Fig. 7.4b), which is negligible. The second VM (i.e., VM2) is given a Hadoop MapReduce task (written in Java using Apache Maven) of average complexity. The task consists of counting the words occurrences in a medium-sized text file. The total migration time in this case recorded $77ms$ (Fig. 7.4b). The third VM (i.e., VM3) is given a complex Hadoop MapReduce task which consists of computing the average electricity consumption for all the cities in the United States per a given year in a large-sized text file. The migration in both the second and third VMs happened at the point where the mapping task was executing, i.e., at the point where mappers were distributively grouping data prior to passing it to the reducers. The total migration time for this third job recorded $126ms$ (Fig. 7.4b).

Overall, we can conclude that the live migration time is almost negligible (i.e., < 1 second) in all the studied scenarios and does not entail any significant impact on the cloud system's performance. If we observe Fig. 7.4b more carefully, we can notice that the migration time is influenced by two main factors, namely the snapshot time and the restore time. As names hint, the snapshot time represents the time needed to take a snapshot of the service being migrated, which is affected practically by the amount of data being transferred and the network bandwidth being used to transfer this data to the destination. Given that the bandwidth rate is nearly the same for all the VMs in our experiments, the amount of data is the main factor that affects the total migration time of the VMs, where for example, the migration in a VM running a heavy Hadoop MapReduce task (i.e., electricity consumption) would take longer than the migration in a VM running a simple word count task on a smaller dataset. On the other hand, the restore time consists of the time needed



(a) Training Time *(b) Classification Time*

Figure 7.5: Training and classification times: The training and classification times entailed by our attackers' types recognition technique are acceptable

to start back the migrated data on a new VM from the point at which migration took place. We can notice from Fig. 7.4b that the restore time is, ipso facto, less affected by the amount of data than the snapshot time.

We study in Fig. 7.5 the efficiency of our attackers' types recognition phase in terms of execution time. Fig. 7.5a shows the time required to train the one-class SVM on various training datasets sizes. To do so, we employ the DDS honeypot data collected from AWSs and whose original size amounts to $65,0000$ rows. To study the impact of the training dataset's size on the training time, we vary the size of the data from $10,000$ to $65,0000$. Unsurprisingly, Fig. 7.5a reveals that the training time increases with the increase in the size of the training dataset and reaches at the extreme case (i.e., $65,0000$ rows) $330s$. The main time complexity lies in the process of constructing an SVM model for each class label. Practically, since we have 6 types of attackers (Table 7.1) serving as class labels for the training dataset, we have to build one SVM model for each single class and train it to differentiate the samples of that class from the samples of all remaining classes (i.e., novelty detection). We argue that the obtained time is insignificant, especially since this phase is executed offline and not required to be repeated at each time moment as discussed in Section 7.3. Having completed the training process, the next step is to execute the actual classification part, which consists of assigning an attack type for each particular sample. Since the classification time is also dependent on the dataset's size as is the case for the training time, we test the classification's time on different dataset sizes. It can be noticed from Fig. 7.5b that the classification time is negligible in all the considered cases, where it does take $0.4s$ to classify samples in a dataset consisting of $65,0000$ rows.

Finally, we study in Fig. 7.6 the execution times of the four considered solutions. At this stage, we exclude the live migration and one-class SVM execution times from the repeated Bayesian Stackelberg model for the two following reasons. First, the execution times of these two phases vary based on external factors, i.e., the job type in case of live migration (Fig. 7.4b) and the dataset's size in case of one-class SVM (Fig 7.5a). Thus, it would be

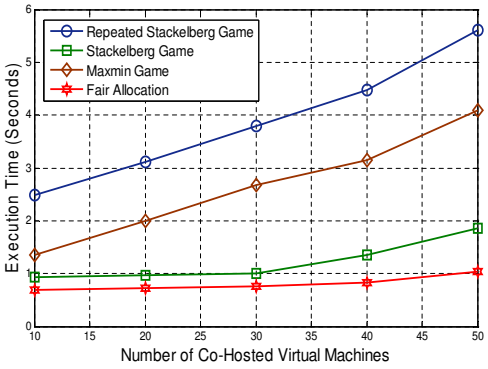


Figure 7.6: Execution time: Our solution is efficient in terms of execution time and grows polynomially with the increase in the number of co-hosted VMs

unfair to rely on one particular choice when comparing with the other approaches. Second, these two phases are not required to be repeatedly executed as explained in Section 7.3. Therefore, we tested their performance independently since including them in the comparisons with the other approaches would not be informative. By examining Fig. 7.6, we can notice that the fair allocation approach yields the fastest performance. This is because the detection load is to be distributed equally across VMs, which removes the time complexity of finding the optimal detection load probability distributions. Moreover, the one-Stackelberg performs faster than the maxmin-based and our repeated Bayesian Stackelberg game. The time difference between the one-stage Stackelberg and maxmin models may be thought of as the time needed by the latter to gather objective and subjective sources of trust and compute the final trust values prior to executing the maxmin game. On the other hand, the time difference between our repeated Stackelberg model and the one-stage Stackelberg and maxmin-based models lies in the time taken by our solution to perform the VMs' risk assessment as well as the computational time entailed by the integration of the attackers' type into the optimization problem. Though, our solution still performs in an efficient manner, where it takes $\approx 5.6s$ to run in a cloud system consisting of 50 co-hosted VMs. We can also notice that the time complexity of our solution grows polynomially with the increase in the number of VMs, which boosts its scalability in large-scale datacenters.

7.5 Conclusion

In this chapter, we introduced a comprehensive detection and defense mechanism for cloud-based systems that consists of the following phases: (1) risk assessment framework that evaluates the risk level of each guest VM; (2) MTD-based defense mechanism that

intelligently migrates services running inside risky VMs to other more secure VMs; (3) machine learning technique that recognizes the types of attackers using honeypot data; and (4) resource-aware Bayesian Stackelberg game that aids the hypervisor in determining the optimal detection load distribution strategy among VMs. To the best of our knowledge, this work is the first that advances such a host-level comprehensive detection and defense strategy against multiple types of attacks in the domain of cloud computing. Experiments conducted using OpenStack, Amazon's public datacenter, and AWSs honeypot data reveal that our solution is able to improve the detection performance up to $\approx 7\%$ and minimize the percentage of attacked services by $\approx 15\%$ compared to the state-of-the-art intelligent detection strategies, namely the maxmin, one-stage Stackelberg, and fair allocation. As for the efficiency, the experimental results show that our live-migration-based defense mechanism needs less than one second to execute irrespective of the complexity of the jobs running inside VMs and that our machine learning technique needs $\approx 330s$ to train in a dataset comprising $65,000$ rows and consisting of six types of attackers. Finally, our detection load distribution strategy takes $\approx 5.6s$ to run in a cloud system of 50 co-hosted VMs and grows polynomially with the increase in the number of co-hosted VMs. Thus, we have achieved our third research objective (**Objective** 3) discussed in Chapter 1, which aimed at maximizing the detection of multi-type intelligent active malicious attacks using a limited budget of resources.

Chapter 8

Conclusion and Future Directions

In this thesis, we proposed a new architecture for boosting the performance and security of cloud-based applications called *multi-cloud services communities* and addressed the design, business, and security challenges of adopting and implementing such an architecture. For each of these aspects, we conducted an in-depth literature review to guarantee the originality of our solutions and their effectiveness in filling the state-of-the-art research gaps. In particular, we developed, for the first time in the domain of cloud computing, a community formation model that accounts for the business potential of the involved services and gives privileges to the market's leader services by allowing them to select the set of (follower) services that they are willing to form community with and deciding about the appropriate offer to make. Experiments conducted on a real services dataset show that this solution maximizes the satisfaction of both leader and follower market's services up to 20% compared to the state-of-the-art community formation models, which motivates hence the participation of services of varying business capabilities in the community formation process. Secondly, we elaborated the first cooperation model in the domain of cloud computing that is able to form communities among services deployed in multiple cloud centers, while accounting for the problem of encountering passive malicious services that seek to illegally maximize their profits. Promisingly, experiments conducted using the CloudHarmony dataset reveal that this solution minimizes the percentage of passive malicious services by 30% compared to the state-of-the-art cloud federation and services community formation models. Third, we advanced the first detection load distribution strategy aids the cloud system in determining the optimal way of distributing the detection load among the guest VMs so as to maximize the detection of DDoS attacks, while respecting a limited budget of security resources. Experiments conducted using Amazon EC2's public cloud reveal that this solution can maximize the detection of DDoS attacks up to 26% compared to the state-of-the-art cloud-dedicated resources distribution techniques.

Finally, we put forward the first host-level cloud-dedicated comprehensive detection and defense mechanism that maximizes the detection of multi-type intelligent simultaneous attacks and protects cloud services from being successful targets for attackers. Experiments conducted using OpenStack and Amazon EC2's public cloud demonstrate that this solution ameliorates the detection performance up to 7% and minimizes the percentage of attacked services by 15% compared to the state-of-the-art intelligent detection strategies.

The following points summarize the main contributions of this thesis:

1. We proposed a business-oriented community formation model that allows services of uneven business capabilities to distributively group themselves into joint communities so as to optimize their reputation, market share, and capacity of handling simultaneous requests. The proposed model fits the real-world services market and motivates the leading cloud services (e.g., Amazon and Google) to get structured into communities.

2. We put forward a comprehensive inter-cloud trust framework that allows services deployed in different cloud data centers to build credible trust relationships toward each other. The proposed trust framework is able to generate credible trust results that are resilient to the collusion attacks even when attackers form the majority.

3. We designed a polynomial-time services discovery algorithm that enables services to inquire about each other's behavior based on the concept of *tagging* in online social networks.

4. We proposed a trust bootstrapping mechanism that combines the concept of *endorsement* in online social networks with the decision tree classification technique to assign initial trust values for the newly deployed services for which no evidence about their former behavior can be found.

5. We introduced a security-oriented community formation model among cloud services deployed in one or multiple cloud centers. The solution is modeled as a trust-based hedonic coalition formation game, where a relevant algorithm that converges to a final stable coalition partition of services is advanced.

6. We developed an intra-cloud trust framework that allows the hypervisor (acting on behalf of the cloud system) to build credible trust relationships toward its guest VMs. The proposed framework uses objective (i.e., monitoring) and subjective (i.e., recommendations) sources of trust to optimize the credibility of the trust values.

7. We designed and solved a trust-based maxmin game between the hypervisor and DDoS attackers. The solution of the game provides the hypervisor with the optimal

detection load distribution strategy over VMs that maximizes the detection of DDoS attacks under a limited budget of security resources.

8. We proposed a risk assessment framework that assists the cloud system with evaluating the risk level of each guest VM and identifying the risky ones that are likely to be targets for attacks. For this purpose, we formulated a risk level determination model that capitalizes on the VMs' potential vulnerabilities, expected threats, and past attack history to make thoughtful decisions.

9. We developed an MTD-based defense mechanism that protects cloud services from being successful targets for attackers. This is done by putting forward an intelligent security-oriented live migration strategy that allows the hypervisor to migrate the services running inside risky VMs to other more secure ones.

10. We elaborated a machine learning technique that provides the hypervisor with a detailed view of the types and objectives of the attackers targeting the cloud system. This is achieved by developing a honeypots' deployment strategy inside risky VMs to collect malicious data and discussing a one-class SVM learning classifier which analyzes this data and predicts the actual types of the attackers.

11. We designed and solved a Bayesian Stackelberg game that guides the cloud system to determine the optimal detection load distribution strategy among VMs that maximizes the detection of multi-type intelligent attackers that are able to continuously observe the detection strategies of the cloud system and update their own attack strategies accordingly.

The first contribution is proposed to answer our first research objective (**Objective** 1), which sought to enable the formation of communities consisting of services belonging to clouds having uneven business capabilities. The second, third, fourth, and fifth contributions are proposed to answer our second research objective (**Objective** 2), which was about enabling the formation of trustworthy communities of multi-cloud services wherein the number of passive malicious services is minimal. The sixth and seventh contributions are proposed to partially answer our third research objective (**Objective** 3), which targeted the maximization of the detection of multi-type intelligent active malicious attacks using a limited budget of resources. Finally, the eighth, ninth, and tenth contributions are proposed to fully answer this third research objective (**Objective** 3).

The above-discussed thesis contributions are effective in solving some interesting research gaps in the literature. However, some points still need further study and investigation. We summarize in the following list the main persisting gaps that we believe, based on our literature reviews, are worth investigating in the future:

- There is a significant gap between the existing federation/community formation models and the real needs of the industrial society, where most of the proposed models, despite their theoretical importance, fail to provide effective cooperation strategies for the cloud providers. In this thesis, we made a first attempt toward bridging this gap by formulating a community formation model that is inspired by the real industrial reality and that considers the practical needs of the providers in the formation process. Yet, further investigations and extensions are needed to come up with additional practical solutions that fit the market's needs.

- The current community and federation formation models disregard the resources allocation problem in the formation process. This raises the need to develop some resource-aware formation solutions that guide each cloud provider to decide about how much resources should be dedicated to the federation/community and how much resources should be kept to serve the provider's own clientele.

- The existing trust models in the domain of cloud computing are focusing only on the trust relationships between customers and cloud providers/services but disregard the trust relationships among the cloud's internal components. In this thesis, we made a first step toward investigating the intra-cloud trust relationships by exploring the trust connections between the hypervisor and its guest VMs. However, further efforts are needed to investigate the relationships among the other components of the cloud centers (e.g., servers, databases, etc.).

- There is a need to develop intelligent detection techniques that go beyond the monitoring of the system and the identification of some traditional attacks to capture and recognize advanced attack patterns such as networks of attacks, key members in attackers' networks, and selective attacks. For this purpose, machine and deep learning techniques could be the best candidate to replace or complement the existing detection approaches.

- There is a scarcity of cloud-specific security datasets available online, which makes it difficult for security researchers to test their work using real parameters. Therefore, it would be of prime importance to develop real attack experiments under cloud environments and gather some meaningful data that can serve as a reference for the future cloud-dedicated security research studies.

Bibliography

[1] Amazon EC2 instances. https://aws.amazon.com/ec2/details/. Accessed: 2017-05-16.

[2] Amazon EC2 pricing scheme. http://aws.amazon.com/ec2/pricing/. Accessed: 2017-05-16.

[3] Amazon Web Services honeypot data. http://datadrivensecurity.info/blog/pages/dds-dataset-collection.html. Accessed: 2017-05-16.

[4] CVE Security Vulnerability Database. http://www.cvedetails.com/. Accessed: 2017-05-16.

[5] National Vulnerability Database. http://web.nvd.nist.gov/view/vuln/search. Accessed: 2017-05-16.

[6] OpenStack. https://www.openstack.org/. Accessed: 2017-05-16.

[7] Red Hat Bugzilla. https://bugzilla.redhat.com/. Accessed: 2017-05-16.

[8] Securityfocus. http://www.securityfocus.com/. Accessed: 2017-05-16.

[9] SPEC Java Virtual Machine Benchmark 2008. http://www.spec.org/jvm2008/. Accessed: 2016-08-30.

[10] J. H. Abawajy and A. M. Goscinski. *Computational Science - ICCS 2006*, volume 3994 of *Lecture Notes in Computer Science*, chapter A Reputation-Based Grid Information Service, pages 1015–1022. Springer Berlin Heidelberg, May 2006.

[11] Stephane Airiau. Cooperative Games and Multiagent Systems. *The Knowledge Engineering Review*, 28(04):381–424, 2013.

[12] Eyhab Al-Masri and Qusay H Mahmoud. Qos-based discovery and ranking of web services. In *Proceedings of 16th International Conference on Computer Communications and Networks (ICCCN)*, pages 529–534. IEEE, 2007.

[13] Eyhab Al-Masri and Qusay H. Mahmoud. Discovering the best web service: A neural network-based solution. In *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics*, pages 4250–4255, San Antonio, TX, USA, 2009. IEEE.

[14] Eric Alata, Vincent Nicomette, Mohamed Kaâniche, Marc Dacier, and Matthieu Herrb. Lessons learned from the deployment of a high-interaction honeypot. In *Dependable Computing Conference, 2006. EDCC'06. Sixth European*, pages 39–46. IEEE, 2006.

[15] Turki Alharkan and Patrick Martin. IDSaaS: Intrusion detection system as a service in public clouds. In *CCGrid 2012*, pages 686–687.

[16] Rabah Amir and Isabel Grilo. Stackelberg versus cournot equilibrium. *Games and Economic Behavior*, 26(1):1–21, 1999.

[17] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010.

[18] Pradeep K. Atrey, M. Anwar Hossain, and Abdulmotaleb El Saddik. Association-based dynamic computation of reputation in web services. *International Journal of Web and Grid Services*, 4(2):169–188, June 2008.

[19] Haris Aziz, Felix Brandt, and Hans Georg Seedig. Computing Desirable Partitions in Additively Separable Hedonic Games. *Artificial intelligence*, 195:316–334, 2013.

[20] Boualem Benatallah, Quan Sheng, and Marlon Dumas. The self-serv environment for web services composition. *IEEE Internet Computing*, 7(1):40–48, 2003.

[21] Djamal Benslimane, Zakaria Maamar, Yehia Taher, Mohammed Lahkim, Marie Christine Fauvet, and Michael Mrissa. A Multi-Layer and Multi-Perspective Approach to Compose Web Services. In *AINA '07: Proceedings of the 21st International Conference on Advanced Networking and Applications*, pages 31–37. IEEE Computer Society, 2007.

[22] Jamal Bentahar, Babak Khosravifar, Mohamed Adel Serhani, and Mahsa Alishahia. On the analysis of reputation for agent-based web services. *Expert Systems with Applications*, 39(16):12438–12450, November 2012.

[23] Saketh Bharadwaja, Weiqing Sun, Mohammed Niamat, and Fangyang Shen. Collabra: a Xen hypervisor based collaborative intrusion detection system. In *ITNG 2011*, pages 695–700.

[24] Philip Bianco, Grace Lewis, and Paulo Merson. Service level agreements in service-oriented architecture environments. Technical report, Software Engineering Institute, 2008.

[25] Ken Binmore and Bruce Linster. *Fun and Games: A Text on Game Theory*, volume 21. DC Heath Lexington, Mass, 1992.

[26] Anna Bogomolnaia and Matthew O Jackson. The stability of hedonic coalition structures. *Games and Economic Behavior*, 38.

[27] George Bojadziev and Maria Bojadziev. *Fuzzy Sets, Fuzzy Logic, Applications*, volume 5. January 1996.

[28] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[29] Thomas M Chen and Varadharajan Venkataramanan. Dempster-shafer theory for intrusion detection in ad hoc networks. *Internet Computing, IEEE*, 9(6):35–41, 2005.

[30] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *NSDI*, pages 273–286. USENIX Association, 2005.

[31] Julio B. Clempner and Alexander S. Poznyak. Stackelberg Security Games: Computing The Shortest-Path Equilibrium. *Expert Systems with Applications*, 42(8):3967–3979, 2015.

[32] F. Groebner David. *Business Statistics: a decision-making approach*. Pearson, 2014.

[33] Shuiguang Deng, Longtao Huang, and Guandong Xu. Social network-based service recommendation with trust enhancement. *Expert Systems with Applications*, 41(18):8075–8084, 2014.

[34] Prachi Deshpande, SC Sharma, SK Peddoju, and S Junaid. HIDS: A host based intrusion detection system for cloud computing environment. *International Journal of System Assurance Engineering and Management*, pages 1–10, 2014.

[35] Nicola Dragoni. A survey on trust-based web service provision approaches. In *Third International Conference on Dependability (DEPEND)*, pages 83–91, Venice, 2010. IEEE.

[36] Nicola Dragoni. A survey on trust-based web service provision approaches. In *Third International Conference on Dependability (DEPEND)*, pages 83–91. IEEE, 2010.

[37] Jacques H. Dreze and Joseph Greenberg. Hedonic Coalitions: Optimality and Stability. *Econometrica: Journal of the Econometric Society*, pages 987–1003, 1980.

[38] Said Elnaffar, Zakaria Maamar, Hamdi Yahyaoui, Jamal Bentahar, and Philippe Thiran. Reputation of communities of web services - preliminary investigation. In *22nd International Conference on Advanced Information Networking and Applications - Workshops*, 2008.

[39] Tom Fawcett. An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[40] Thomas S Ferguson. Game theory. *Mathematics Department, UCLA*, 2008.

[41] Mark A Friedl and Carla E Brodley. Decision tree classification of land cover from remotely sensed data. *Remote Sensing of Environment*, 61(3):399–409, 1997.

[42] Borivoje Furht and Armando Escalante. *Handbook of cloud computing*, volume 3. Springer, 2010.

[43] Diego Gambetta. *Trust: making and breaking cooperative relations*. Oxford Basil Blackwell, 1988.

[44] Inigo Goiri, Jordi Guitart, and Jordi Torres. Characterizing cloud federation for enhancing providers' profit. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 123–130. IEEE, 2010.

[45] Nils Gruschka and Meiko Jensen. Attack surfaces: A taxonomy for attacks on cloud services. In *3rd international conference on Cloud Computing*, pages 276–279. IEEE, 2010.

[46] Nils Gruschka and Norbert Luttenberger. *Security and Privacy in Dynamic Environments*, volume 201 of *IFIP International Federation for Information Processing*. Springer US, 2006.

[47] Barbara Guttman and Edward A Roback. *An introduction to computer security: the NIST handbook*. DIANE Publishing, 1995.

[48] Michael S Hamada, Alyson Wilson, C Shane Reese, and Harry Martz. *Bayesian reliability*. Springer Science & Business Media, 2008.

[49] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: Concepts and techniques*. Elsevier, 2011.

[50] Chung-Wei Hang, Anup K. Kalia, and Munindar P. Singh. Behind the curtain: Service selection via trust in composite services. In *2012 IEEE 19th International Conference on Web Services*, pages 9–16. IEEE, 2012.

[51] Mohammad Mehedi Hassan, Mohammad Abdullah-Al-Wadud, Ahmad Almogren, SK Rahman, Abdulhameed Alelaiwi, Atif Alamri, Md Hamid, et al. Qos and trust-aware coalition formation game in data-intensive cloud federations. *Concurrency and Computation: Practice and Experience*, 2015.

[52] Jay Jacobs and Bob Rudis. *Data-Driven Security: Analysis, Visualization and Dashboards*. John Wiley & Sons, 2014.

[53] Nicholas R Jennings. On agent-based software engineering. *Artificial intelligence*, 117(2):277–296, 2000.

[54] Finn Verner Jensen and Thomas Dyhre Nielsen. *Data Mining: Concepts and techniques*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 3rd edition, July 2011.

[55] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision support systems*, 43(2):618–644, 2007.

[56] Audun JÃ¸sang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007.

[57] Chris Karlof and David Wagner. Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures. *Ad Hoc Networks*, 1(2):293–315, 2003.

[58] Babak Khosravifar, Mahsa Alishahi, Jamal Bentahar, and Philippe Thiran. A game theoretic approach for analyzing the efficiency of web services in collaborative networks. In *2011 IEEE International Conference on Services Computing*, pages 168–175, 2011.

[59] Babak Khosravifar, Jamal Bentahar, Rabeb Mizouni, Hadi Otrok, Mahsa Alishahi, and Philippe Thiran. Agent-based game-theoretic model for collaborative web services: Decision making analysis. *Expert Systems with Applications*, 40(8):3207–3219, June 2013.

[60] Babak Khosravifar, Jamal Bentahar, and Ahmad Moazin. Analyzing the relationships between some parameters of web services reputation. In *2010 IEEE 19th International Conference on Web Services*, pages 329–336, Miami, FL, 2010. IEEE.

[61] Babak Khosravifar, Jamal Bentahar, Ahmad Moazin, Zakaria Maamar, and Philippe Thiran. Analyzing communities vs. single agent-based web services: Trust perspectives. In *2010 IEEE International Conference on Services Computing (SCC)*, pages 194–201, Miami, FL, 2010. IEEE.

[62] Babak Khosravifar, Jamal Bentahar, Ahmad Moazin, and Philippe Thiran. Analyzing communities of web services using incentives. *International Journal of Web Services Research*, 7(3):30–51, 2010.

[63] Ehsan Khosrowshahi-Asl, Jamal Bentahar, Hadi Otrok, and Rabeb Mizouni. Efficient Community Formation for Web Services. *IEEE Transactions on Services Computing*, in press, 2014.

[64] David M. Kreps, Paul Milgrom, John Roberts, and Robert Wilson. Rational Cooperation in the Finitely Repeated Prisoners' Dilemma. *Journal of Economic theory*, 27(2):245–252, 1982.

[65] Tobias Kurze, Markus Klems, David Bermbach, Alexander Lenk, Stefan Tai, and Marcel Kunze. Cloud federation. In *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization*, 2011.

[66] Bo Li, Peng Liu, and Li Lin. A cluster-based intrusion detection framework for monitoring the traffic of cloud environments. In *CSCloud 2016*, pages 42–45.

[67] Hai-Hua Li, Xiao-Yong Du, and Xuan Tian. A review-based reputation evaluation approach for web services. *Journal of Computer Science and Technology*, 24(5):893–900, 2009.

[68] Lei Li and Yan Wang. A subjective probability based deductive approach to global trust evaluation in composite services. In *IEEE International Conference on Web Services*, pages 604–611, Washington, DC, 2011. IEEE.

[69] Lei Li, Yan Wang, and Ee-Peng Lim. Trust-oriented composite service selection and discovery. In *Service-Oriented Computing*, pages 50–67. Springer, 2009.

[70] Erbin Lim, Philippe Thiran, Zakaria Maamar, and Jamal Bentahar. On the analysis of satisfaction for web services selection. In *2012 IEEE Ninth International Conference on Services Computing*, pages 122–129, 2012.

[71] Wenjie Lin and David Lee. Traceback Attacks in Cloud–Pebbletrace Botnet. In *32nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 417–426. IEEE, 2012.

[72] Zhangxi Lin, Dahui Li, Balaji Janamanchi, and Wayne Huang. Reputation distribution and consumer-to-consumer online auction market structure: an exploratory study. *Decision Support Systems*, 41(2):435–448, January 2009.

[73] An Liu, Qing Li, Liusheng Huang, Shi Ying, and Mingjun Xiao. Coalitional game for community-based autonomous web services cooperation. *IEEE Transactions on Services Computing*, 99(3):387–399, 2012.

[74] Ling Liu and Malcolm Munro. Systematic analysis of centralized online reputation systems. *Decision Support Systems*, 52(2):438–449, January 2012.

[75] Chi-Chun Lo, Chun-Chieh Huang, and Joy Ku. A cooperative intrusion detection system framework for cloud computing networks. In *39th international conference on Parallel Processing Workshops (ICPPW)*, pages 280–284. IEEE, 2010.

[76] Flavio Lombardi and Roberto Di Pietro. Secure virtualization for cloud computing. *Journal of Network and Computer Applications*, 34(4):1113–1122, 2011.

[77] Alina Madalina Lonea, Daniela Elena Popescu, and Huanglory Tianfield. Detecting DDoS attacks in cloud computing environment. *International Journal of Computers Communications & Control*, 8(1):70–78, 2013.

[78] Gehao Lu, Joan Lu, Shaowen Yao, and Yau Jim Yip. A review on computational trust models for multi-agent systems. *The Open Information Science Journal*, 2(2):18–25, 2009.

[79] David G Luenberger. *Introduction to linear and nonlinear programming*, volume 28. Addison-Wesley Reading, MA, 1973.

[80] Zhanyu Ma and Arne Leijon. Bayesian estimation of beta mixture models with variational inference. *IEEE Transactions on Pattern Analysis and Machine Intellelligence*, 33(11):2160–2173, November 2011.

[81] Zakaria Maamar, Soraya Kouadri Mostefaoui, and Hamdi Yahyaoui. Toward an agent-based and context-oriented approach for web services composition. *IEEE Transactions on Knowledge and Data Engineering*, 17(5):686–697, 2005.

[82] Zakaria Maamar, Sattanathan Subramanian, Jamal Bentahar, Philippe Thiran, and Djamal Bensilamane. An approach to engineer communities of web services: Concepts, architecture, operation, and deployment. *International Journal of E-Business Research (IJEBR)*, 5(4):1–21, 2009.

[83] Áine MacDermott, Qi Shi, Madjid Merabti, and Kashif Kifayat. Security as a service for a cloud federation. In *15th Post Graduate Symposium on the Convergence*

of Telecommunications, Networking and Broadcasting (PGNet2014), pages 77–82, 2014.

[84] Zaki Malik and Athman Bouguettaya. Rater credibility assessment in web services interactions. *World Wide Web*, 12(1):3–25, March 2009.

[85] Zaki Malik and Athman Bouguettaya. Rateweb: Reputation assessment for trust establishment among web services. *The VLDB Journal*, 18(4):885–911, 2009.

[86] Zaki Malik and Athman Bouguettaya. Reputation bootstrapping for trust establishment among web services. *IEEE Internet Computing*, 13(1):40–47, 2009.

[87] V. Mareeswari and Dr. E. Sathiyamoorthy. A survey on trust in semantic web services. *International Journal of Scientific & Engineering Research*, 3(2):1–5, February 2012.

[88] Lena Mashayekhy, Mahyar Movahed Nejad, and Daniel Grosu. Cloud federations in the sky: Formation game and mechanism. *IEEE Transactions on Cloud Computing*, 3(1):14–27, 2015.

[89] E. Michael Maximilien and Munindar P. Singh. Conceptual model of web service reputation. *SIGMOD Record*, 31(4):36–41, December 2002.

[90] E. Michael Maximilien and Munindar P. Singh. Multiagent system for dynamic web services selection. In *Proceedings of 1st Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE at AAMAS)*, pages 25–29, Utrecht, The Netherlands, 2005.

[91] Brahim Medjahed and Athman Bouguettaya. A dynamic foundational architecture for semantic web services. *Distributed and Parallel Databases*, 17:179–206, 2005.

[92] Mohamad Mehdi, Nizar Bouguila, and Jamal Bentahar. Trustworthy web service selection using probabilistic models. In *Proceedings of the 2012 IEEE 19th International Conference on Web Services*, ICWS '12, pages 17–24, Washington, DC, USA, 2012. IEEE Computer Society.

[93] Mohamad Mehdi, Nizar Bouguila, and Jamal Bentahar. A qos-based trust approach for service selection and composition via bayesian networks. In *2013 IEEE 20th International Conference on Web Services*, pages 211–218, Santa Clara, CAL, 2013. IEEE.

[94] Bruckner Michael and Tobias Scheffer. Stackelberg Games for Adversarial Prediction Problems. In *Proceedings of the 17th ACM SIGKDD International Conference*

on Knowledge Discovery and Data Mining, pages 547–555, San Diego, California, USA, 2011. ACM.

[95] Oskar Morgenstern and John Von Neumann. *Theory of Games and Economic Behavior*. Princeton University Press, 1953.

[96] Guofang Nan, Zhifei Mao, Mei Yu, Minqiang Li, Honggang Wang, and Yan Zhang. Stackelberg Game for Bandwidth Allocation in Cloud-based Wireless Live-streaming Social Networks. *IEEE Systems Journal*, 8(1):256–267, 2014.

[97] John F. Nash. Equilibrium Points in n-person Games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

[98] John F. Nash. Cooperative Game Theory: Basic Concepts and Computational Challenges. *IEEE Intelligent Systems*, 27(3):86–90, 2012.

[99] Surya Nepal, Wanita Sherchan, Jonathon Hunklinger, and Athman Bouguettaya. A fuzzy trust management framework for service web. In *IEEE International Conference on Web Services*, pages 321–328, Miami, FL, 2010. IEEE Computer Society.

[100] Hien Trang Nguyen, Weiliang Zhao, and Jian Yang. A trust and reputation model based on bayesian network for web services. In *2010 IEEE International Conference on Web Services*, pages 251–258, Miami, FL, 2010. IEEE.

[101] Jason Nikolai and Yong Wang. Hypervisor-based cloud intrusion detection system. In *International Conference on Computing, Networking and Communications (ICNC)*, pages 989–993. IEEE, 2014.

[102] Dusit Niyato, Athanasios V Vasilakos, and Zhu Kun. Resource and revenue sharing with coalition formation of cloud providers: Game theoretic approach. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 215–224. IEEE Computer Society, 2011.

[103] Talal H Noor and Quan Z Sheng. Trust as a service: a framework for trust management in cloud environments. In *WISE*, pages 314–321. Springer, 2011.

[104] Martin Osborne. *An Introduction to Game Theory*. Oxford University Press, 1st edition, 2003.

[105] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT press, 1994.

[106] Hadi Otrok, Mona Mehrandish, Chadi Assi, Mourad Debbabi, and Prabir Bhattacharya. Game theoretic models for detecting network intrusions. *Computer Communications*, 31(10):1934–1944, 2008.

[107] Sharon Paradesi, Prashant Doshi, and Sonu Swaika. Integrating behavioral trust in web service compositions. In *IEEE 19th International Conference on Web Services*, pages 453–460, Los Angeles, CA, 2009. IEEE.

[108] Praveen Paruchuri, Jonathan P Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez, and Sarit Kraus. Playing games for security: an efficient exact algorithm for solving bayesian stackelberg games. In *AAMAS 2008*, pages 895–902.

[109] Jyotishman Pathak, Samik Basu, and Vasant Honavar. Assembling composite web services from autonomous components. In *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 394–405, Amsterdam, The Netherlands, 2007. IOS Press.

[110] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. big'web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, pages 805–814. ACM, 2008.

[111] Wei Peng, Feng Li, Chin-Tser Huang, and Xukai Zou. A moving-target defense strategy for cloud-based services with heterogeneous and dynamic attack surfaces. In *IEEE ICC*, pages 804–809, 2014.

[112] Diego Perez-Botero, Jakub Szefer, and Ruby B Lee. Characterizing hypervisor vulnerabilities in cloud computing servers. In *Proceedings of the 2013 international workshop on Security in cloud computing*, pages 3–10. ACM, 2013.

[113] Hebert Pérez-Rosés, Francesc Sebé, and Josep Maria Ribó. Endorsement deduction and ranking in social networks. *Computer Communications*, 73:200–210, 2016.

[114] Suronapee Phoomvuthisarn. A survey study on reputation-based trust mechanisms in service-oriented computing. *Journal of Information Science and Technoogy*, 2(2):1–12, 2011.

[115] James Pita, Manish Jain, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Using Game Theory for Los Angeles Airport Security. *AI MAGAZINE*, 30(1):43–57, 2009.

[116] Niels Provos. A virtual honeypot framework. In *USENIX Security Symposium*, volume 173, pages 1–14, 2004.

[117] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and Communications Security*, pages 199–212, 2009.

[118] Benny Rochwerger, David Breitgand, Eliezer Levy, Alex Galis, Kenneth Nagin, Ignacio M Llorente, Ruben Montero, Yaron Wolfsthal, Erik Elmroth, Juan Caceres, et al. The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):4–1, 2009.

[119] Walid Saad, Zhu Han, Tamer Başar, Mérouane Debbah, and Are Hjorungnes. Hedonic coalition formation for distributed task allocation among wireless agents. *IEEE Transactions on Mobile Computing*, 10(9):1327–1344, 2011.

[120] Noel Sardana and Robin Cohen. Modeling agent trustworthiness with credibility for message recommendation in social networks. In *AAMAS*, pages 1423–1424. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[121] Lloyd Shapley and Herbert Scarf. On Cores and Indivisibility. *Journal of mathematical economics*, 1(1):23–37, 1974.

[122] Lloyd S. Shapley. A Value for n-person Games. In *Contributions to the Theory of Games*, volume 28, pages 307–317. Princeton University Press, 1953.

[123] Lloyd S. Shapley. Cores of Convex Games. *International journal of game theory*, 1(1):11–26, 1971.

[124] Ryan Shea and Jiangchuan Liu. Understanding the impact of denial of service attacks on virtual machines. In *Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service*, pages 1–27. IEEE Press, 2012.

[125] Onn M. Shehory, Katia Sycara, and Somesh Jha. Multi-agent Coordination through Coalition Formation. In *Intelligent Agents IV Agent Theories, Architectures, and Languages*, volume 1365, pages 143–154. Springer, 1998.

[126] Wanita Sherchan, Seng W. Loke, and Shonali Krishnaswamy. A fuzzy model for reasoning about reputation in web services. In *Proceedings of the ACM Symposium on Applied Computing*, SAC '06, pages 1886–1892, New York, NY, USA, 2006. ACM.

[127] Florian Skopik, Daniel Schall, and Schahram Dustdar. Modeling and mining of dynamic trust in complex service-oriented systems. *Information Systems*, (7):735–757, November.

[128] Gaurav Somani, Manoj Singh Gaur, Dheeraj Sanghi, Mauro Conti, and Rajkumar Buyya. DDoS Attacks in Cloud Computing: Issues, Taxonomy, and Future Directions. *ACM Computing Surveys*, 2015.

[129] Anne Randi Syversveen. Noninformative bayesian priors. interpretation and problems with construction and applications. *Preprint Statistics*, 3, 1998.

[130] Nicholas A. Thurow and John D. Delano. Selection of web services based on opinion mining of free-text user reviews. In *Proceedings of the International Conference on Information Systems*, page 42, St. Louis, MO, 2010. Association for Information Systems.

[131] Thomas T. Tran, Robin Cohen, and Eric Langlois. Establishing trust in multiagent environments: Realizing the comprehensive trust management dream. In *TRUST@AAMAS*, volume 1740 of *CEUR Workshop Proceedings*, pages 35–43, 2014.

[132] Kristal K. Trejo, Julio B. Clempner, and Alexander S. Poznyak. A Stackelberg security game with random strategies based on the extraproximal theoretic approach. *Engineering Applications of Artificial Intelligence*, 37:145–153, 2015.

[133] John W Tukey. Exploratory data analysis. 1977.

[134] Udaya Tupakula, Vijay Varadharajan, and Naveen Akku. Intrusion detection techniques for infrastructure as a service cloud. In *EEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC)*, pages 744–751. IEEE, 2011.

[135] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 228–235. IEEE, 2010.

[136] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[137] Sokratis Vavilis, Milan Petkovic, and Nicola Zannone. A reference model for reputation systems. *Decision Support Systems*, 61:147–154, May 2014.

[138] Heinrich von Stackelberg. *Marktform und Gleichgewicht*. Springer-Verlag Wien New York, 1934.

[139] Heinrich von Stackelberg. *Market Structure and Equilibrium*. Springer Berlin Heidelberg, translation edition, 2011.

[140] Omar Abdel Wahab. Cooperative clustering models for Vehicular Ad Hoc Networks. Master's thesis, Lebanese American University, 2013.

[141] Omar Abdel Wahab, Jamal Bentahar, Hadi Otrok, and Azzam Mourad. A survey on trust and reputation models for Web services: Single, composite, and communities. *Decision Support Systems*, 74:121–134, 2015.

[142] Omar Abdel Wahab, Jamal Bentahar, Hadi Otrok, and Azzam Mourad. Misbehavior detection framework for community-based cloud computing. In *FiCloud*, pages 181–188. IEEE, 2015.

[143] Omar Abdel Wahab, Jamal Bentahar, Hadi Otrok, and Azzam Mourad. How to distribute the detection load among virtual machines to maximize the detection of distributed attacks in the cloud? In *2016 IEEE International Conference on Services Computing (SCC)*, pages 316–323. IEEE, 2016.

[144] Omar Abdel Wahab, Jamal Bentahar, Hadi Otrok, and Azzam Mourad. A stackelberg game for distributed formation of business-driven services communities. *Expert Systems with Applications*, 45:359–372, 2016.

[145] Omar Abdel Wahab, Jamal Bentahar, Hadi Otrok, and Azzam Mourad. I know you are watching me: Stackelberg-based adaptive intrusion detection strategy for insider attacks in the cloud. In *IEEE ICWS*, pages 728–735. IEEE, 2017.

[146] Omar Abdel Wahab, Jamal Bentahar, Hadi Otrok, and Azzam Mourad. Optimal load distribution for the detection of VM-based DDoS attacks in the cloud. *IEEE Transactions on Services Computing*, 2017.

[147] Omar Abdel Wahab, Moulay Omar Hachami, Arslan Zaffari, Mery Vivas, and Gaby G. Dagher. DARM: a privacy-preserving approach for distributed association rules mining on horizontally-partitioned data. In *Proceedings of the 18th International Database Engineering & Applications Symposium*, pages 1–8. ACM, 2014.

[148] Omar Abdel Wahab, Azzam Mourad, Hadi Otrok, and Jamal Bentahar. CEAP: SVM-based intelligent detection model for clustered vehicular ad hoc networks. *Expert Systems with Applications*, 50:40–54, 2016.

[149] Omar Abdel Wahab, Hadi Otrok, and Azzam Mourad. VANET QoS-OLSR: QoS-based clustering protocol for Vehicular Ad hoc Networks. *Computer Communications*, 36(13):1422–1435, 2013.

[150] Omar Abdel Wahab, Hadi Otrok, and Azzam Mourad. A cooperative watchdog model based on Dempster-Shafer for detecting misbehaving vehicles. *Computer Communications*, 41:43–54, 2014.

[151] Omar Abdel Wahab, Hadi Otrok, and Azzam Mourad. A Dempster-Shafer Based Tit-for-Tat Strategy to Regulate the Cooperation in VANET Using QoS-OLSR Protocol. *Wireless Personal Communications*, 75(3):1635–1667, 2014.

[152] Omar Abdul Wahab, Jamal Bentahar, Hadi Otrok, and Azzam Mourad. Towards trustworthy multi-cloud services communities: A trust-based hedonic coalitional game. *IEEE Transactions on Services Computing*, 2016.

[153] Hongbing Wang, Bin Zou, Guibing Guo, Jie Zhang, and Danrong Yang. Integrating trust with user preference for effective web service composition. *IEEE Transactions on Services Computing*, 10(4):574–588, 2015.

[154] Wei Wang, Ben Liang, and Baochun Li. Multi-resource fair allocation in heterogeneous cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 26(10):2822–2835, 2015.

[155] Yao Wang and Julita Vassileva. Toward trust and reputation based web service selection: A survey. *International Transactions on Systems Science and Applications (ITSSA) Journal*, 3(2), 2007.

[156] J Scott Ward and Adam Barker. Varanus: In situ monitoring for large scale cloud systems. In *5th International Conference on Cloud Computing Technology and Science (CloudCom)*, volume 2, pages 341–344. IEEE, 2013.

[157] Larry Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer Publishing Company, Incorporated, 2010.

[158] Michael R Watson, Angelos K Marnerides, Andreas Mauthe, David Hutchison, et al. Malware detection in cloud computing infrastructures. *IEEE Transactions on Dependable and Secure Computing*, 13(2):192–205, 2016.

[159] Guiyi Wei, Athanasios V Vasilakos, Yao Zheng, and Naixue Xiong. A game-theoretic method of fair resource allocation for cloud computing services. *The journal of supercomputing*, 54(2):252–269, 2010.

[160] Eyal Winter. On Non-Transferable Utility Games with Coalition Structure. *International Journal of Game Theory*, 20(1):53–63, 1991.

[161] Michael Wooldridge. Agent-based software engineering. *IEE Proceedings-software*, 144(1):26–37, 1997.

[162] Bing Wu, Jianmin Chen, Jie Wu, and Mihaela Cardei. A survey of attacks and countermeasures in mobile ad hoc networks. In *Wireless Network Security*, pages 103–135. Springer, 2007.

[163] Hamdi Yahyaoui. Trust assessment for web services collaboration. In *IEEE International Conference on Web Services*, pages 315–320, Miami, FL, 2010. IEEE.

[164] Hamdi Yahyaoui. A trust-based game theoretical model for web services collaboration. *Knowledge-Based Systems*, 27:162–169, 2012.

[165] Bin Yu and Munindar P Singh. An evidential model of distributed reputation management. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 294–301. ACM, 2002.

[166] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *Proceedings of the 12th international conference on World Wide Web, ser. WWW '03*, pages 411–421, 2003.

[167] Jin Zhang and Qian Zhang. Stackelberg Game for Utility-Based Cooperative Cognitive Radio Networks. In USA New Orleans, LA, editor, *Proceedings of the Tenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 23–32. ACM, 2009.

[168] Tao Zhang, Jianfeng Ma, Cong Sun, Qi Li, and Ning Xi. Service composition in multi-domain environment under time constraint. In *IEEE International Conference on Web Services*, pages 227–234, Santa Clara, CA, 2013. IEEE.

[169] Rong Zhou and Eric A Hansen. Breadth-first heuristic search. *Artificial Intelligence*, 170(4):385–408, 2006.