DECEPTION IN CYBERSPACE: CON-MAN ATTACK IN CLOUD SERVICES

A Dissertation
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Md. Minhaz Chowdhury

In Partial Fulfillment of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

Major Department:
Computer Science

June 2018

Fargo, North Dakota

North Dakota State University
Graduate School

Title

DECEPTION IN CYBERSPACE: CON-MAN ATTACK IN CLOUD SERVICES

By

MD. MINHAZ CHOWDHURY

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

DOCTOR OF PHILOSOPHY

SUPERVISORY COMMITTEE:

Dr. Kendall E. Nygard
Chair

Dr. Saeed Salem

Dr. Oksana Myronovych

Dr. Jacob Glower

Approved:

6/22/2018	Dr. Kendall E. Nygard
Date	Department Chair

ABSTRACT

A con-man deception appears in services from cyberspace, e.g., in cloud services. A cloud-service provider deceives by repeatedly providing less service than promised and deliberately avoids service monitoring. Such a repeated shortfall is beneficial for the cloud-service provider but victimizes the service's legitimate consumers. This deception is called a con-man deception. A con-man-resistant trust algorithm is used as a proactive measure against such deception, reducing the deception's severity on the victim's end. This trust algorithm detects a con-man deception by evaluating a cloud service's expected versus actual behavior. This detection application reveals the con-man-resistant trust algorithm's previously veiled properties. With this dissertation, a study of these properties reveals some necessary enhancements for this algorithm. The previous con-man-resistant trust-algorithm applications only considered the pattern of service-shortfall repetition. However, for cloud applications, the service-shortfall magnitude at each repetition is also important. Hence, an exponential growth-function-based extension of this algorithm is proposed and implemented. The algorithm's initial parameter configuration has a significant influence on the con-deception detection pace. Some consumers tolerate intense repetition of service shortfall, and some consumers can tolerate mild repetition. Hence, the deception-detection pace has a correlation with the consumer's perspective. A machine-learning extension of the con-man-resistant trust algorithm can ascertain a consumer's perspective by analyzing that consumer's historical usage of the same cloud service. The result of this learning is a parameter configuration that reflects the consumer's perspective. The loss associated with a con deception is significant on the consumer's side. Hence, the work presented in this dissertation contributes to cybersecurity by attempting to minimize such deception in cyberspace.

ACKNOWLEDGEMENTS

I would, first, like to thank my immediate family members, my parents, my brother, and my sisters, for supporting me while I pursued this degree. I would also like to thank Dr. Kendall E. Nygard for his support and advice while completing my research. In 2009, I started my M.S. under Dr. Nygard as a graduate research assistant. In 2012, I went to the software industry and completed my M.S. degree in 2013. At the end of 2013, I enrolled as a part-time Ph.D. student with Dr. Nygard as my academic adviser. In 2015, I left the software industry and became a full-time student. Dr. Nygard helped me arrange an assistantship for my Ph.D. program. His involvement with cybersecurity motivated me to continue research in cybersecurity, and as result, my Ph.D. dissertation topic falls under the cybersecurity umbrella.

I would also like to thank my dissertation committee’s members: Dr. Saeed Salem, Dr. Oksana Myronovych, and Dr. Jacob Glower. Finally, I would like to thank the Computer Science Department at North Dakota State University and Joseph Latimer for providing the funding for my graduate study.

DEDICATION

When I was very young, I listened to the villagers' stories about my grandfather. Some of the stories are local tales now. I used to read his books about the physics of "light" from the early 1900s and felt wonder at his chemistry lab. I was born about 100 years after him. I went to the same high school that he did, except that the timeline was about 100 years apart. His B.S. degree and my Ph.D. degree awarding time are also a little more than 100 years apart. His legends motivated me to come to the USA, to know the USA's industry culture, and to achieve a Ph.D.

This doctoral disquisition is dedicated to my grandfather, the late Alfazuddin Chowdhury, and my grandmother, the late Mehrun Nissa Chowdhury.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1. GENERAL INTRODUCTION

1.1. Introduction

A con man, or confidence man, has a sequence of interactions with a victim [1][2][3][4][5][6][7]. The con man carries out consecutive cooperative interactions (cooperation) or good transactions, elevating the victim's trust. When high trust is achieved, the con man defects or defrauds the victim by engaging in a transaction that is bad for the user. The con man regains the victim's trust with a similar series of cooperating, or good, transactions. The con man then deceives the victim again. This bad transaction following good transaction behavior is cyclic, regaining trust with good transactions and misusing this trust by engaging in bad transactions. This cycle is a con-man trick and is also a type of deception.

This con-man behavior can appear in the service-oriented architecture, e.g., the cloud services [4][5][6][7]. The work in this dissertation contributes by, first, modeling the con behavior in a cloud environment and then applying the con-man-resistant trust algorithm to detect the cloud services' con behavior. The algorithm uses the con-man-resistant model which is aimed at measuring a trust value. The implemented algorithm measures and monitors the cloud-service provider's con deception in the form of actual versus standards of performance for the contracted cloud services. Until now, there was no detection methodology for the cloud service's con deception.

The con-man-resistant trust algorithm generates a trust value that depends upon measuring the metric values that are based upon monitoring the interactions between two agents. This trust value reflects the perception of one agent from the second agent's view. If the first agent does not show a lapse in expected behavior to the second one, it is referred to as defection. If the first agent meets the second agent's need or provides the expected behavior to the second agent, then it is referred to as a cooperation. Trust values are increased when cooperation happens and decreased when a defection happens. The changes are non-linear and depend upon prior rates of occurrence for the cooperation and the defection.

The efficiency of the implemented con-man-resistant trust algorithm depends on how well it can reflect the consumer's perspective of trust for the service. This reflection further depends upon how aligned the algorithm's parameters are with the consumer's trust perspective. Hence, in this presented work, the parameters' characteristics are studied, analyzed, and explained in detail. Also, a suitable machine-learning algorithm can learn the algorithm's configuration or parameter alignment for a specific consumer. The dissertation further contributes by proposing and implementing a machine-learning algorithm where the con-man-resistant trust algorithm learns about the consumer's choice from that consumer's historical service use. The implemented algorithm has the ability to classify real-time data as trustworthy or untrustworthy, based on the specific consumer's perspective, after learning via the training phase. As a result, the algorithm achieves context sensitivity with parameter tuning or algorithm configuration.

Again, for the cloud services, the consumer is concerned about the service's performance magnitude. As another contribution, for the con-man-resistant trust algorithm, the different aspects of this performance are included in order to detect the cloud service's con-deception actions. The parameter's characteristics, analysis, and explanation are the other contributions of the presented study. The dissertation also recommends different possible applications of the con-man-resistant trust algorithm in a cloud-computing environment.

In summary, the research first studied and modeled the con deception in a cloud-computing environment. Then, the work examined and extended the con-man-resistant trust algorithm as a mitigation technique against the cloud-service provider’s con deception. The dissertation studied the principles that this algorithm follows as well as the algorithm’s properties. The project further enhanced the mitigation algorithm’s efficiency by learning the consumer’s perspective of the trust. The significance of the con-man-resistant trust algorithm is determining a cloud service’s instability as well as its service deficiency.

1.2. Significance of the Research

The significance of the presented work is that is provides both the consumer and the service provider with a context-sensitive tool to detect a cloud service’s deficiency by identifying an unstable cloud-service behavior as a con deception. The tool can be applied by both the consumer and the service provider. The consumer can automatically monitor a cloud service’s performance by using the con-man-resistant trust algorithm as a tool. The service provider can also monitor its business services using this tool. The service provider can revise investment decisions by reviewing the business-service evaluation score given by the tool.

Another significance is that consumers or producers can tune the algorithm’s parameters, depending upon their choice of flexibility (e.g., flexible consumer versus conservative consumer). If the consumers or producers want the deficiency tolerance to be reflected in the service-deficiency detection as a con deception following their perspective, they can apply the recommended machine-learning version of the con-man-resistant trust algorithm.

The con behavior has economic effects because it incurs financial loss, directly or indirectly. Hence, the presented work has significance for cloud economics because the cloud’s con-man-resistant trust algorithm for prevents loss by early detection of the con deception.

The con-man-resistant trust algorithm was first applied to improve a smart electrical grid’s reliability [3]. The con-man-resistant trust algorithm is used to find stable electrical grid node. In terms of an electrical grid, the con-man-resistant trust algorithm states that the less stable a node’s frequency is, the more it affects the components of the electrical grid. Similarly, the cloud service can be improved by applying the con-man-resistant trust algorithm as an evaluation method. This application problem can be restated as the less stable the service quality is, the more it affects the service’s consumer.

1.3. Motivation for the Research

This work is motivated by the business effect of repeated service deficiencies in a service-oriented architecture. Further study suggests that repeated service shortfalls can be a result of the service provider’s deception. When a repeated service shortfall appears as a deception, then the deception is identical to a con-man attack. The consumer or the cloud-service provider can improve awareness about unstable cloud-service behaviors or con deceptions by reviewing the referenced examples of the con behavior’s appearance in service-oriented architecture and such behavior’s effect in business. Similar concepts which are equivalent to the con-man deception in the service-oriented architecture are as follows: the cloud-service provider predicts a consumer’s usage pattern for the service and allocates resources according to the pattern; there is a repeated service-shortfall behavior by the provider in a peer-to-peer network; the peer follows an oscillation of a good and acceptable reputation with the goal of profit maximization; and there cyclic patterns of good and bad transactions by the service provider in a distributed network. Repetitive bad service reduces the consumer number of a business. Real-life examples of such business loss are presented in this work. Examples are that softlayer.com lost its consumers [8]; some users of the Animato site left permanently [9]; and the social-networking and gaming site friendster.com failed because its consumers left gradually due to repeated slow responses [10].

The repeated service deficiency in cloud services is an unstable service behavior. This instability is comparable with an unstable electrical node where the node's frequency fluctuates beyond a limit and fluctuation repeats. The con-man-resistant trust algorithm identifies the fluctuating, or unstable, behavior as a con [3]. The con-man trust value works as a node's reliability measure. A similar concept can be used to identify stable cloud services.

1.4. Peer-Reviewed, Published Papers and Their Relationship

Six papers are published for this dissertation. Four papers are specifically about the con-man-deception application. With these four papers, Minhaz Chowdhury is responsible for the innovations described, but the dissertation's academic adviser is a co-author. The rest two papers mentions about the con-man-resistant trust algorithm application in cloud computing. However, these two papers did not present any new research. Hence, these two papers are connected to the dissertation but in less-direct ways.

The first paper's title is "Deception in Cyberspace: An Empirical Study on a Con Man Attack" [4]. Chapter 2 presents this paper. First, the possibility of the con-man deception as a repetitive performance-degradation issue is presented. Cloud software as a service (SAAS) and infrastructure as a service (IAAS) are simulated to represent the cloud services. SAAS simulation represented a web-service call that provided a service at each fixed interval. Hence, the resulting data are a time series.

The con-man attack is simulated by adding server overloads and following a Poisson inter-arrival time. Due to the overload, the web-service task submitted at that time needed more time to complete, resulting in a longer response time. This time is the performance degradation. Hence, at each Poisson inter-arrival time, a performance degradation happened, simulating the repetition of service shortfalls.

With IAAS, fewer resources are provided at each Poisson inter-arrival time. The result is time-series data with repeated under-provision. This recurring under-provision is the IAAS simulation of a con-man attack. After the simulation, the con-man-resistant trust algorithm is implemented to catch the cloud service's con behavior. The cloud-service provider is accused of this deception because the tradeoff between a cloud provider's investment and service performance can eliminate many repeated shortfalls.

In prior con-man-resistant trust algorithm research, the algorithm's parameter settings were recommended [1][2][3]. In this dissertation, these recommended parameter settings is applied. The con-man-resistant trust algorithm's performance is compared with another trust model [11]. The comparison proves the con-man-resistant trust algorithm's efficiency to spot the con deception in cloud environment.

At the end of Chapter 2, a time-window-based version of the con-man-resistant trust algorithm is proposed and implemented. The characteristics of this time-window-based version algorithm are also presented. However, this time-window-based algorithm is not a part of Paper 1. When the work presented in the first paper was in progress, it became obvious to study the con-man-resistant algorithm's properties for proper application of the algorithm in the cloud-computing domain.

Studying the properties revealed some shortcomings for previous versions of the con-man-resistant trust algorithm. One shortcoming is the algorithm's calculated trust value is dependent on parameter values from last few interactions rather only the last interaction. Another shortcoming is not to include the cloud service's particular issue, e.g., performance-degradation amount. The last shortcoming is this algorithm's inability to learn the consumer's preference. The subsequent papers overcame these shortcomings.

The second paper's title is "An Empirical Study on a Con Resistant Trust Algorithm for Cyberspace" [6]. In this paper, the properties of the con-man-resistant trust algorithm for a cloud service are studied, and an enhancement is proposed. The study resulted in a set of principles that drive the algorithm. The principles give insight about this algorithm's configuration during its application in the cloud service's con-deception pattern detection. The con-detection pace is dependent upon the configuration.

The third paper's title is "Deception in Cyberspace: Performance Focused Con Resistant Trust Algorithm" [5]. In this paper, a performance-degradation-focused, con-man-resistant trust algorithm extension for the cloud services is developed and evaluated. With the previous con-man-resistant trust algorithm, only the service shortfall frequency was considered. In this extension, the service-shortfall magnitude is also included.

The service-shortfall magnitude is exponentially related to the penalty associated with the shortfall. The penalty amount affects the con deception's pace. Hence, the higher the shortfall magnitude, the higher the penalty is, resulting in a shorter time to detect the con. Three exponential growth functions are used to model the shortfall and penalty relationship. Each function is applicable in specific cloud scenarios. Two more authors were added to this paper; they reviewed the finished work to suggest possible enhancements.

The fourth paper's title is "Machine Learning Within a Con Resistant Trust Model" [7]. The methodology to automatically configure the con-man-resistant trust algorithm for a particular user is developed. A machine-learning algorithm is proposed and implemented to discover the consumer's preference from his or her historical data for using a certain cloud service. The output of this machine-learning algorithm is a parameter setting of the con-man-resistant trust algorithm that reflects the consumer's preference. The parameter tuning, or algorithm configuration, for a specific situation makes the algorithm context sensitive.

One of the loosely coupled papers is "Trust and Purpose in Computing" which was published as a conference proceeding [12]. The paper claims that an automated trust calculation can be part of the system design. The con-man-resistant trust algorithm for cloud services which is presented in this dissertation is described in a section of this paper. However, the paper does not go into the implementation details and simulation results. Hence, this paper is not presented in the dissertation.

Another paper is "People and Intelligent Machines in Decision Making" which was published in a journal's special issue [13]. This paper first explains the benefits of artificial-intelligence-based machines. In this paper, the con-man-trust algorithm for the cloud domain is added as a section, no implementation or simulation results are presented. Hence, this paper is loosely connected to the dissertation and is not part of this dissertation.

1.5. Organization of the Dissertation

Chapter 2 presents the research work completed in the first paper titled "Deception in Cyberspace: An Empirical Study on a Con Man Attack." Chapter 3 presents the second paper: "An Empirical Study on Con Resistant Trust Algorithm for Cyberspace." Chapter 4 presents an extension to the con-man-resistant trust algorithm for a cloud service. Chapter 4's contents are from the third paper titled "Deception in Cyberspace: Performance Focused Con Resistant Trust Algorithm." Chapter 5 presents the fourth paper: "Machine Learning Within a Con Resistant Trust Model." Finally, Chapter 6 presents the Conclusion.

CHAPTER 2. AN EMPIRICAL STUDY ON A CON-MAN ATTACK

2.1. Introduction

Deception happens in social-life which is now omnipresent in cyber domains. As a result, deception is also evident in cyberspace. Deception in cyberspace is diverse and often differs from traditional social deceptions. This deception has a cost or a loss which is often incurred by the deception's victim. Hence, it is important to detect deceptions and to protect assets. In this dissertation, empirical work and analysis of the con-man-attack deception are conducted. This work is in the context of the cloud services.

The conceptual model of a con-man-attack deception for the cloud-computing environment is designed and implemented with a proposed solution. The cloud-service consumer's firsthand experience is used to evaluate the deception. The cloud-service consumers can be victims of this deception. The cloud-service providers that exhibit the con-man behaviors may do so to maximize profits or to minimize costs. The losses may be directly financial or subtle, such as a loss of reputation, customer base, or good will. In this chapter, a con-man-resistant trust algorithm is applied to detect the cloud-service provider's deception. The detection results are contrasted with a another trust algorithm [11], revealing that the con-man-resistant trust algorithm provides the best results.

The presented algorithm measures and monitors the cloud service provider's deception[4][5][6][7]. Consumers can carefully choose and develop the conditions in their service-level agreement (SLA) with a cloud-service provider after reviewing the service provider's trust history (provided that the history is available to the consumers).

Using trust can reduce the cost of a consumer's investment or the cost to verify the integrity of a submitted cloud task. For example, if the consumer uses a cryptographic algorithm for such verification, then the verification overhead (the computing cost) is 10^3 to 10^9 times more than the task's computing cost [14][15]. The overhead of trust calculation, as a contrast, is a much smaller amount. These term pairs are used interchangeably for the rest of this chapter: con man and cloud-service provider, victim and consumer, and trust and computational trust.

This chapter is organized into six sections. Section 2.2 describes the work's objective as a Problem Statement. Section 2.3 presents the motivation for the work. Section 2.4 describes the concepts related to the problem. Section 2.5 presents the solution for sub problem 1, i.e., detecting the cloud-service provider's cyclic fraud behavior by using the con-man-resistant trust algorithm with simulated cloud-services data. Section 2.6 uses these simulated data to solve sub problem 2: comparing this con-man-resistant trust algorithm's performance with another comparable trust algorithm. Finally, Section 2.7 presents the solution for sub problem 3, i.e., considering a batch of interactions between the consumer and the cloud service, and calculates the con-man-resistant trust value for each such batch in order to detect the deception. Section 2.9 concludes this chapter.

2.2. Problem Statement

Deception happens in the realm of cyberspace. Often, the deceiver benefits, and the victims incur losses. Thus, it is important to detect deception and to differentiate between deceivers and non-deceivers [16]. However, a common characteristic of deception is effort on the deceiver's part to conceal the deceit [17]. People may not be skilled at detecting the deception [18]. Hence, computer-aided tools to detect deception are useful. Several approaches have been suggested to detect the deception [18]. One approach described in [18] is to create a deception model and a deception-detection process by synthesizing applicable theories. This basic approach is followed and extended in this chapter.

The goal for this chapter is to empirically investigate the behavior of a specific deception called the con-man attack in a specific cyberspace environment. The cloud service is chosen as an example cyberspace environment. The problem in terms of the cloud services is the cloud-service provider's con-man-attacker behavior detection. The context, then, is the upholding or lapsing of the cloud resource's performance relative to quality of service metrics. This lapse in the cloud-service performance incurs a loss on the consumer's end.

The cloud-service provider's con-behavior formalization is inspired by the peers' short-change deception in a peer-to-peer network [19]. In this network, one peer's central processing unit cycles can be shared with another peer. Unfortunately, there are deceptive peers which provide fewer resources than promised.

In this dissertation, the repetition of this short-change deception is identified as the cloud-service provider's cyclic fraud behavior. This short changing is the repeated under-provisioning scenario for IAAS and the repeated, unexpected response time for SAAS.

This chapter's goal is to use the con-man-resistant trust algorithm to identify the cloud-service provider's cyclic fraud behavior. The problem can be further divided into sub problems:

- Sub problem 1: Applying the con-man-resistant trust algorithm to identify the cloud-service provider's cyclic fraud behavior by using the simulated cloud-service data. The conceptual model of the con-man deception for a cloud-computing environment is designed and implemented with a proposed solution.
- Sub problem 2: Contrasting this algorithm's performance with another trust algorithm by using simulated cloud-service data.

- Sub problem 3: Applying this algorithm when trust is updated for each batch of interactions between a cloud service and its consumer by using the simulated cloud-service data.

2.3. Motivation for the Empirical Study

This chapter identifies and models of the trust concept as the metric of interest in determining a cloud service's con behavior. Two cloud services are considered: software as a service (SAAS) and infrastructure as a service (IAAS). The trust is evaluated for each service's specific quality attributes. Each attribute has its own quality metrics. The quality attributes and the corresponding quality metrics vary between cloud services and cloud organizations.

Examples of the quality attributes include availability, performance, scalability, reliability, reusability, and sustainability [20]. Examples of the quality metrics are service-response time for availability, CPU speed for performance, and the mean time to failure for reliability [11][20] [21]. The selected quality attributes for this work are availability and performance; the corresponding metrics are, respectively, service-response time or latency of service (in milliseconds) for SAAS [11] and CPU speed (in millions of instructions per second, MIPS) [22][23][24] for IAAS. These two metrics are measured following the study presented in [11][22].

This chapter contributes by providing the design and a simulation of a method to measure and monitor a cloud-service provider's deception. The work is divided into two areas. The first area is designing two deception models which occur in a cloud environment. One deception is assigning priority to the consumer requests in proportion to the consumers' revenue contract, which may result in substandard service for lower-revenue consumers. The cloud-service provider is assiduously seeking to maximize revenue by limiting the cost of the provided services. The second area is applying metrics that measure these deceptions in the form of actual versus expected service behavior. This metric is referred to as the con-man-resistant trust algorithm.

Figure 1 shows an example of a service shortfall in SAAS. The actual versus expected service for the service-response time is shown. In this figure, “t” represents the time when a service is observed or received by the consumer.

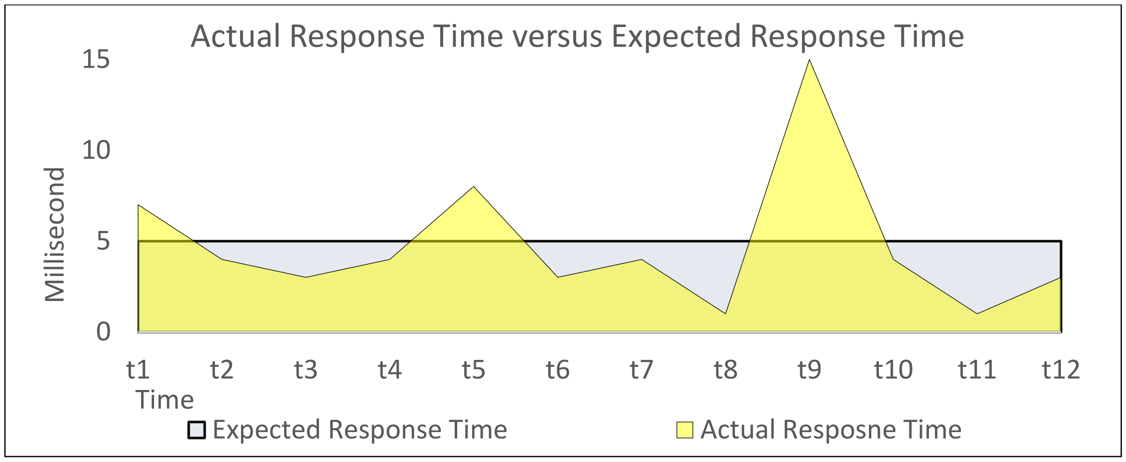


Figure 1. Example of Actual versus Expected Response Time for Service.

The service is measured by the response time. The consumers expect the service time, or response time, to be 5 milliseconds each time they receive service. When the response time is above 5 milliseconds, then the consumers wait more than they expect; hence, this service is considered as an unexpected service quality. Similarly, when the response time is equal to or below 5 milliseconds, then the response time is considered to match the consumers’ expected service quality. The figure conceptualizes the repletion of unexpected service quality. As time increments go by (t_1, t_2, …, t_{12}), the response time fluctuates; hence, the response time at some time stamps does not meet the consumers’ given threshold; that is, the consumers’ experienced service quality is unexpectedly low. This scenario happens when the service provider does not deliver the promised service, regardless of the source for this unexpected service quality. This event is the consumers' firsthand experience. One important characteristic of a cloud service's response-time issue is that the response time grows exponentially when the server is overloaded [25].

A similar example of a repeated service shortfall is described in [9] and as Figure 2 (p. 11). The example given in [9] depicts a repeated IAAS service shortage. Each shortfall is known as under-provisioning. The figure shows a schematic representation of this repeated under-provision. In this figure, the under-provision happens when the resource demand surpasses the service provider's capacity.

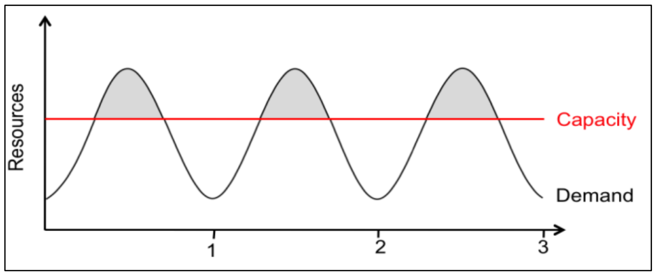


Figure 2. Repeated Under-Provision.

A cloud-service provider can predict the consumer's usage pattern for the service and can allocate resources according to the pattern, rather allocating all the time [26][27]. Therefore, if the consumer changes the pattern, he or she experiences a repeated service shortfall.

Again, the provider shows repeated service-shortfall behavior in a peer-to-peer network [19]. The oscillating-reputation issue is present in a peer-to-peer network [28]. A peer exhibits good service to achieve a good reputation, followed by acceptable service until the reputation touches the reputation's lower bound. This oscillating reputation maximizes the profit. A cyclic pattern of good and bad transactions is camouflaged by a service provider and from the consumer in a distributed network in order to deceive consumers [29].

The work is motivated by the loss associated with such con behavior. There is a significant loss from the con behavior because it decreases the number of consumers for a business. For example, softlayer.com lost its consumers due to repetitive bad service [8]; some users of the Animato site left permanently due to repetitive unsatisfactory service [9]; the social networking and gaming site friendster.com failed because its consumers left gradually due to repeated slow responses [10].

In this dissertation, these two scenarios (service shortfall for two cloud services: SAAS and IAAS) are implemented. The cloud services are simulated by utilizing the Cloudsim toolkit in order to obtain the results [30][31]. Finally, the approach's effectiveness is established by comparing the simulation results with prior work in the same domain.

2.4. Related Concepts

2.4.1. Deception in Cyberspace

The realm of deception is not new. Deception has been studied and defined in earlier scholarly works. Deception can occur in the form of agreement violations if there is an agreement between the two parties [32]. The medium of deception in this work is the cloud services, a prototypical and common domain in cyberspace. The concepts and the approach apply to many cyberspace domains.

A cloud-service provider is a profit-driven organization. If deception benefits the provider's profit, then it is possible for the provider to be involved with deception. Table 1 summarizes the service-provider deception research, including the cloud-service provider's deception.

Table 1. Service-Provider Deception.

Cloud-Service Provider Deception	Deception Description
A cloud-service provider colludes with another service provider to deceive [14].	A cloud-service provider can deceive by avoiding the computation part of a complex calculation for a submitted task. Identifying this avoidance can be costly. A relatively cost-effective way is to submit the computation to two cloud-service providers and then to compare the results. However, the two service providers can engage in collusion by cooperating with each other and deceiving the consumer with the verification process. Using a smart contract and crosschecking the computation with these two clouds, verifiability can be achieved in a cost-efficient way without the collusion and corner-cutting activity of these two cloud-service providers. The smart contract is called the traitor's contract; the service provider reveals the collusion information and receives more benefit than the collusion contract benefits the two service providers.
Less resource provision [33][34]	A cloud-service provider deceives by providing less CPU and memory resources for a virtual machine than promised in the SLA.
Deceives the auditing process [26][27]	The service provider only follows the SLA at the time stamps when the auditor will check the quality. The service provider also tracks the consumer's usage pattern and provides the required resources described in the SLA for the time stamps when the consumer utilizes the resources.
False quality of service advertisements [36]	There is the possibility of false quality of service advertisements from the cloud-service providers. The cloud-service providers can advertise an exaggerated service performance with respect to the real amount which they can provide in order to attract consumers. A cloud service can be selected based on different attributes; one attribute is the resource amount. The exaggerated resource amount biases the selection process because the number is input into this selection algorithm.
Data-storage deception [35]	Consumers can use a proposed cryptographic idea to confirm the integrity of the stored data even if the cloud-storage service deceives. The consumer stores both the data and a hash function that he or she generates with the data. The service provider can only save the hash key or throw away the data. Generating the hash from the data is resource intensive for the service provider. Another conceptual example of the possible deception is when service providers, such as Amazon S3, claim to have three copies of the consumer data. Amazon can only store one copy of the data but sent the same copy three times to mimic that it has three copies of the same data.
Denying authorized user and manipulating user ciphertext [39]	The cloud-service provider and cloud server are interchangeable entities. The cloud server can be involved with two deceptions. In the first type, the service provider deceives by manipulating the consumer's ciphertext (when the server is compromised). In the second type, the cloud-service provider can deceive consumers to save cost by reducing the computation resource amount. The legitimate or authorized user is denied access to the service, reducing the cost. This access denial will reduce the number of users and save cost.
Data stored in a lower-cost data center [40]	A cloud-service provider stores data in different locations than promised, e.g., at a lower-cost data center instead of the promised data center.
Reduced computation resources [37]	Cloud servers can deceive about computation to reduce the computation-resource load but still claim the credit. Also, the cloud server can be compromised by doing unreliable computations. The cloud provider's choice to hide computation makes these issues worse. The stored data are manipulated in the storage-deceiving attack model. The cloud service performs the wrong computation in the computation-deceiving attack model. The confidential data are compromised in the privacy-deceiving attack.
Virtual machine CPU speed shortage or reduction [38]	The cloud-service provider can be deceptive about the virtual machine's CPU speed. This deception helps the service provider to support more users compared to cases when the service provider delivers the exact resource amount. The user incurs a loss here because his or her submitted computation will take more time to finish. In addition to this deception, the service provider tempers the logged conversion time.
Shared-resource shortfall [19]	In a peer-to-peer network, a peer sharing its resources with another peer can deceive by sharing a smaller resource amount than promised. In this dissertation, it is argued that the same deception is possible for a cloud-service provider.
Oscillating service quality [28]	Peers can cheat by building a high reputation at the beginning. Then, they can start cheating, occasionally, in such a way that the cheating does not affect their existing reputation very much. Another cheating option is an oscillation of building reputation that is followed by acceptable reputation. When the reputation is less than acceptable, then an investment is made to rebuild the reputation. This cycle continues. In this dissertation, it is argued that the same deception is possible from a cloud-service provider.
Cyclic pattern of benevolent and fraudulent behavior [29]	Malicious peers on a distributed network follow a cyclic pattern of good and bad transactions when chosen as a service provider. The authors addressed this behavior as a camouflage. The authors suggested punishing such behavior. Three types of behavior are mentioned: an oscillating pattern, an increasing and decreasing pattern, and a random-behavior pattern. In this dissertation, it is argued that the same deception is possible from a cloud-service provider.

There is research that demonstrates the cloud-service provider's deception. The examples are conceptual models described in [14][26][27][33][34][35][36][37][38][39]. Sources [36] and [40] describe deception without explicitly modeling it.

2.4.2. Service-Level Agreement

A service-level agreement, or SLA, is a contract between an end user (a consumer) and a service provider. The service provider describes the details of the promised service in the SLA. The SLA is a formal way of representing the consumer's expected service. Table 2 presents the SLA definition from different authors.

Table 2. Service-Level Agreement Definitions.

Definition by	Definition
[43]	"A service level agreement is a document which defines the relationship between two parties: the service provider and the recipient. This is clearly an extremely important item of documentation for both parties. If used properly, it should • Identify and define the customer's needs • Provide a framework for understanding • Simplify complex issues • Reduce areas of conflict • Encourage dialog in the event of disputes • eliminate unrealistic expectations" (p. 517).
[44]	"A typical Service Level Agreement (SLA) within the Cloud Service Agreements (CSA) describes levels of service using various attributes such as availability, serviceability or performance. The SLA specifies thresholds and financial penalties associated with violations of these thresholds"(p. 5).
[45]	"An SLA is a document that includes a description of the agreed service, service-level parameters, guarantees, and actions and remedies for all violations" (p. 606).

When a consumer receives the expected service is measured by matching the received service's attributes with the information given in the service's SLA. The SLA negotiation is the process of bargaining for the service cost and the service quality between the cloud-service provider and the consumer.

In cloud-computing architecture, the SLA plays a vital role because the provided service's characteristics are mapped into the SLA to find the discrepancies between the promised and the given service characteristics [41][42]. Maintaining the SLA's described service is a challenge for the cloud-service providers due to the heterogeneous architecture. A service-quality fluctuation can happen at each transaction or interaction between the consumer and the cloud service due to load fluctuations, resulting in an SLA violation [42].

When the consumers' service falls below the SLA-described characteristics, then it is considered a violation and is called an SLA violation. In other words, if the consumers do not receive their expected service, then an SLA violation happens. An SLA violation represents two terms in this work: SLA violation magnitude and SLA violation repetition (frequency) over time.

2.4.3. Quality of Service

Quality of service (QoS) is a measure of the service performance in a service-oriented architecture. The QoS measurement is taken at the consumer's end rather than the service provider's end. The QoS measurement is essential for service-oriented architectures, e.g., for the cloud-computing paradigm [46].

The service's expected performance is described in the SLA, resulting in the consumer expecting the QoS to be a part of the SLA. The SLA can be addressed as an "agreement on quality of service" [47, p. 12]. The number of resources should be adopted dynamically in service-oriented architectures to meet the QoS described in the SLA. The adoption includes changing the system's architecture to facilitate the service. This change often includes allocating additional resources.

However, it is a challenge to facilitate additional resources and the resulting architectural change for a service-oriented architecture, e.g., cloud service. It is not possible to overcome this challenge all the time. Failing to overcome the challenge results in a resource shortage or a lower QoS compared to the promised QoS. The work in this dissertation is about the frequent appearance of a lower QoS for the consumer. This repetitive occurrence is a cycle. These cycles should be detected; the detection saves the consumer from the loss incurred by a lower QoS. Table 3 presents the QoS definition from different authors.

Table 3. Quality of Service Definitions.

Definition Summary	Definition
QoS is the degree of satisfaction [48].	"The collective effect of service performance which determine the degree of satisfaction of a user of the service." (p. 3)
QoS is a combined aspect of multiple performance attributes [48].	The QoS is the collective aspect of service-support performance, service-operability performance, serve-ability performance, service-security performance, and other factors.
QoS is comprised of all aspects of a telecommunication connection [49]	"In telecommunication systems there are several measures to characterize the service provided. The most extensive measure is Quality-of-Service (QoS), comprising all aspects of a connection as voice quality, delay, loss, reliability etc." (p. 109)
IBM defined QoS [50].	"QoS covers a whole range of techniques that match the needs of service requestors with those of the service provider's based on the network resources available. By QoS, we refer to non-functional properties of Web services such as performance, reliability, availability, and security" (p. 1).

2.4.4. Trust Model: Definition and Applications

In 1994, Marsh first introduced the concept of computational trust in an artificial-agent environment and defined trust as follows [51]: "Trust in an artificial agent is a means of providing an additional tool for the consideration of other agent and the environment in which it exists" (p. i).

According to Marsh [51], trust is

- A means for understanding and adapting to the environment's complexity
- A means of providing added robustness to independent agents
- A useful judgment in light of experiencing others' behavior
- Applicable to inanimate others.

Marsh formally defined trust to embed the concept of social trust into the artificial-agent environment by using mathematics. According to Marsh, trust is a computational concept from the artificial agent's perspective and is the tool one agent can use to assess another agent [51].

The trust is defined as a continuous variable over the range [-1, +1]. To globalize the same meaning for the trust value for all artificial agents in a system, trust is stratified into qualitative values: blind trust, very high trust, high trust, … very high distrust, and complete distrust. Each qualitative value is assigned to a trust-value range. Marsh also described the trust usage for another agent's anomalous behavior detection.

The contribution of this dissertation is applying the formal trust concept to evaluate a cloud service. The trust is defined in [52], for the cloud services, as follows:

> "Trust in a cloud service means that one believes in and is willing to depend on the cloud service provider and the cloud service IT artifact as well as the platform provider and the platform service IT artifact" (p. 70).

Trust is defined for the service-oriented architecture by an earlier research work [53] as follows: "In Service-oriented network environments, we define Trust as the belief that the Trusting Agent has in the Trusted Agent's willingness and capability to deliver a quality of service in a given context and in a given Timeslot" (p. 7). Table 4 presents some other definitions of computational trust.

Table 4. Trust Definitions.

Definition Reference	Definition
[54]	"Trust: a subjective expectation an agent has about another's future behavior based on the history of their encounters" (p. 1).
[55]	"Trust is indeed an expectation. It is a probability that things will work and keep working as they are supposed to. But when it comes to security, trust should be zero or one, i.e., one trusts an information system, network, etc. or does not" (p. 3740-3741).
[56]	"Trust is a subjective assessment of another's influence in terms of the extent of one's perceptions about the quality and significance of another's impact over one's outcomes in a given situation, such that one's expectation of, openness to, and inclination toward such influence provide a sense of control over the potential outcomes of the situation" (p. 33).
[57]	"Trust is a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (or independently or his capacity ever to be able to monitor it) and in a context in which it affects his own action" (p. 216).
[58]	"Reputation Trust: Trust is the subjective probability by which an individual, A, expects that another individual, B, performs a given action on which its welfare depends" (p. 94).
[58]	"Decision Trust: Trust is the extent to which a given party is willing to depend on something or somebody in a given situation with a feeling of relative security, even though negative consequences are possible" (p. 95).

Cloud computing architecture is an example of service-oriented architecture. Hence, the definition of trust for the service-oriented architecture is also applicable for the cloud domain. The trust characteristics are presented in [55]. Table 5 presents the characteristic summary that is given in in [55].

Table 5. Trust Characteristics' Summary [55, p. 3741].

Characteristic	Example
"Trust can be measured"	"Entity A trusts more in entity B than A trusts in entity C."
"Trust is context-aware"	"Entity A may trust B to perform URL filtering but does not trust B to perform authentication tasks."
"Trust changes with time"	"The amount A trust B may grow or reduce as interactions take place. "
"Trust is directional"	"Entity A may trust B, but B may not trust A."
"Trust is social-aware"	"Entity A may trust entity C because C was introduced by B to A, and A already trusts B."

Computational trust has the ability to represent a historical interaction pattern between two agents using a single parameter (trust value). Hence, this computational trust is an interest of many researchers. This curiosity led to numerous applications of computational trust in the area of computing. Table 6 represents some state-of-the-art computational trust applications.

Table 6. Trust Applications.

Application Summary	Trust Application
Trust and information security [55]	The relationship between trust and information security is presented and is evaluated. The trust measure is presented as an essential element of information-security measurement.
Introducing the con-man attack and its counter-trust model [59]	A decentralized trust model is proposed; it is exploitation resistant against two types of attacks: con-man attack and witness-based collusion attacks. An agent's trustworthiness is calculated by evaluating information from multiple sources. The vulnerability of three trust models to the con-man attack is analyzed.
Trust identifying malicious peer behavior [60]	A trust model is developed to identify malicious peer behavior in peer to peer computing. Trust is calculated by an weighted aggregation of multi-dimensional trust factors e.g., interaction, rating, availability, etc.
Trust-function selection for reputation based on the trust model [61]	The trust model has a trust function to evaluate trust. A reputation-based trust model can use multiple trust functions. A trust-function selection scheme is presented and recommends a trust function for a particular application. This selection is based on the function's classification: subjective or objective, transaction based or opinion based, rank based versus threshold based, etc.
Trust application in cyber-attack scenarios [62]	Trust is used to evaluate different techniques of cyber-attack detection. Example techniques are the big-data and decision-making techniques (information fusion and dynamic data-driven application systems). Such trust can be evaluated using a framework that utilizes Bayesian and Dempster-Shafer theory.
Attack-resistant trust metric [63]	In cryptographic trust implementation, a trust graph is considered. Each node in this graph is a public key with the edges as certificates. When there is an edge from one node to the other, it means the second node is trustworthy. An attack can take place where an attacker can make an edge from this second node to his node. The owned node corresponds to a fraud-computing resource. The link from such a node is a fraud certification referring to an insecure registration. A trust metric is proposed to resist this type of attack. The attack's resistance is measured by counting such compromised edge numbers. If the edge numbers are below a threshold or minimum, then the system is attack resistant.

As described earlier, the trust concept is applicable in an agent-based system. This trust value helps with an agent's decision making about another agent or about the environment and situation where it is. Also, the utility-computing environment is represented using the agent-based modeling in many studies. Hence, the computational trust application in the utility-computing domain, e.g., the cloud-computing domain, is prevalent. Table 7 presents an application for computational trust in this cloud-computing domain.

Table 7. Trust-Model Applications in the Cloud-Computing Domain.

Application Summary	Trust Application in the Cloud
Trust value evaluating the reliability of file exchange in the public cloud [64]	The trust value for a storage resource is calculated from the resource's performance during storage operations. Resources with a high direct-dynamic trust value are highly reliable for file-exchange operations in public clouds. During the calculation, a high score was given to larger storage space, higher processing capacity, and non-varying node capacity over the time, etc.
Trust parameters from SLA [65]	The cloud-service provider's trustworthiness is evaluated, and the set of parameters measuring this trustworthiness is identified and quantified. These parameters are selected and extracted from the SLA.
Trust by comparing the SLA and the consumer's experience [66]	The proposed trust model calculates the trust by comparing the SLA's specified values and the user's firsthand experience with the cloud service. Hence, this trust value scores the cloud service's reliability. It is also proposed that an automated agent monitors this trust and that the cloud-service provider repays the consumer's financial loss from bad service. The consumer can select a cloud service by focusing on the service's trust value.
Trust-based IAAS scheduling [67]	A trust model for cloud infrastructure as a service is proposed. A scheduling algorithm considers this trust value. This algorithm enhances a cloud service's QoS.
Trust-based cloud-task scheduling [68]	Tasks are submitted to the cloud resources by a cloud consumer. The tasks are scheduled into these resources to efficiently execute them. Trust-based scheduling reduces the scheduling-execution time. This trust is calculated by fusing three trusts: one indirect trust and two direct trusts. The indirect trust is reputation based, and the direct trusts are data trust and communication trust. The Bayesian model fuses the direct trusts. Data trust is used to select cloud resources. Communication trust is calculated from the client's resources, e.g., bandwidth.
Trustworthiness of IAAS providers [69]	The cloud-service providers can use a reputation-based trust model to evaluate the trustworthiness of an IAAS provider. This trust is calculated by monitoring the SLA compliance, the service provider's rating, and the service provider's behavior. Service providers give opinions about the IAAS services. Also, the uncertainty of this opinion is considered. An uncertainty model is presented to help the reputation-based trust model.
Trust-model scoring for the cloud-service provider in the marketplace [70]	The trust model used to differentiate the cloud-service quality is proposed. The trust score for a cloud-service provider in this model depends on the selected sources on which trust is calculated and the service provider's filled-out questionnaire about its capability for different aspects, e.g., compliance, information security, and governance.

In this era of computational intelligence, things are becoming more automated. However, there are still many human users for these systems. Humans make decisions that vary from person to person. They want to use a system that they trust. This trust develops over time and mostly depends on past behaviors that are in the form of interactions. As trust develops, humans expect that the system will continue its past behavior [62]. This expectation implies that, if trust is degraded for some system behavior that can be called as a lack of cooperation for the system, then, most likely, the person does not want to have any more interactions with that system. The expectation also implies that, if trust is gained, people want to continue interacting with the system when needed. Hence, past behaviors are important to determine a system's future use for systems that involve humans. To benefit from this human-interaction pattern with the system, it is important to have an ability to measure the trust [55]. This measurement needs to be automated due to the volume of users who are interacting with the system. The work presented here is about the trust evaluation of services, focusing on the service attribute's repetitive nature.

How often should this evaluation happen? For a dynamic system, e.g., a cloud-computing environment, the system's state changes over time; i.e., the system state is dependent on time. During each interaction between the service and its consumer, the system can behave differently. Hence, trust needs to be measured at each interaction. The trust model should be a dynamic trust model, i.e., a trust model that has time dependency [71]. This dynamic-trust evaluation should happen at each discrete time stamp and be separated by a fixed time interval; i.e., time slots between two consecutive interactions of the consumer and the service are not asynchronous [71].

In this dissertation, an existing dynamic-trust model is applied, analyzed, revised, and extended with the goal of fraud- or deception-cycle identification. This trust model observes each interaction, over discrete times and separated by a fixed time interval, between two agents. An agent's action is identified as cooperative when it meets the commitment between two agents [72]. The two agents are the cloud-service consumer and the cloud-service provider. The commitment is the cloud-service provider's promised service. Each observation updates the trust value. The trust value determines the cloud-service provider’s fraud or deception. This application, analysis, revision, and extension, which aims for fraud or deception detection, is the contribution of this work.

2.4.5. Trust and Security

Trust can be defined as a security concept as follows [73]: “Trust is a kind of soft security which complements the traditional hard security like encryption, authorization, and authentication. An agent exists in complex heterogonous environment must possess both two securities in order to be safe and effective” (p. 18).

On the other hand, trust and security are differentiated in [53] as follows:

> “Trust and security are not the same thing in the world of e-Commerce. Unfortunately a variety of uses, particularly of the term ‘trust’, could lead to some confusion. In this section, we clearly distinguish trust and security and when they could be synonymous and when they are not. Security focuses on protecting users and businesses from anonymous intrusions, attacks, vulnerabilities etc., while Trust helps build consumer confidence and a stable environment for customers or businesses to carry out interactions and transactions with a reduction in the risk associated with doing these in a virtual world, thus allowing one to more fully reap the possible rewards of the increased connectivity, information richness and flexibility” (p. 4).

Trust is a tool to verify the expected versus the actual behavior of a cybersecurity concept [55]. The security measures and the measurement limits should be transparent to the consumer. The transparency illustrates the acceptability of these measures and the limits for people. Trust gives such transparency [55]. When trust is applied with cybersecurity concepts, its suggested value is binary (0 or 1) [55]. However, this suggested trust value is a perfectionist’s perception because it is not easy to gain a trust of 1 or 100%. A pictorial relationship for trust, information assurance, information security, and cybersecurity is presented in [55] and as Figure 3 (p. 3734).

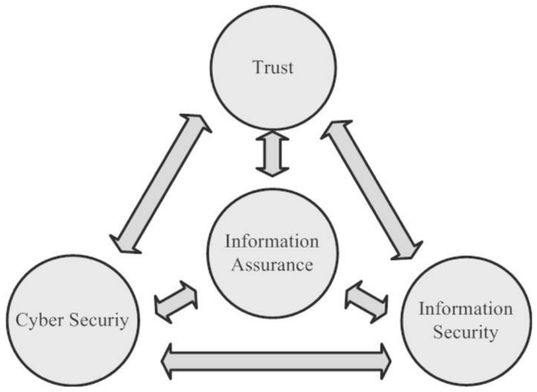


Figure 3. Relationship Among Trust, Cybersecurity, and Information Security.

The figure explains that, when cybersecurity, information assurance, and information security are maintained adequately, then trust can be achieved. Similarly, when trust is achieved, the cybersecurity, information security, or information assurance is partially confirmed. The dependency is partial because these components have other dependencies; e.g., information assurance is also dependent upon cybersecurity and information security.

Trust is included in cybersecurity with the goal of the security improvements. The following subsection describes examples of trust integration with a cybersecurity application.

2.4.5.1. Example: Trust in Information-Security Architecture

Trust concepts can be integrated with the information-security architecture to ensure cybersecurity [55]. The trust needs to be confirmed for each component of this architecture. The review and architecture presented in [55] show the significance of computational trust in cybersecurity, either by integrating the trust with information security or considering the trust as confidence for a security measure.

2.4.5.2. Example: Trust to Ensure Hardware-Level Security

Trust is used to ensure hardware-level security. Secured cryptographic operations are executed by dedicated hardware called a crypto-processor. A secure crypto-processor is hardware in the microprocessor that ensures cryptographic operation at the hardware level. This chip ensures physical security. The Trusted Platform Module (TPM) is an international standard for this secure crypto-processor. The TPM is used to attest a computer hardware platform's integrity. TPM's goal is to propose a safe computing environment (introduced by the Trusted Computing Group). The TPM establishes a chain of trust between the hardware components. The TPM can calculate this trust both locally and remotely.

2.4.5.3. Example: Zero-Trust Model

The incremental use of trust in cybersecurity requires a standard. The National Institute of Standards and Technology (NIST) presented a guideline to establish trust for information security [74]. The guideline is called the zero-trust model of information security and requires all packets in the network (both the internal and external network) to be considered as untrusted. All network traffic is a threat. All the traffic should be authorized, inspected, and secured. Strict access control with the least possible privilege is recommended by NIST. In this trust model, the trust is replaced by "verification." Rather than trusting a user's action, the action is verified. The zero-trust model changes the proverb "trust but verify" into "never trust but always verify" by requiring the inspection and logging of all traffic (both internal and external). The drawback of this guideline is that it is ideal but may not practical to implement due to an organization's cost constraints. For example, building a zero-trust network architecture involves an investment in the infrastructure as well as in the research and development to optimally use the network.

2.4.5.4. Example: Trust Measures Security

Trust can be a measure of the security [75]. Trust can be used to predict the security threats [66]. A security-aware cloud is a cloud platform that uses a security-related trust model. This trust model has an internal trust component. This component calculates trust depending on whether standards for personal-security measures are maintained, e.g., ID management or key management. This security-aware trust model for the cloud has a second trust component called contracted trust. The contracted trust first delegates security for the information by including the security in the SLAs. The contracted trust is calculated by the service provider's ability to offer security as a service.

2.4.5.5. Example: Trust Assesses Security Risk

Trust can assess the security risks [76]. A game-theory model can optimize trust between the consumer and the service provider in cyberspace. Hence, the participants of a three-player game are the consumers, the service providers, and the attackers. This game imposes the consumer's data-protection concern on the service provider.

2.4.6. Trust and Reputation

A system with a mostly positive reputation may not be trustworthy. The trust-value calculation counts the interactions as time-series data while the reputation does not count the ratings as time-series data. Alternatively, the trust-value calculation may have extra components which contribute to trust increments.

A system with a mostly bad reputation has a high chance of becoming untrustworthy. A series of bad-reputation events eventually take the trust score to a lower value. A trust and reputation comparison can be found in [77]. The widely accepted definition of trust and reputation is adopted in [77] as follows:

"Trust: a subjective expectation an agent has about another's future behavior based on the history of their encounters.

Reputation: a perception that an agent creates through past actions about its intentions and norms" (p. 29).

Reputation is an antecedent to trust [78]. Hence, it is implied that trust can be derived from the reputation. Regardless of the fact that trust can be derived from the reputation, the similarities and differences between these two concepts are evident. The following subsections describe the similarities and differences.

2.4.6.1. Similarity Between Trust and Reputation

2.4.6.1.1. Trust and reputation change after each transaction

The similarity between trust and reputation is defined in [78] as follows: "Trust and reputation scores must increase after the agent under evaluation fulfills an obligation and must decrease after the agent violates an obligation. The weights ascribed to either the fulfillment or the violation of an obligation may be different" (p. 3).

2.4.6.1.2. Trust and reputation both fulfill accountability and represent the QoS fulfillment

Trust and reputation can represent the degree of QoS fulfillment [77]. The trust and reputation, in terms of measuring QoS fulfillment, are an indirect measure of accountability [77]. The similarity between these two concepts is this accountability. When the SLA is violated by the service provider in a service-oriented architecture, it results in lower trust values and a lower reputation rating. Monitoring the SLA violations can manage the accountability. The trust-and-reputation engine can match the actual versus expected service quality by comparing the SLA values and real-time service quantities. These engines can give trust values and reputation ratings after matching or comparing the actual versus the expected values or quantities. These trust values and reputation ratings are the service provider's feedback. The provider can take these values seriously and can make a decision about investing in the cloud resources so that the SLA-described quality values are maintained. This way, the trust values and reputation ratings can work as a degree of QoS fulfillment and, hence, can put pressure on the service providers.

2.4.6.1.3. Trust and reputation both focus on relationships

Trust and reputation both focus on a specific relationship between a service provider and its consumer [79]. An example of the relationship is how well the provided service's encryption or authentication is maintained.

2.4.6.2. Discrepancies Between Trust and Reputation

The trust is a state of confidence: the higher the trust, the higher the confidence and vice versa. However, reputation is a score based on an entity's past behavior for a certain attribute or goal. The trust and reputation differences from various authors are presented in the following sections.

2.4.6.2.1. Trust versus reputation: uncertain versus well informed, subjective versus objective

The trust is uncertain and subjective. The authors in [79] claimed that reputation has the opposite characteristics: knowledge of an entity's past behaviors (knowledge as information or observations) and an objective concept.

2.4.6.1.2. Reputation derives trust, but trust does not derive reputation

A suitable metric can calculate the trust, aggregating the reputation ratings of an entity's attributes [78][80]. Hence, the reputation ratings can measure the trust, and the reputation affects the trust. For example, in a mobile, ad-hoc network, a node's reputation ratings from all other nodes can be used to determine the trustworthiness of that single node. However, the same reputation cannot be derived from the calculated trust.

2.4.6.1.3. Trust is between two agents; reputation is all other interacting agents giving a rating

The difference between trust and reputation is given as a definition in [78]:

"The trust that an agent has on another agent under evaluation is given by a score computed using the past contracts established between both agents. In turn, the reputation of the agent under evaluation is computed using all past contracts established with this agent in the Electronic Institution" (p. 3).

2.4.6.1.4. Trust calculation can avoid direct interaction by counting on reputation

The difference between trust and reputation is defined in [73] as follows:

"Trust toward specific agent is generated through recognition and experience under repeated transactions with that agent. Reputation is the socialized trust which can be propagated through a social network of agents. It helps agents trust the target agent without any direct interaction with the target agent" (p. 18).

2.4.7. Game Theory

Game theory studies the interaction between two agents or objects by modeling their interaction mathematically. The agents are capable of making their own decisions. Game theory is defined in [81] as follows: "Game theory is the study of the ways in which interacting choices of economic agents produce outcomes with respect to the preferences (or utilities) of those agents, where the outcomes in question might have been intended by none of the agents" (p. 1).

There are different types of games in game theory. The "cooperative game" and "non-cooperative game" is a type of game, described in [82, p. 322]. This type of game is the interest of this dissertation. In a “cooperative game” and “non-cooperative game”, both agents evaluate each other’s action or interaction. One agent evaluates the other one by considering the interaction either as a cooperation or a defection [57][81][83]. Cooperation of the other agent, from the first agent’s perspective, means that the other agent follows the agreement between these two agents. A defection is when the other agent does not follow the agreement. Cooperation is defined in [81] as “keeping the agreement” (p. 1), and “breaking the agreement” (p. 1) is defection. There is an agreement between the consumer and the service provider in a service-oriented architecture; the agreement describes the consumer’s expected service performance. It is phenomenal that there are situations when the service provider fails to deliver the expected service. This failure breaks the agreement. Hence, for a service-oriented architecture, e.g., a cloud-computing environment, “breaking the agreement,” or failing to provide the expected service to the consumer, is the defection. Similarly, “keeping the agreement,” or providing the expected performance to the consumer, is cooperation. Hence, a cycle of expected or good service along with unacceptable or bad service is essentially the cycle of the game theory’s cooperation and defection concepts. Hence, the definition of game theory presented in [81] relates the game theory’s concepts in terms of cooperation with and defection from the problem specified in Section 2.2.

Game theory has been applied in the cloud-computing domain. For example, game theory is applied in “admission control” (p. 318) of the cloud-service requests [82]. The “admission control” means controlling the acceptance of any new service request while serving the current requests. The SAAS provider and the requests are considered to be players; SAAS is one player, and requests are multiple players. Therefore, for the lifetime of the SAAS service evaluation, the service is an n+1 non-cooperative game, a two-player game for a single SAAS service request.

2.4.8. Con-Man Attack

Although the concept of the con man is not new, its introduction in the utility-computing domain is described in [59]. The counter-trust models that deal with this type of attack are described in [59]. This attack and its counter-trust model subsequently appeared in [1], [2], and [3]. Again, the work presented in this dissertation is about this con-man attack and its resistance [4][5][6][7].

A con man is a person or agent who uses a confidence trick on another person or agent to gain something, e.g., financial gain [1][2][3][59][84]. The con man and the other person, or agent, interact over time, possibly in multiple ways. The con man's behavior is expected to be cooperative and trustworthy. An unexpected behavior is a defection and causes a loss of some type, e.g., financial, for the other agent. A con man gains the other agent's trust by showing cooperative behavior for a certain number of interactions. When the con man determines that it has gained the other person or agent's trust, it proceeds to seek unwarranted gains by manifesting defection behaviors. If the victim agent observes the defections and notices the losses, the agent's trust for the deceiver may be diminished or lost altogether. However, the con man may seek to regain trust by exhibiting cooperative behaviors in subsequent interactions. Once a con man regains trust, defection behaviors may be used. Thus, over an extended time, there can be patterns where cooperation is the norm for periods of time, followed by defections for other periods of time. The victim agent may never realize that it has become a victim of the con man. This scenario is referred to as a con-man attack.

The con-man behavior can be detected using the con-man-resistant trust algorithm where the trust value measures the con behavior [1][2][3][59][85]. This con-man-resistant trust algorithm has been successfully applied to application domains, such as the smart electrical grid, to detect unstable electrical nodes [1]. This chapter is inspired from applying the con-man-resistant algorithm in the smart-grid domain.

The con-man trust model implemented here is SLA based. This trust model is objective-trust based, transaction based, localized information, and threshold-based trust, following the classification of [61].

The con-man-resistant trust algorithm increments and diminishes the trust value based on cooperative and defective behavior. In the previous con-man-resistant trust algorithm research, the trust values were as follows [1]: trustworthy (Trust is close to 1.), not yet known, and untrustworthy (Trust is close to -1.). In this dissertation, trust values are modeled on a continuous closed interval [-1,+1], where +1 indicates fully trustworthy and -1 is fully untrustworthy. Values between +1 and -1 are neither trustworthy nor untrustworthy. Concerning the future, positive trust values between 0 and +1 suggest that the target is trusting the deceiver while negative values between -1 and 0 suggest an untrustworthy view. As interactions proceed, trust increments and diminishments are denoted by α and β, respectively. The values of α and β are adjusted dynamically as a function of history. Basically, increment β increases when deception occurs persistently, and α increases when the cooperation is persistent. Initial values, α_0 and β_0, are set in accordance with the approach presented in [2] and [3].

2.4.9. Con-Man-Resistant Trust Algorithm's Trust-Updating Scheme

If the con-man-resistant trust algorithm is applied to a utility computing domain, then each interaction between the victim and the con man is a time when the consumer receives a service from the provider. Each time a consumer encounters a service, the consumer's trust for the service is updated, depending on the consumer's satisfaction. According to game theory, when the encountered service is good (from the consumer's perspective), it is called cooperation, and when the service is not good, it is called a defection. The evolution of this trust follows the con-man-resistant trust algorithm's formulas called a "trust updating scheme." According to this scheme, which is presented in [1], [2], and [3], "T" represents the current interaction's trust, and "T′" represents the updated trust value when a defection or cooperation happens.

Table 8 and Table 9 show the trust-updating schemes from earlier con-man-resistant trust algorithm research [1][2][3].

Table 8. Trust-Value Dependencies.

T	Cooperation	Defection
> 0	$T' = T + \alpha(1 - T)$	$T' = \dfrac{T + \beta}{1 - min\,(\lvert T\rvert, \lvert\beta\rvert)}$
< 0	$T' = \dfrac{T + \alpha}{1 - min\,(\lvert T\rvert, \lvert\alpha\rvert)}$	$T' = T + \beta(1 - T)$
= 0	$T' = \alpha$	$T' = \beta$

Table 9. Trust-Parameter Dependencies.

Cooperation	Defection
$\alpha' = \min\,(\alpha + \gamma_C\,(\alpha_0 - \alpha),\ \alpha_0)$	$\alpha' = \alpha \times (1 - \lvert\beta\rvert)$
	$\beta' = \beta - \gamma_D \times (1 + \beta)$
$\gamma'_C = 1 - \lvert\beta\rvert$	$\gamma'_D = C \times \lvert T\rvert$

It is clear from the equations in the both tables that trust parameters are dependent on the α, β, and C values. α is the reward parameter; β is the punishment parameter; and C is a constant value. Both the reward parameter, α, and punishment parameter, β, have initial values: α_0 and β_0, respectively. C (= e^{-1}), α_0, and β_0 were recommended by earlier works [1][2][3]. These recommended values were used in a smart electrical-grid application [3].

When defection happens, the reward diminishes (α decreases), and punishment increases (β decreases or |β| increases). These increments (Δα) and diminishments (Δβ) are dependent on C (Table 9) along with their previous values.

Now, trust is incremented if the consumer encounters good service or cooperation. The updated trust is the current trust, T′. If the previous trust value (T) was higher, then this trust increment shall be more than a lower previous trust value. Also, there should be a reward amount that results from good service or cooperation. Hence, the size of trust increase depends on a positive reward value (α) and the previous trust value (T). This scenario is represented by Table 8’s second column. Again, the better service or cooperation that there is, the higher the reward value, α, is. Similarly, with worse service or defections, the smaller the reward value, α, is. This relationship between the service quality and the reward value explains the Δα. This relationship is represented by the second row of Table 9’s first column and the second column’s second row. In the second column’s second row, the reward parameter, α, is multiplied by a number less than 1 ($1 - |\beta| < 1$), resulting in the new α or the current value of α (α′) being smaller than the previous α.

Trust is diminished if the consumer encounters unexpected bad service, or a defection (column three’s second and third rows in Table 8). If the previous trust value (T) was lower, then this trust diminish should be more than a higher previous trust value. There should also be a punishment amount that results from bad service or a defection. Hence, the size of a trust diminish depends on a negative punishment value (β) and the previous trust value (T). This scenario is represented by Table 8’s third column. Again, the worse service or defections there are, the higher the punishment value, β. This relationship between service quality and punishment value explains the Δβ. This relationship is represented by the second column’s third row in Table 9. In the second column's third row, the defection, or punishment parameter, β', is derived by deducting a value from β (previous β value); i.e., $\beta - \gamma_D \times (1 + \beta)$. The β value is always negative, and deducting an amount from β further reduces the new β value, meaning that β' becomes more negative than β. However, this deducted amount is smaller than 1 because $(1 + \beta) < 1$ and $\gamma_D \times (1 + \beta) < (1 + \beta)$.

2.5. Sub Problem 1: Detecting Cyclic Fraud for a Cloud Service with Simulated Data

In this dissertation, the repetition of a cloud-service shortage where the provided cloud service (resource or task execution) is shorter than the promised cloud service is the issue. Third-party auditing tools to monitor the cloud services are mentioned in [19], [86], [87], [88], [89], [90], and [91]. These tools monitor specific aspects of the SLA. If a new SLA violation scheme emerges, the existing tool may not be able to monitor that scheme. The repetitive performance-issue scenario is a con-man attack scenario and a new SLA violation scheme. The violation might be overlooked due to the longer duration between consecutive violations, but this repetition has an accumulated loss on the cloud-service consumer's end. Hence, a specific methodology that identifies the con behavior is necessary. This section's goal is to model this con-man deception in a cloud-computing environment and to implement a counter algorithm in order to detect the deceit.

2.5.1. Motivation

The deception-detection research presented in this dissertation is motivated by a cause-and-effect analysis. On the cause side, there is evidence that providers in a service-oriented architecture engage in deception. The following subsection describes the deception evidence. Each ruse has the ability to create additional deceptions. The modeled and mitigated con-man deception for the cloud services is a derivative of the deception evidence. Hence, this dissertation about con-man-deception resistance is inspired from [1], [2], [3], [19], [26], [27], [28], and [29].

When an SLA violation is repeated in a cloud service for a long time, the consumers cancels the service. The consumers' dissatisfaction is the effect. Repetition of the cloud service's performance shortfall (short-change or under-provision) or a repeated SLA violation incurs a loss on the consumers' end. The cloud-service provider may pay for this loss [92]. The service provider may pay a penalty that results from an SLA violation [93][94]. Even if the service provider agrees to penalize itself for the loss, it is hard to calculate the loss amount, i.e., the monetary effect of under-provisioning [9]. In addition, there are cases where it is not possible to replenish the losses. In this dissertation, the inability to attract new consumers as well as losing the existing consumers due to frequent service-performance shortfalls are the concerning losses. The following subsections present evidence that shows how the repeated and unexpected negative performance upsets the cloud-service consumers. The reduced number of consumers incurs a financial loss for the cloud business. Hence, the con-man-attack deception needs to be predicted and spotted in order to reduce the loss.

2.5.1.1. Deception Evidence: Deception-Monitoring System

Recent research suggests the involvement of an external audit party which is called a third-party auditor (TPA) to monitor the cloud services [86][87][88][89][90][91]. The audit service works as the consumer's delegate to verify the cloud service's integrity or performance, e.g., how secure the service is, how well the SLA is maintained, integrity of the stored data, etc.

There are cloud-service providers which trick the consumer by deceiving the SLA [26][27]. This deception reduces the service provider's investment in the cloud service. The claim is that the cloud-audit party cannot catch many of the cloud-service provider's deceits. The cloud service's consumers do not have knowledge about when a provider lowers their allocated resources or service. Fewer resources mean reducing the service provider's investment or saving costs. At the same time, the cloud-service provider wants to be competitive in the cloud market. Hence, the company tries to hide these short-change issues, e.g., hiding frequent under-provisions, slow-response events, security incidents, etc.

How does a deceptive cloud-service provider hide the issue? The answer is presented in [26] and [27]: the company takes advantage of service-monitoring time gaps and the auditor's limitation to access the cloud service's user data. (Access can be restricted by the service provider by citing a security policy.) The auditor monitors the cloud service by following intervals. The service provider often gives preferential access to the user's service data (many times due to a security policy). Hence, if the service provider delivers short-changed service during this interval period, the time when the auditor is not auditing, then the auditor will not catch this short-changing behavior or the SLA violation. This inability means that no SLA violation happened because there is no evidence about the violation, except in the user's experience or feedback.

Two deceptive scenarios are illustrated in [26] and [27]. A third-party auditor is capable of addressing these two deceptions [26]. The intermediate entities can crosscheck the QoS data with the auditor-informed QoS data [27].

In the first deception scenario, the cloud-service provider tempers the QoS information to deceive the auditor. The provider has both higher- and lower-configuration resources or services with respect to the SLA-described resources or service characteristics. However, the cloud-service provider allocates a lower-configuration resource. When this low-configuration resource or service is chosen, the service provider's cost is reduced, but the consumer suffers. The status of the higher-configuration resource or service is returned when the resource's status is sought by the auditor. The provider continues this process for the service's entire lifecycle. Hence, the undetected SLA violation repeats. This process is a deception. This deception has all the characteristics of a con-man attack.

The con-man-resistant trust algorithm can be at the consumer's side and can monitor the consumer's service. This algorithm's trust-value evolution can evaluate the service's con-man deception. Additionally, the algorithm's consumer-side residence allows the validation of the audit data.

In the second deception scenario, the cloud-service provider predicts the consumer's usage pattern for the service. Then, the provider only upholds the SLA at those times, i.e., only ensuring the promised resources at the predicted times. Therefore, when the consumer changes the service's usage pattern, then the service provider delivers a smaller service or resource amount than promised. This action is a deception because the resources were supposed to be consistent over time. It is guaranteed that the service provider cannot predict the consumer's service-usage pattern with 100% accuracy. At the time stamps where the predicted pattern does not match the real pattern, the SLA violation can be spotted. Over time, this scenario of SLA violations is duplicated. This repetition is part of the evidence about the deception, and the pattern matches the con-man attack. Hence, the con-man attack is a derivative of this service provider's deception with the exception of deceptive cloud-service provider (deceptive CSP) definitions.

Unfortunately, the methodologies offered to detect the deceptions in [26] and [27] are not designed to spot the deception's repetitive nature. The derivative con-man attack or the repetitive

SLA violation reduces the cloud-service provider's investment and deprives the legitimate consumer.

2.5.1.2. Deception Evidence: Shorter Service in a Peer-to-Peer Network

In a peer-to-peer network, one peer or connected computer's CPU cycles can be shared with another peer [19]. The peer that shares resources is bound to provide the promised resource amount. However, there are cases when these resource providers deceive by providing fewer resources than promised [19]. The provider further deceives by billing for the promised resources rather than the served amount. This shortage scenario is a short-change deception [19].

In this dissertation, repeating this short-change deception is identified as the cloud-service provider's cyclic fraud behavior: the con-man attack or deception. This short-change action is the repeated under-provisioning scenario for IAAS and the repeated, unexpected response time for SAAS.

2.5.1.3. Deception Evidence: Oscillating Reputation in a Peer-to-Peer Network

An example of oscillating service-performance behavior in a peer-to-peer network is described in [28]. The oscillating service-performance behavior is necessarily a con-man behavior in the peer-to-peer network. A malicious peer considers the peer-to-peer network's environment as a game and can maximize the peer's profit by deploying a trick that is not detectable by the game's rules. The number of deceptions can be numerous. One example is that peers can cheat by building a high reputation at the beginning. Then, they can start cheating occasionally in such a way that the cheating does not affect their existing reputation very much. This cheating gives them more profit because maintaining good service all the time is investment intensive compared to this cheating model. The oscillating-performance issue of this cheating model is identical to the con man's repeated defection.

Another trick, targeting the profit maximization, is an oscillation of building a reputation and taking advantage of that reputation. By taking advantage, it means to give acceptable service where good service was expected following the good reputation. When the reputation goes below the acceptable level, then an investment is made to rebuild the reputation. This cycle continues. This oscillation is the con-man behavior. A dynamic, reputation-based trust model is applied in [28] to identify this deception. The issue with this trust model is that it needs feedback about the evaluated peer from many network peers. The feedback may not be available all the time, and the feedback numbers vary, adding uncertainty to the evaluation.

2.5.1.4. Deception Evidence: A Camouflaged Cyclic Pattern in a Distributed Network

The trust value is used in [29] to avoid a peer with fraudulent or malicious interactions. One behavior of a malicious peer is following a pattern of good-and-bad transactions when chosen as a service provider. This pattern can have three types: an oscillating pattern, an increasing and decreasing pattern, and a random pattern. The authors suggested punishing all three patterns [29].

The oscillating pattern is a cycle of consecutive good transactions for a period of time followed by consecutive bad transactions for the next period of time. The oscillating behavior is presented in [29] and as Figure 4 (p. 548) to describe the peer’s oscillating, malicious behavior.

The figure’s x-axis represents the amount of the peer’s promised behavior. The scale is from 0 to 1. Here, 0.0 represents that the 0% of the promised behavior is provided by the peer; i.e., the promised behavior is never met by the peer. A 1.0 means that 100% of the promised performance is achieved. The figure’s y-axis represents the transactions between this peer and the other peers. The peer provides 100% of its promised behavior to the other peers for the transaction at time t = 0.

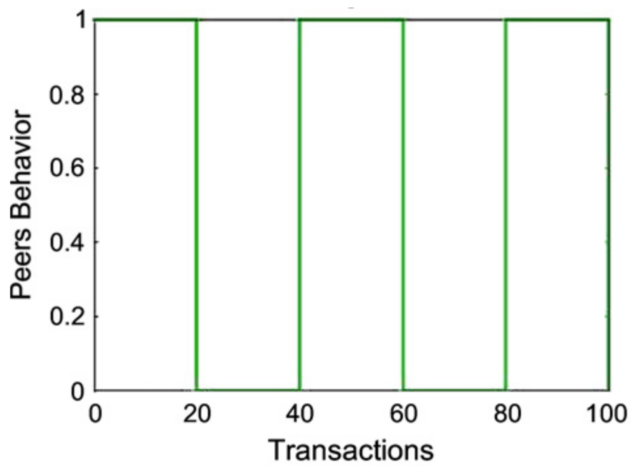


Figure 4. Example of Oscillating Behavior.

However, at t = 20, the peer cannot behave as promised; i.e., it is only able to perform 0% of its promised behavior. Again, at t = 40, the peer is able to come back to its promised behavior of 100% performance. At t = 60, the peer fails to keep its promise and performs only 0% of the expected behavior. This cycle of expected behavior and not-expected behavior continues. The authors of [29] addressed this behavior as a camouflage.

The increasing and decreasing pattern is described in [29] and as Figure 5 (p. 548). Here the peers' transactions (as a behavior) gradually become good and then gradually become bad. This cycle continues.

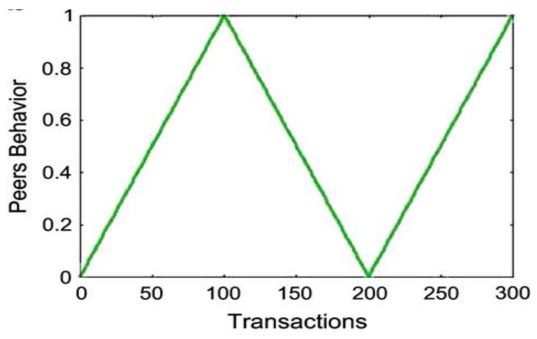


Figure 5. Example of an Increasing and Decreasing Pattern.

The random behavior pattern is described in [29] and as Figure 6 (p. 548). Here, the peers' transactions (as a behavior) become better and worse, but may not follow a uniform cycle.

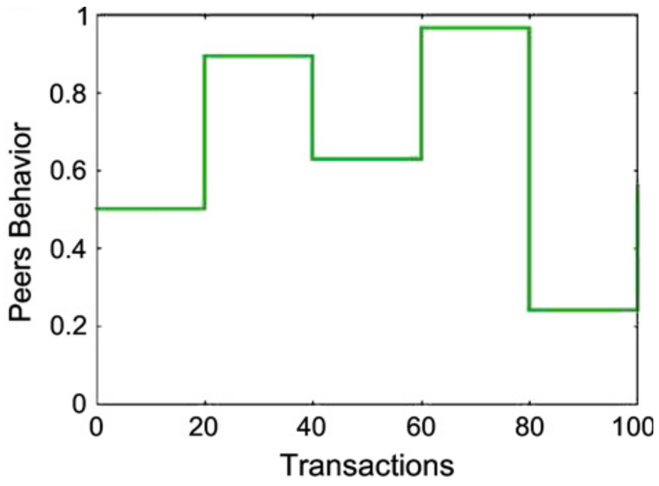


Figure 6. Example of Random Behavior.

2.5.1.5. Evidence of a Repeated Short-Change Action Repelling the Consumer: Softlayer.com

Softlayer.com is a cloud-hosting service which is now owned by IBM (since 2013). In 2011, the webhostingtalk.com forum claimed that this service provider's cloud-storage service had repeated downtime; one consumer reported it as an "unreliable" storage service [8]. The reported downtime instances were first spotted in 2009. The consumer reported that this cloud-service provider did not fix the issue until 2011. Along with these downtimes, other unstable cloud-environment-related issues were reported; these problems repeatedly happened each week. Other consumers shared similar experiences about cloud services that were part of Softlayer.com. Due to the repeated issues of downtime and other instability with the rented cloud services, the Softlayer.com consumers changed service providers.

2.5.1.6. Evidence of a Repeated Short-Change Action Repelling the Consumer: Animato Site

A real-life scenario of repeated, unexpected service or repeated under-provision was presented by [9]. The authors claimed that, when the cloud-service shortfall continued or was frequent, some users left the service site permanently.

When the demanded resource amount surpasses the available resources, a shortfall happens. When the shortfall continues, the existing consumer base dwindles, and the number of new consumers attracted to the service also decreases. This continuous decline for the number of consumers reaches a point where the demand is no longer larger than the available resources (because the customer base is small). At that point, the shortfall or under-provision frequency decreases. This behavior is explained in [9] and as Figure 7 (p. 11). A real-life reference for a social-networking and gaming site (Animoto) that experienced this consumer behavior is presented in [9].

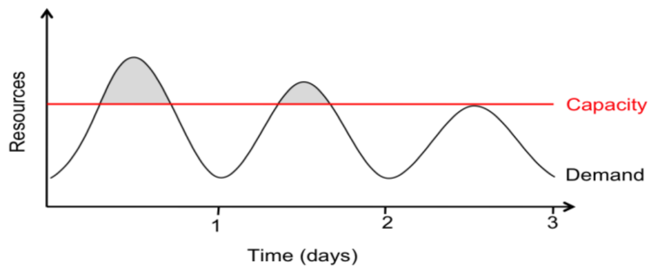


Figure 7. A Frequent Shortfall Reduces the Number of Consumers, Resulting in a Lower Demand.

2.5.1.7. Evidence of a Repeated Short-Change Action Repelling the Consumer: Friendster.com

The social-networking and gaming site Friendster.com failed because the site's consumers gradually left [10]. The site had competitors: Myspace and Facebook. The site's consumers were dissatisfied with its slow response time (reported to be as much as 40 seconds) and switched to its competitors. Friendster.com underestimated the importance of redesigning the existing system to accommodate a future high-consumer load [95]. The company tried to resolve the issue by using its existing design and by adding new components, rather than doing a complete redesign and restructuring. The company also focused on the business side instead of the technical side. As a result, Friendster.com became unreliable (based on the consumers' perspective), and consumers migrated to the competitors' sites. This real-life example proves that, when a service's performance suffers repeatedly, the company gradually loses consumers, leading to the failure of the service provider's business.

2.5.2. Con-Man-Deception Scenario

The deception scenarios of the previous subsection has a cyclical behavior of good and bad services. Hence they are a con-man deception scenario. Additional con-man deception scenarios are presented in this subsection.

The provider of any service-oriented business, e.g., the cloud service, wants to attract consumers by promising and providing good service during the startup period. The SLA specifies the consumer's quality expectations. The anticipated quality includes the consumer's desired values for the quality attributes, e.g., the service's response time or latency, the CPU speed, and various conditions for storage. As time goes by, the number of the consumers can increase. The resource constraints can easily result in diminishing service levels for things such as cloud computing, because the number of the consumer rose. Inadequate service can diminish the service provider's acceptance to the consumer. The cloud-service provider may have few choices. These options may include preventing inequity by expanding resources to prevent shortages, distributing shortages equally among the consumers, or taking no action. Another choice is to distribute the shortages among the consumers in an unequal way, possibly through policies that amount to deception. For example, large data sets which are moved to low-cost tertiary storage may severely limit access times and violate the SLA. The provider may implicitly or explicitly have threshold levels for which the consumer satisfaction is expected to be acceptable or at least not severe enough to cause the consumer to switch service providers. Some choices are investment intensive while others risk the consumer’s dissatisfaction or defection. One such deception mechanism is that the con-man attack is modeled and experimentally analyzed in this research.

In the con-man attack scenario, the cloud-service provider may try to maintain a high confidence by communicating assurances that its service is good even if it is not. Service providers may engage in excellent service for a time and then fall back to a poor level of service. The service provider may cleverly deal with complaints to suggest that the user is not properly handling himself or herself while using the system, thereby shifting the blame to the end user. This blaming scenario is sort of like the old tactic of blaming the victim when something bad happens, basically suggesting that the victims brought the harm upon themselves.

As an example of this con-man scenario, a cloud-service provider with a relatively large number of consumers is considered. The consumer engages in requests for service that demand resources over time. There are situations when the resources requested by the collective set of consumers during a certain time interval is more than the resources available. This resource shortage scenario can happen if the provider is overcommitted or if some resources have failed [9]. When shortages occur, the service provider makes different choices for resource allocation; these decisions are often related to revenue considerations. For example, high-revenue consumers may have their demands met often, and the low-revenue customers may have their needs partially met. Thus, resource shortages drive service inequities. When the service falls short, trust in the service provider may be diminished. This situation may cause the service provider to deliver good service when it is trying to keep a customer who may defect and to provide lesser service when the company is confident that the customer is secure. This good and poor service provision results in patterns, or cycles, of good and poor service, the basic meaning of a cloud-service provider's con-man behavior.

Overall, the cloud-service providers are the deceiver, and the service's consumers are the targets. The cost implications for a victim can be large.

2.5.3. The Deception Modeling and Implementation

2.5.3.1. Con-Man Attack in SAAS

The Cloudsim simulator was used to implement the con-man behavior in cloud computing for software as a service (SAAS). The concept is that the service provider cannot provide the expected service to consumers at certain times. The provider delivers the expected service at other certain times. This cycle simulates the con-man behavior where the ability to provide the expected service is cooperation, and the inability to provide the expected service is a defection.

The SLA concept in a real-life case is presented in [42]. This use case is about how enterprise consumers (the gaming industry) outsource their data using cloud services. The importance of the response time or a service's latency for the gaming industry is presented. For SAAS, the response time (latency of service) is the metric used to measure service in this dissertation.

In the Cloudsim simulator, a web-service call is simulated by submitting tasks to a simulated cloud resource, e.g., into a virtual machine (VM), at regular time intervals. Hence, for each simulation time, there is a task submission. Each task is called a cloudlet. Over time and at each time stamp, each cloudlet’s completion time is set as the response time for the corresponding web service. If the response time is beyond a threshold, then latency happens. This latency is the amount by which the expected QoS is not met.

The dynamic nature of the traffic load is incorporated into the web-service simulation by giving more input cloudlets to the same virtual machine (VM) that is processing the regular cloudlets or tasks. When more tasks appear at the same VM, they overload the VM, and the VM delays the regular tasks, resulting in a latency, or longer response time, for the web service.

Again, certain cloudlets are given high priority (1), and the regular-interval cloudlets are given low priority (0). The lower-priority cloudlets are the consumer’s simulated tasks. The cloudlet or task-scheduling algorithms of Cloudsim are modified (space shared and time shared). The modification schedules the higher-priority tasks or cloudlets first and, then, the low-priority tasks. The space-shared scheduling algorithm simulates the intentional cost minimization or revenue maximization. The higher-priority tasks are given most of the available space in the shared space because their list is selected first. On the other hand, the lower-priority tasks are scheduled in the leftover spaces if there are any leftovers. If there is no space, then at the next scheduling cycle, the lower-priority tasks are scheduled. However, the priority application does not work in the time-shared scheduling although the time-share algorithm scheduled higher-priority items first and the lower priority next. Time sharing also simulates revenue maximization because Cloudsim is not creating additional resources (simulating real-life additional-resource investments).

The unexpected QoS at certain time intervals is simulated by introducing the tasks' or cloudlets' load along with the regular tasks into the VM. Figure 8 describes the simulation scenario.

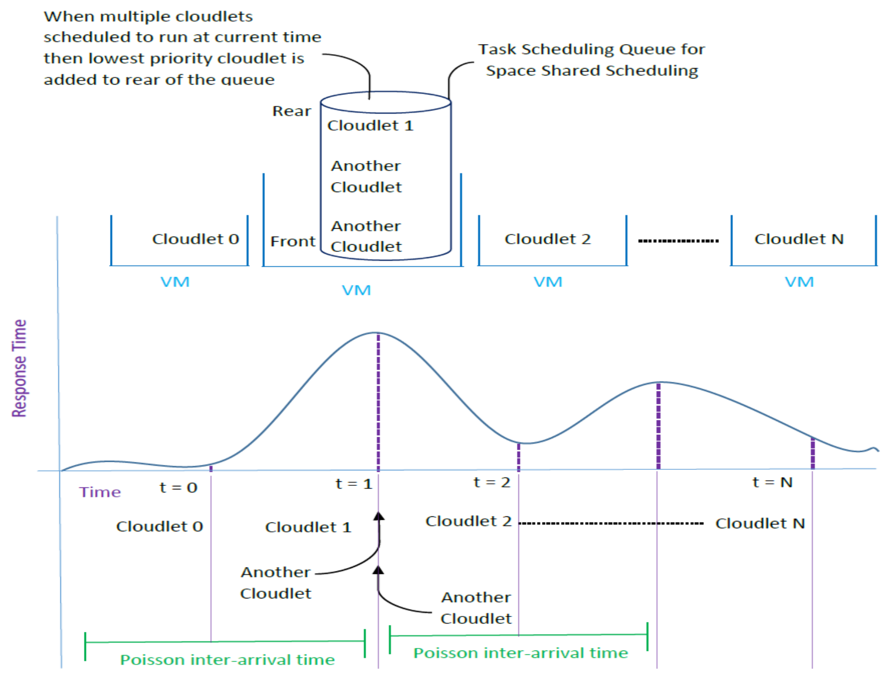

When multiple cloudlets scheduled to run at current time then lowest priority cloudlet is added to rear of the queue
Task Scheduling Queue for Space Shared Scheduling
Rear
Cloudlet 1
Another Cloudlet
Another Cloudlet
Front
Cloudlet 0
Cloudlet 2
Cloudlet N
VM
VM
VM
VM
Response Time
Time
t = 0
t = 1
t = 2
t = N
Cloudlet 0
Cloudlet 1
Cloudlet 2
Cloudlet N
Another Cloudlet
Another Cloudlet
Poisson inter-arrival time
Poisson inter-arrival time

Figure 8. The Web Service as Cloudlets with Normally Distributed Demand and Poisson-Distributed Arrival Time for the Demands.

The load tasks that arrive together follow Poisson inter-arrival times [96][97]. Again, the number of task simulates the load level. Therefore, the number of tasks follows a normal distribution.

In the Cloudsim simulator, submitting individual cloudlets at each time interval and also submitting a collection of cloudlets at a specific time is not straightforward. All the cloudlets are submitted to the VM at the simulation's start time. Each cloudlet has its own submission time. The submission times are set during the simulation's start time when all the cloudlets are submitted.

When the Cloudsim simulator reads the cloudlet information, the simulator acknowledges the cloudlet's start time. For example, if the goal is to model a 4-time-unit, or milliseconds, web-service call that experienced a con-man attack, then first, more than 4 cloudlets (e.g., 5 cloudlets: cloudlet 1, cloudlet 2, … cloudlet 5) are submitted at the simulation's start time. If the simulation times are 0.0, 1.0, 2.0, and 3.0 milliseconds, then the first 4 cloudlets' start is delayed by 0.0, 1.0, 2.0, and 3.0 milliseconds. If the delay is not added, then all 4 cloudlets would start at 0.0 milliseconds, defeating the purpose of web-service call or task submission at a uniform interval. Now, consider that an overload is introduced at 1.0 (time t = 1). Therefore, the 5th cloudlet's start is delayed by 1.0 milliseconds. Cloudsim simulator runs two cloudlets (cloudlets 1 and 5) at time 1.0 millisecond (t = 1.0). Consider that the VM where the cloudlets are submitted only has resources to run one cloudlet. Therefore, if more than one cloudlet is submitted at the same time, the resource is shared between the two cloudlets, resulting in a longer completion time for the tasks associated with the cloudlets; i.e., performance degradation will happen. Hence, at t = 1.0, a resource overload results in performance degradation. Figure 9 shows this scenario where all the tasks, or cloudlets, are submitted when the simulation starts, but the execution is delayed until the desired time.

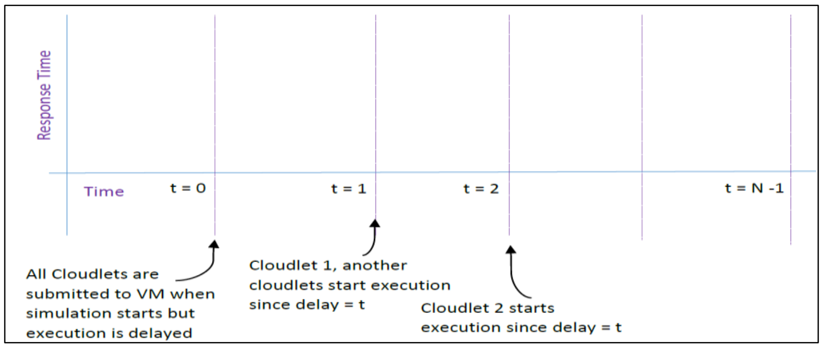

Response Time
Time
t = 0
t = 1
t = 2
t = N -1
All Cloudlets are submitted to VM when simulation starts but execution is delayed
Cloudlet 1, another cloudlets start execution since delay = t
Cloudlet 2 starts execution since delay = t

Figure 9. Task-Submission Simulation in Cloudsim.

Figure 10 shows sample SAAS simulation data. Both the service time (discrete) and the response time are in milliseconds.

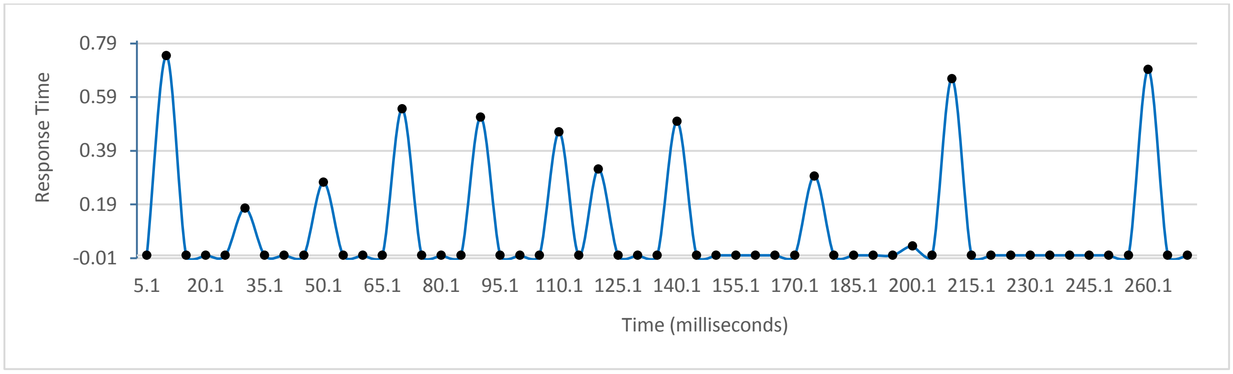


Figure 10. Sample SAAS Simulation Data.

2.5.3.2. Con-Man Attack in IAAS

The Cloudsim simulator is used to implement the con-man behavior in an infrastructure as a service (IAAS). Like SAAS, in IAAS, the expected service is represented using a metric. The metric employed here is millions of instructions per second (MIPS) [22][23][24]. The consumer requests a level of MIPS from the service provider. The provider delivers either the requested MIPS or a lower MIPS. When a lower-level MIPS is provided, the expected QoS falls short. This shortage in QoS is the SLA breach by the service provider.

In the IAAS simulation, the cloud resource's configuration is checked or monitored at each simulation time that is separated by a fixed duration (i.e., at time t = 1.0, 2.0, 3.0, 4.0 …. milliseconds). A cloud consumer is given a certain amount of resources from a virtual machine (VM) as the rented infrastructure. MIPS is an attribute of the VM in the Cloudsim simulator. The cloud-service provider shall give its consumer a VM's MIPS that the customer requests. MIPS means how many instructions the VM can process in a time unit of seconds. If the VM's MIPS is shorter than the consumer's requested amount at any time, then the SLA breach happens. Hence, at each simulation time (e.g., t = 0.0, 1.0, 2.0, 3.0, 4.0, etc.), the requested MIPS versus the allocated MIPS for the VM is monitored. Consumers may need a different MIPS at each monitoring time because their task load is variable. Although the VM shall have a fixed and SLA-mentioned amount of MIPS available all the time, in real life, the amount is not available due to the cloud infrastructure's inherent issues. Hence, at each monitoring time, the consumer's requested MIPS needs to be match the allocated MIPS.

The SLA violation is simulated by injecting it at some t values. Figure 11 shows an example of the IAAS simulation.

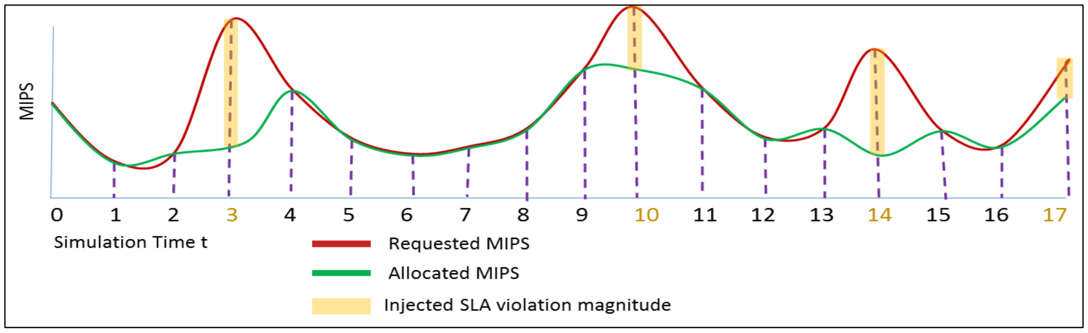


Figure 11. IAAS Simulation.

The x-axis represents the simulation time, and the y-axis represents the MIPSs that are requested and allocated. The time, t, of the SLA violation follows a Poisson distribution. For example, if λ = 5, then one SLA violation happens for every 5 simulation time units.

There are 10 Poisson numbers generated by following λ = 5. The assumed numbers are 3, 5, 4, 2, 3, 1, 4, 5, 3, 1. At the 1st simulation-time duration of 5 time units, an SLA violation will happen at the 3rd time unit. For the 2nd simulation-time duration of 5 time units, another SLA violation will happen at the 5th time unit, etc. The 1st SLA violation happens at t = λ×0+the 1st generated number = 5×0+3 = 3.0 time unit. The 2nd SLA violation happens at t = 5×1+the 2nd generated number = 5×1+5 = 10.0, the 3rd violation at t = 5×2+4 = 14, the 4th violation at t = 5×3+2 = 17.0, etc. Therefore, at t = 3.0, 10.0, 14.0, 17.0, etc., the SLA violation happens. Hence, at t = 3.0, 10.0, 14.0, 17.0, etc., the consumer experiences fewer MIPSs than requested at his or her allocated VM.

The simulated SLA-violation magnitude, or resource-shortfall amount, follows a normal distribution. Consider a normal distribution with mean of 0.5 and a standard deviation of 0.5. The generated series of normal numbers from this distribution is 7.0, 0.3, 0.4, 0.9, 0.5, 0.1, 0.6, 0.8, 1.0, and 0.2. The 1st SLA violation causes an MIPS shortage of 70%. If the consumer's need or request 100 MIPS, then only 30 MIPS are allocated. The 2nd SLA violation allocates 70% of the MIPS requested. Similarly, the 3rd, 4th, 5th, and 6th SLA violations provide 40%, 90%, 50%, and 10% of the VM's requested MIPS. The 1st SLA violation happens at t = 3.0 with 70% of the requested MIPS amount, a 70% SLA violation. The 2nd violation is at t = 10.0 with a 30% SLA violation, the 3rd at t = 14.0 with a 40% SLA violation, the 4th at t = 17.0 with a 90% SLA violation, etc.

At t = 0.0, there is no SLA violation; hence, the MIPS is allocated as requested. Similarly, at t = 1.0 and 2.0, the MIPS is also allocated as requested. However, when t = 3.0 is reached, 30% of the requested MIPS is not provided. At t = 4.0, the exactly requested MIPS is provided. Similarly, at t = 5.0, the allocated MIPS is the exact amount that was requested. the exact requested MIPS provision continues until t = 10.0 is reached. At t = 10.0, the SLA violation is injected by allocating 30% of the requested MIPS. The MIPS request at t = 11.0, 12.0, and 13.0 is also executed normally with the requested MIPS allocated. At t = 14.0, the requested MIPS is shortened, and 40% of the requested MIPS are allocated. The IAAS is simulated in this way, including the SLA violation or under-provisioning.

An interesting question is why the SLA-violation injection follows the Poisson inter-arrival time. The answer is explained in Figure 12. A consumer is allocated with a VM as the consumer's requested infrastructure. The consumer demands a certain configuration from that VM. However, the configuration is not available all the time in real life, although, ideally, it should be what is promised.

The VM's inter-arrival time follows a Poisson distribution [23][98]. There are time intervals when multiple VM-generation requests are executed at the same time as several consumer requests for resources. If the total MIPS or other resources available from the data centers can accommodate the VMs, then no performance degradation happens. If the total MIPS or other resource amount is less than all the VM requests arriving at that time duration, then due to resource management, the existing VMs' resource-shortage happens. (e.g., During migration, some percentage of resource shortage happens.) The arrival of requests is one reason for a resource shortage or SLA violation. For example, at t = 1, multiple VM requests arrive, and a shortage happens (Figure 12).

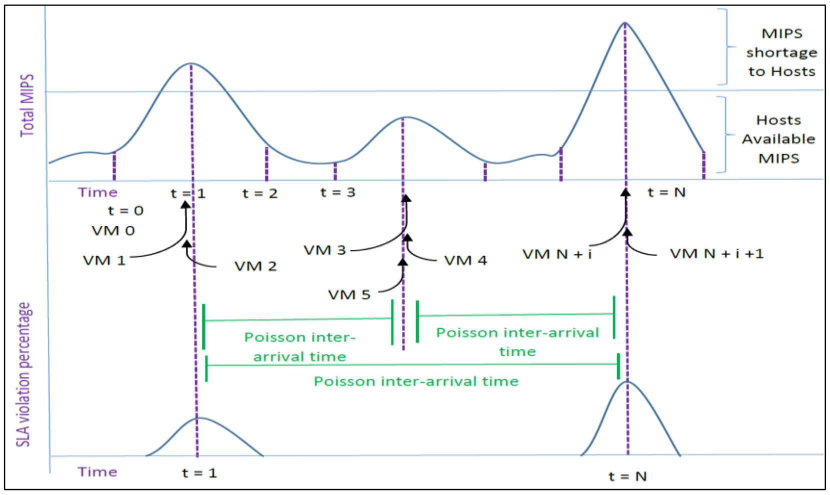


Figure 12. The IAAS' SLA-Violation Issues Follow a Poisson Distribution.

At t = 4, multiple VM requests arrive, but a shortage never happens. Again at t = N, multiple VM requests arrive, and a shortage happens. Hence, the duration between the first (t = 1) and this last resource shortage or SLA violation (t = N) is the summation of two Poisson numbers. The aggregation of Poisson numbers itself is a Poisson number, the resource shortage due to unexpected consumer requests also follows a Poisson distribution.

2.5.3.3. Summary of the Con-Man Attack in SAAS and IAAS

Each reading from the simulator against the simulation time represents QoS data. The QoS data for a specific time are known as individual interactions. Each interaction may or may not violate the SLA. As described previously, each SLA violation has components about when the violation happened and the violation's magnitude. The number of interactions or inter-arrival interactions between two consecutive SLA violations follows a Poisson distribution. The SLA-violation magnitude follows a normal distribution.

An online monitoring system, http://downrightnow.com/, shows the outage history during the past 24 hours for popular cloud-service providers. The outage history is from user-reported downtime as well as official announcements and feeds.

2.5.4. Deception-Detection Implementation

2.5.4.1. Simulator Settings

The precision of decimal places is important in this work. Hence, when data are generated following some statistical distributions (using an R script), 10 decimal places are used. All the data which are read and generated by the con-man-resistant trust algorithm in the simulation with Cloudsim have 10 decimal-place precision. Previous work had 5 decimal points, e.g., α, β values [1][2][3]. Hence, to keep consistency with the earlier research, when results for the con-man-resistant trust algorithm's simulation were retrieved, 5 decimal-point precision was utilized. For additional information, Table 10 lists the simulated cloud-resource configuration used in this work (for the Cloudsim simulator).

Table 10. Simulated Cloud-Resource Configuration.

Datacenter Characteristics	Host Characteristics
System architecture x86	Four Processing Elements for each 100,000 MIPS
Linux operating system	16,384 MB of host memory
Virtual machine manager "Xen"	1,000,000 GB host storage
Location of the resource in time zone 10.0	10,000 units of bandwidth
The cost for processing in this resource is 3.0.	
The cost for memory in this resource is 0.05.	
The cost for storage in this resource is 0.1.	
The cost for bandwidth in this resource is 0.1.	

2.5.4.2. Deception Detection: Defection-and-Cooperation Cycle Detection

Figure 13 shows a cooperation-and-defection cycle. The figure shows the SAAS response times and the trust value in the x-axis, and the pass of discrete time is in the y-axis.

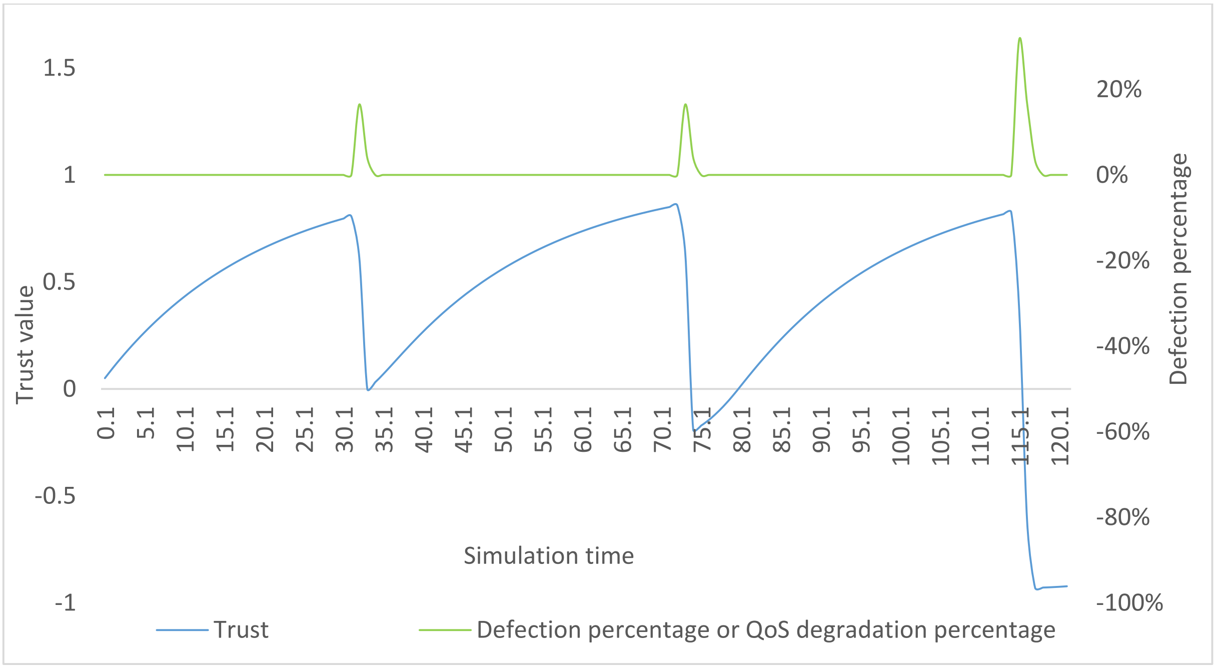


Figure 13. A Sample of the Defection-Cooperation Cycle (SAAS).

This figure depicts the fundamental characteristics of the con-man-resistant trust algorithm. It shows that, the more the unexpected bad service, the less trust there is, and the more the expected positive service there is, the higher the trust is. Figure 13 also shows that, as bad outcomes repeat, the recovery of trustworthiness takes longer.

Figure 14 shows this defection-cooperation cycle for the IAAS data. Here, QoS happens in clusters and is also repeated. From this figure, it is observed that, if the repetition continues further, trust can converge to its minimum possible value of -1. Once the trust converges to -1, it never goes up even though no defections are present.

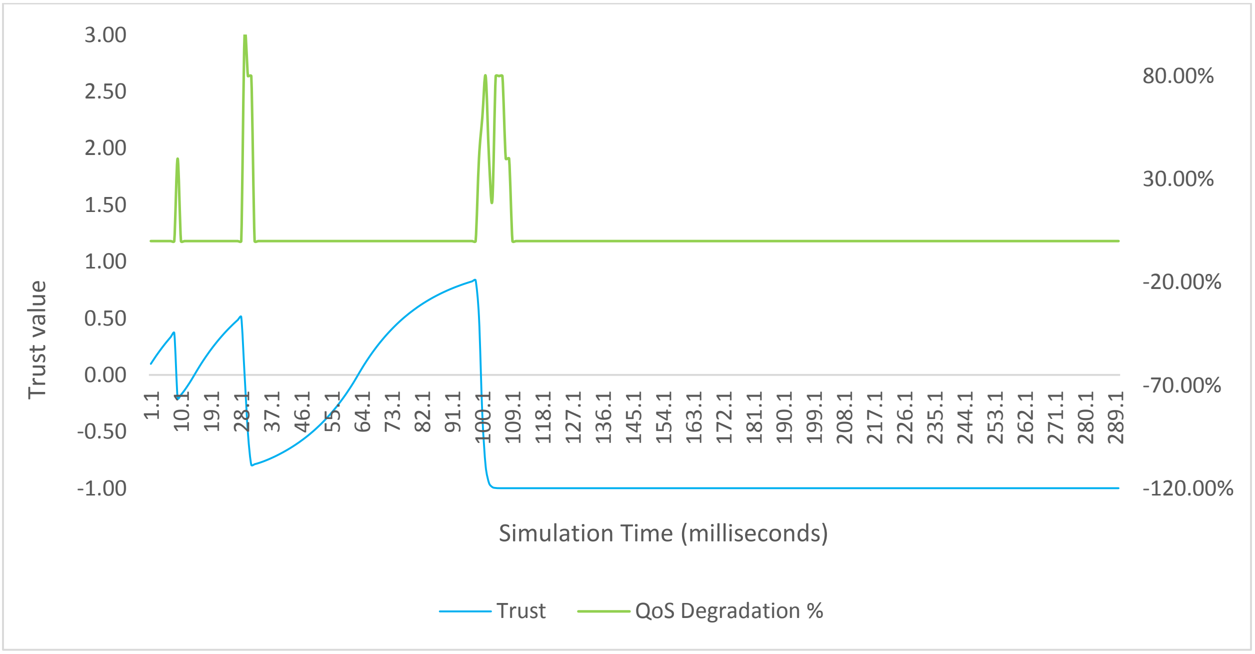


Figure 14. A Sample of the Defection-Cooperation Cycle (IAAS).

The consistency of trust value at -1 establishes the ability of the con-man-resistant trust algorithm to catch the cloud service’s deceptive behavior or, more generically, to catch the deceptive behavior in cyberspace domains. The summary of Figure 14 is, the trust diminishes with repetitive unexpected bad service.

2.5.4.3. Trust-Convergence Results

Figure 15 shows how the trust value converges to -1 as the service provider's con behavior continues for an SAAS simulation. The convergence time depends on variable different combinations. Here, the α_0 and β_0 ratios' effect on trust evolution is shown (A ratio of 1:3 and 1:10 were suggested by previous work [1][2]).

The following observations can be made from this figure:

- For the same β_0, the number of interactions for trust convergence toward -1 is proportional to α_0 i.e. the lower the value of α_0, the faster the convergence or the smaller the number of such interactions.
- For the same α_0, the number of interactions for trust convergence toward -1 is inversely proportional to β_0 i.e. the higher the value of β_0, the faster the convergence or the smaller the number of interactions.

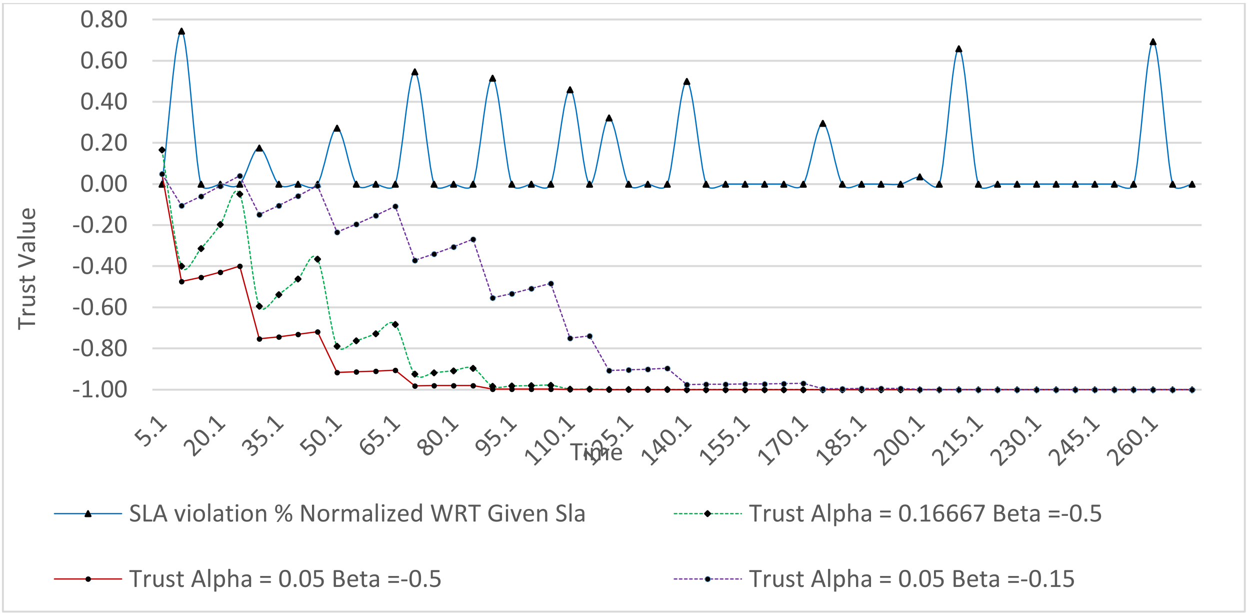


Figure 15. Trust Convergence to -1 for Two Different Parameter Ratios (SAAS).

2.6. Sub Problem 2: Comparing the Con-Man-Resistant Trust Algorithm's Performance

To compare the performance of the con-man-resistant trust algorithm for deception-cycle detection, it is important to select an algorithm that is applicable to solve a similar problem. There is research that counts the performance or QoS fluctuations, giving a low trust score if the fluctuations are greater than expected [11]. The repeated bad service following good service of con man attack scenario is the service quality fluctuates. Hence, the con-man-attack scenario is necessarily the fluctuations of the presented work in [11].

The presented work in [11] is compared with the con-man-resistant trust algorithm in the cloud-computing domain. In the rest of this section, a comparable trust model is described; then, the SLA violation is re-calculated in the con-man-resistant trust model for comparison purposes, followed by describing the inputs for both algorithms and comparing the results.

2.6.1. Comparable Trust Model

There are prior works that provide evaluations for the actual versus promised behaviors of a cloud service, e.g., the research mentioned in [11], [69], [99], and [100]. However, none of them addressed this con-man-trick behavior.

Repeated SLA violations or QoS instability for a cloud service that result from the con-man deception is also an example of fluctuating QoS behavior [11]. Hence, this con-man application is comparable with the fluctuating QoS and flexible SLA work [11], the only prior work that deals with trust modeling for unstable QoS with a cloud resource. In terms of fluctuating QoS, the con-man algorithm gives a lower score for a higher fluctuation of QoS and the algorithm's generated lower score indicates a reduced trustworthiness.

A con-man-resistant trust algorithm was never applied on historical data in the prior research on con-man-resistant trust [3][2][1]. The concept is applied to historical data in this work. The con-man-resistant trust value which results from the historical data is compared with the fluctuating QoS and flexible SLA work. The degree of satisfaction with fluctuating QoS and a flexible SLA, before and after normalization, falls in the range of $\{0\} \cup [0.5, 1]$ and $\{-1\} \cup [0, 1]$, respectively, resulting in aggregated trust: ϵ [0, 1] and ϵ [-1,1], respectively. It is evident that the trust value before normalization is comparable to the con-man-resistant trust, ϵ [-1, 1]. The fluctuating QoS and the flexible SLA work's trust value normalization is eliminated to confirm the similarity and contrast of both trust values: the fluctuating QoS and the flexible SLA trust value and the con-man-resistant trust value.

The consumer's satisfaction is affected by continuous SLA violations or a QoS below the specified threshold. The con-man-resistant trust value reflects the effect of such dynamics. The more persistent the continuity is, the lower the trust value is. The con-man-resistant trust value converges toward its lowest possible value (-1) if the QoS violation repeats. The con-man-resistant trust algorithm calculates the trust value against time. Hence from the beginning of a historical time series data till the end, the trust value changes. This change shows the dynamism of service quality using trust value. However, the fluctuating QoS and flexible SLA process cannot capture such dynamism because the process calculates the trust value after reading all data for a certain historical record [11].

In fluctuating QoS and flexible SLA studies, even though the $overall_{SLA}$ is satisfied, if the $peak_{SLA}$ is violated even once in the individual historical record, the resulting trust value is assigned the minimum possible value (-1) for that record. However, only violating the SLA once in a large time span (historical record) is given a waiver in a complex system such as a cloud. The con-man-resistant trust algorithm assigns a penalty in the trust value for such a violation but does not assign the lowest possible trust value (marked as untrustworthy) all at once.

2.6.2. Pre-Comparison Step: Con-Man-Resistant Trust Algorithm's SLA-Violation Recalculation

For this comparison, the con-man-resistant trust algorithm presented in this dissertation was tuned. The SLA violation was calculated in a prior comparison for this con-man-resistant trust algorithm, was based on the requested and allocated MIPS, and was expressed as a percentage. However, in the fluctuating QoS and flexible SLA research, this SLA violation is recalculated with respect to the $overall_{SLA}$. Hence, for tuning, the presented con-man-resistant trust algorithm also recalculates the SLA violation, resulting in a comparison version for this trust model. This new SLA violation percentage is calculated from the $overall_{SLA}$ and the SLA violation percentage from the simulator using the following equation.

$$\frac{X_{SLA} - overall_{SLA}}{100 - overall_{SLA}} = \frac{SLA_{overallSLA}}{100}$$

Here, X_{SLA} is the SLA violation percentage from the simulator as the current input. $SLA_{overallSLA}$ is the SLA violation with respect to the $overall_{SLA}$, which is calculated from the SLA violation percentage or the current input. If $overall_{SLA}$ = 50% and X_{SLA} or SLA violation percentage = 75%, then $SLA_{overallSLA} = 50\%$. Similarly, for X_{SLA} = 60%, $SLA_{overall} = 20\%$.

2.6.3. Inputs for the Trust Algorithms

A collection of historical records is taken as the input to the fluctuating QoS and flexible SLA method. Hence, a collection of historical records is taken as the input to both the con-man-resistant trust algorithm and the fluctuating QoS and flexible SLA method. Again, in the fluctuating QoS and flexible SLA method, four constraints, called the SLA set, were defined to split the input historical records collection. Hence, in this chapter four SLA set are defined. Each historical record only follows one such constraint. The four categories are as follows:

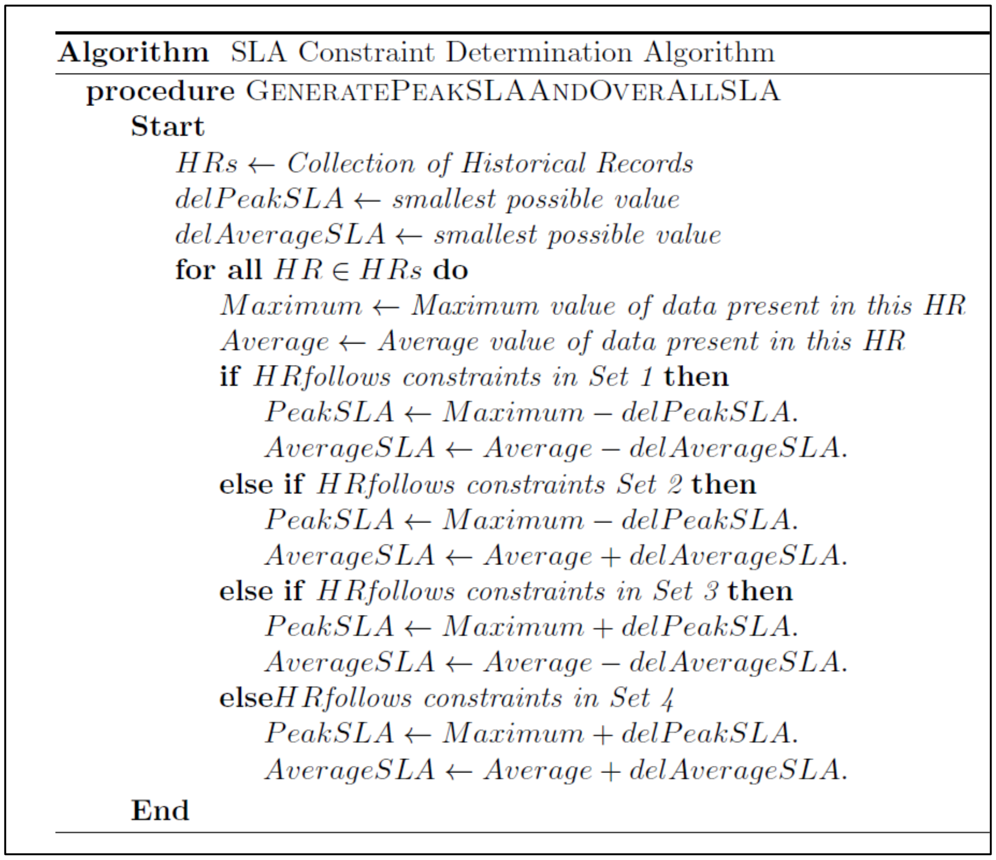
SLA set 1: both $overall_{SLA}$ and $peak_{SLA}$ are violated.

SLA set 2: only $peak_{SLA}$ is violated.

SLA set 3: only the $overall_{SLA}$ is violated.

SLA set 4: neither the $overall_{SLA}$ nor the $peak_{SLA}$ are violated.

The $overall_{SLA}$ and $peak_{SLA}$ values are determined by Algorithm 1. The $overall_{SLA}$ and $peak_{SLA}$ for historical records was determined satisfying their associated SLA set category. For example, in the con-man-resistant trust algorithm version for this comparison, 40% of the historical records' $overall_{SLA}$ and $peak_{SLA}$ are determined with the constraints of SLA set 1, 30% of SLA set 2, 20% of SLA set 3, and 10% of SLA set 4. From each set, e.g., in 40% of the historical records, each record in the 40% has its own $overall_{SLA}$ and $peak_{SLA}$.



```
Algorithm  SLA Constraint Determination Algorithm
  procedure GENERATEPEAKSLAANDOVERALLSLA
    Start
      $HRs \leftarrow$ Collection of Historical Records
      $delPeakSLA \leftarrow$ smallest possible value
      $delAverageSLA \leftarrow$ smallest possible value
      for all $HR \in HRs$ do
        $Maximum \leftarrow$ Maximum value of data present in this HR
        $Average \leftarrow$ Average value of data present in this HR
        if $HR$follows constraints in Set 1 then
          $PeakSLA \leftarrow Maximum - delPeakSLA.$
          $AverageSLA \leftarrow Average - delAverageSLA.$
        else if $HR$follows constraints Set 2 then
          $PeakSLA \leftarrow Maximum - delPeakSLA.$
          $AverageSLA \leftarrow Average + delAverageSLA.$
        else if $HR$follows constraints in Set 3 then
          $PeakSLA \leftarrow Maximum + delPeakSLA.$
          $AverageSLA \leftarrow Average - delAverageSLA.$
        else$HR$follows constraints in Set 4
          $PeakSLA \leftarrow Maximum + delPeakSLA.$
          $AverageSLA \leftarrow Average + delAverageSLA.$
    End
```

Algorithm 1.   SLA Constraint-Determination Algorithm.

The algorithm helps to empirically calculate the $\text{overall}_{\text{SLA}}$ and peak_{SLA} for individual records. Each record may have different values for the $\text{overall}_{\text{SLA}}$ and peak_{SLA} since the records belong to independent consumers. The algorithm calculates the $\text{overall}_{\text{SLA}}$ and peak_{SLA} in such a way that, for a certain record, both SLAs, no SLAs, or only one SLA is violated.

The algorithm calculates both the $\text{overall}_{\text{SLA}}$ and peak_{SLA} for each historical record. The algorithm then determines into which SLA set category the current historical record falls. If the record falls under SLA set 1, then both constraints ($\text{overall}_{\text{SLA}}$ and peak_{SLA}) are violated. The $\text{overall}_{\text{SLA}}$ needs to be below the mean of the record’s individual interactions, and the peak_{SLA} needs to be lower than the maximum value of the interaction data. If the $\text{overall}_{\text{SLA}}$ and peak_{SLA} are generated this way, both the maximum value of a record’s interactions and the average is above $\text{overall}_{\text{SLA}}$ and peak_{SLA}, respectively. Hence, the $\text{overall}_{\text{SLA}}$ is calculated by deducting a small number from the average, and the peak_{SLA} is calculated by deducting a small number from the maximum value.

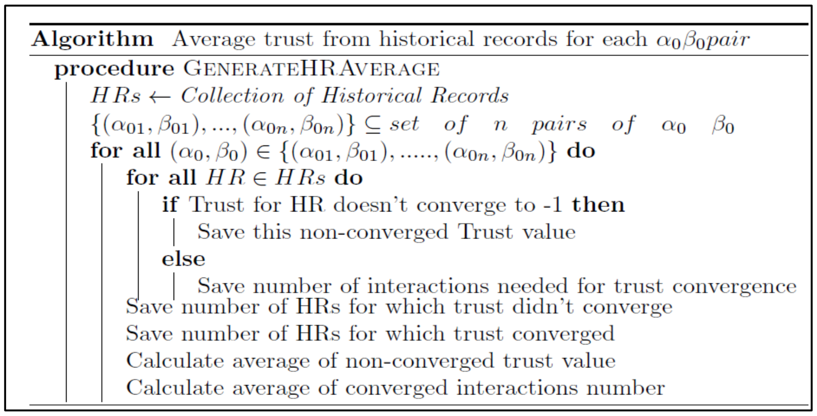
Similarly, if the record belongs to SLA set 2, then only the $\text{overall}_{\text{SLA}}$ is violated; i.e., the $\text{overall}_{\text{SLA}}$ is below the average, but the peak_{SLA} is above the maximum value among the record’s interactions. Hence, the $\text{overall}_{\text{SLA}}$ is calculated by deducting a small number from the average, and the peak_{SLA} is calculated by adding a small number to the maximum value.

If the record is in SLA set 3, then only the peak_{SLA}, not the $\text{overall}_{\text{SLA}}$, is violated. Hence, $\text{overall}_{\text{SLA}}$ is calculated by adding a small number to the average, and the peak_{SLA} is calculated by deducting a small number from the maximum value.

The records in SLA set 4 do not have any SLA violations; i.e., neither the $\text{overall}_{\text{SLA}}$ nor the peak_{SLA} is violated. Hence, if the record falls in SLA set 4, then the $\text{overall}_{\text{SLA}}$ is calculated by adding a small number to the average, and the peak_{SLA} is calculated by adding a small number to the maximum value. The algorithm ends with a result where each record has a corresponding overall$_{SLA}$ and peak$_{SLA}$.

2.6.4. Trust-Model Comparison

With previous con-man research, pairs of α_0 and β_0 were suggested. The con-man-resistant trust algorithm is applied to each historical record for a particular α_0 β_0 pair. Trust either converges to -1 or not for each α_0 β_0 pair which is applied to the historical record. For each pair's each historical record where trust converges to -1, the interaction numbers' average is calculated. With the historical records where trust did not converge, the average trust for each α_0 β_0 pair is calculated. Algorithm 2 shows this process.



```
Algorithm  Average trust from historical records for each $\alpha_0\beta_0 pair$
  procedure GenerateHRAverage
    $HRs \leftarrow Collection\ of\ Historical\ Records$
    $\{(\alpha_{01}, \beta_{01}), ..., (\alpha_{0n}, \beta_{0n})\} \subseteq set\ of\ n\ pairs\ of\ \alpha_0\ \beta_0$
    for all $(\alpha_0, \beta_0) \in \{(\alpha_{01}, \beta_{01}), ....., (\alpha_{0n}, \beta_{0n})\}$ do
      for all $HR \in HRs$ do
        if Trust for HR doesn't converge to -1 then
          Save this non-converged Trust value
        else
          Save number of interactions needed for trust convergence
      Save number of HRs for which trust didn't converge
      Save number of HRs for which trust converged
      Calculate average of non-converged trust value
      Calculate average of converged interactions number
```

Algorithm 2. Average Trust from Historical Records for Each α_0 β_0 Pair.

A collection of historical records is taken as the input to both the fluctuating QoS and flexible SLA method and the con-man-resistant trust algorithm. During the comparison of these two methods, 10 historical records (HRs) are used. Some HRs are identified as untrustworthy by the con-man-resistant trust algorithm (The trust value converged to -1.) for a certain α_0 β_0 pair.

For the untrustworthy HRs, Figure 16 shows the average number of interactions. Nine α_0 β_0 pairs are present.

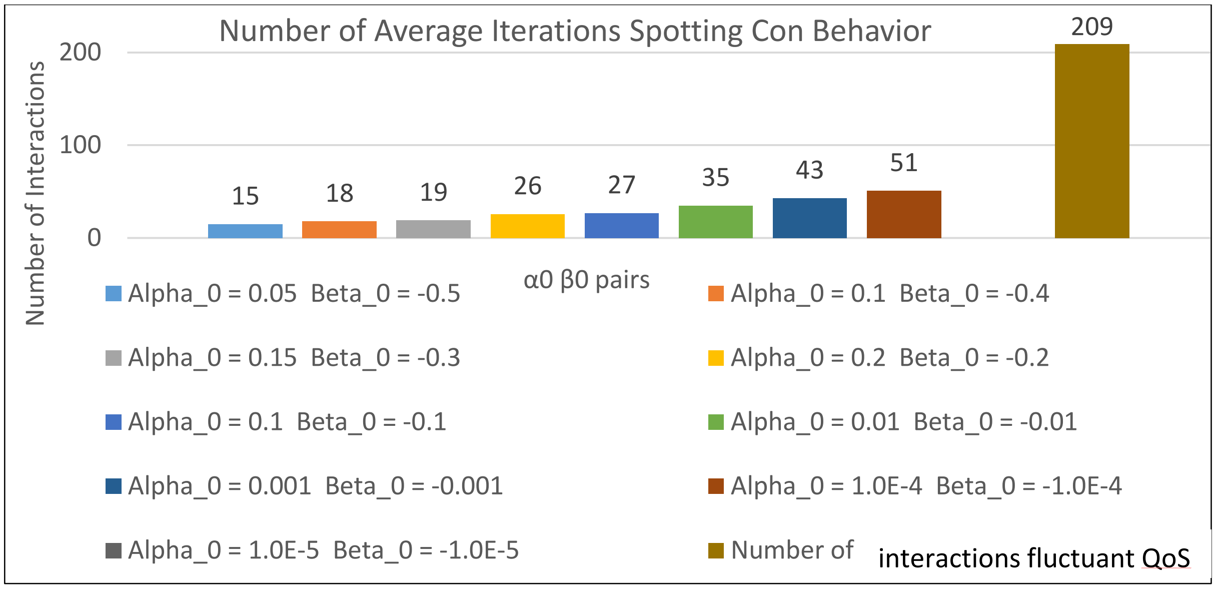


Figure 16. Number of Average Interactions for Converged Trust with Each α_0, β_0 Pair.

Only the untrustworthy records' interaction number to spot untrustworthiness is averaged for each pair. The figure shows that the pairs took a minimum of 15 interactions and a maximum of 51 interactions to identify the con behavior (when the trust value converged to -1). The fluctuating QoS and flexible SLA method took 209 interactions (the average length of all HRs) but never gave a trust value of -1. The fluctuating QoS and flexible SLA were never able to spot the con deception. Therefore, it is clear that the con-man-resistant trust algorithm takes fewer interactions, with respect to the fluctuating QoS and flexible SLA methods, to spot the con deception.

When the fluctuating QoS and flexible SLA method and the con-man-resistant trust algorithm are applied on the collection of HRs, some HRs are identified as trustworthy by the con-man-resistant trust algorithm.

Figure 17 shows how many HRs are identified as trustworthy and how many HRs are identified as not trustworthy i.e. how many HRs detected the con deception. Most HRs (9) spotted the con behavior, and 1 HR did not, for all α_0 β_0 pairs except 1.

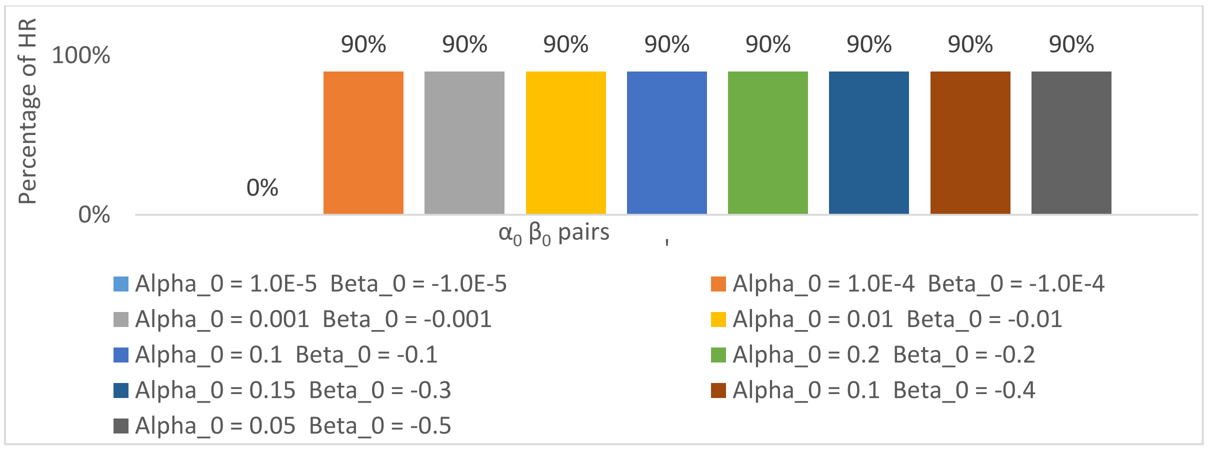


Figure 17. Percentage of HRs Converging Toward Untrustworthiness.

For the trustworthy HRs, trust values from the method and the algorithm are compared and shown in in Figure 18. The average con-man-resistant trust value for trustworthy HRs is shown for a certain α_0 β_0 pair. Trust converged to -1 (con deception detected) for most of the α_0 β_0 pairs (with 1 exception in 9), for 90% of the HRs.

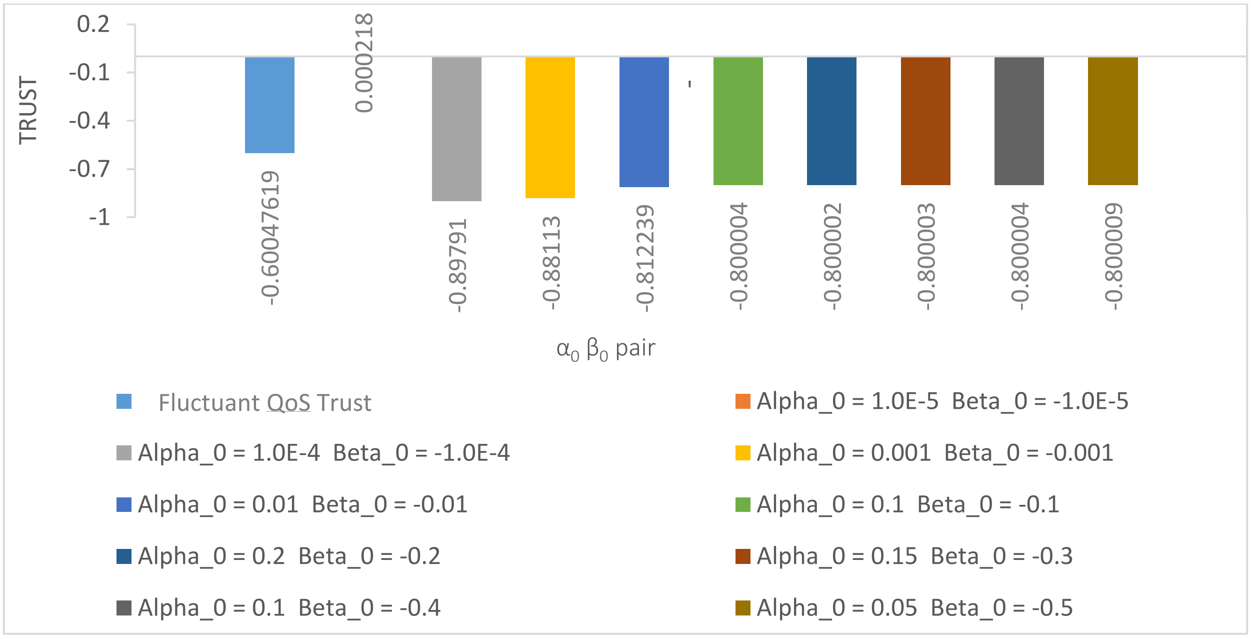


Figure 18. Number of Average Interactions for Converged Trust with Each α_0 β_0 Pair.

The figure also shows that the trust values for the con-man-resistant trust algorithm are closer to untrustworthy with respect to the fluctuating QoS and flexible SLA model. For example, the minimum trust value is about -0.9, and the maximum is -0.8 (except for the one trustworthy α_0 β_0 pair with a trust value of 0.0002).

The following findings come from Figure 18:

- β_0 and the number of interactions are inversely related: the lower the β_0, the higher the number of interactions.
- Con-man-resistant trust values for most α_0 β_0 pairs are lower than the fluctuating QoS and flexible SLA trust values.

2.7. Sub Problem 3: Deception Detection with the Batch Processing of Data

2.7.1. Trust from Windows

A time window that consists of fixed- or variable-size time units or time stamps for the time series plays an important role when trust is calculated for time-series data. There is a plethora of research on trust calculation considering a time window: [28], [101], [102], [103], [104], [105], [106], [107], [108], and [109].

An oscillating service-performance behavior example in a peer-to-peer network is described in [28]. This oscillating behavior is an example of the con-man-attack pattern or the con-man deception in this dissertation. A malicious peer considers the peer-to-peer network's environment as a game. The peer can maximize its profit by deploying a trick that is not detectable by the game's rules. The amount of cheating can be large. For example, peers can cheat by creating a highly positive reputation at the beginning. Then, they can start cheating occasionally so that the cheating does not affect their existing reputation. This cheating model gives the malicious peer a profit over the non-cheating model, i.e., the model where the peer maintains good service all the time, because this ideal case of maintaining good service all the time is investment intensive. This cheating model is identical to the con man's repeated defection which is presented in this dissertation.

Another trick, targeting profit maximization, is an oscillation of building a good reputation and then taking advantage of that reputation [28]. When the reputation goes below the acceptable value, then an investment is made to rebuild the reputation. The cycle of reputation building and diminishing continues. This cycle, or oscillation, is the con-man behavior in this dissertation.

The authors of [28] applied a dynamic trust model which considered a recent time window to deal with the oscillation behavior. One peer calculates the trust of all other peers from the feedback about these peers in the recent time window. A smaller time window is considered when a peer's performance is doubted. The smaller time window reacts better when the peers' dynamic behavior is present. The dynamic behavior consists of both the peers' performance issues and the previously described malicious behavior. The behavior is the con-man behavior in this dissertation. Hence, for this dissertation, the con-man-resistant trust algorithm is calculated by including the recent time window as the current game-theory interaction to calculate a cooperating or defection behavior.

In the peer-to-peer network, a smaller window is used to catch the dynamic or malicious behavior [28]. The window is kept small in order to the catch sudden changes for this dynamic behavior. The "recent" window is included to exclude the older reputation data during the trust calculation.

In e-commerce, a seller provides good transactions followed by bad transactions. The dynamic trust is calculated by taking all the buyer's transactions with a system from a certain time duration or window. Trust is a weighted sum of the concerned buyer’s trust regarding the seller and a reputation calculation. The buyer’s trust is another weighted sum of the buyer’s trust from the current transaction and the previous transactions in the recent time window. The reputation is the average of the trust from all other buyers in the system during that time window (similar to the concerned buyer’s trust).

Similarly, in e-commerce, a smaller sliding time window is included for trust calculation [109]. The larger time window gives enough time for a malicious agent to perform fraudulent transactions before the agent’s trust value (in terms of reputation) diminishes. On the other hand, a smaller time window reduces the severity of the malicious agent’s activity due to early detection.

In the con-man-resistant trust algorithm for cloud services, the following concepts about a time window are of great importance:

- Excluding older interactions.
- Keeping the window size smaller to catch sudden interaction-pattern changes during the current trust calculation.
- Keeping the window size smaller to reduce the deception’s effect with early detection.

2.7.2. Trust from a Non-Overlapping Window versus Trust from a Sliding Window

The time window can be non-overlapping and sliding (overlapping). Trust can be calculated using both methods. For example, a sliding time-window concept is applied in [110] to measure dynamic trust with each window. In every window, good and poor interaction counts are recorded. Trust concept is used to identify malicious nodes. Each interaction is the cooperation between two nodes. The interactions happen at every time unit or time stamp. This interaction is equivalent to the interplay between a consumer and a cloud provider in the con-man-resistant trust algorithm.

The non-overlapping window for time-series data does not share any time-unit or time-stamp data between windows. Each time the window proceeds forward with time in a sliding window, the oldest time-unit data are dropped, and the new time-unit data are added.

Each window in the non-overlapping window represents an interaction of the con-man-resistant trust algorithm. Each window contains a collection of time-unit data where every time unit is when the consumer receives service from the cloud-service provider. In such a non-overlapping window, the same issue is not considered twice; i.e., if a performance issue is found inside a window, then the issue does not appear in another window.

On the other hand, time stamps in a sliding window are shared; hence, a performance issue for a certain time stamp can appear in more than one consecutive window. Therefore, in a sliding window, the same performance issue propagates into multiple windows, influencing consecutive trust values. This sliding-window property allows the same performance issue to penalize the service provider multiple times because the issue affects consecutive trust values. Hence, in this dissertation, sliding-window-based trust is discouraged.

2.7.3. Window-Based Con-Man Algorithm

In the prior con-man algorithm research, data read for particular times were considered as individual interaction [1][2][3]. For example, if a consumer receives a service from a service provider at time $t = 0$ then the consumer had an interaction with the provider at $t = 0$. At the next time $t = 1$ if the consumer again receives a service from the same provider, then another interaction happened at $t = 1$. Each interaction resulted in one trust value corresponding to that time. The most-recent interaction generates the latest trust. An extension of the con-man trust algorithm is implemented in this work to account for a fixed number of multiple data entries over time as one window of data. Each window is further counted as one interaction for this con-man algorithm.

For example, if datum no. 0 is read, then the first window would end at datum no. “window size –1,” where window size represents how many data the algorithm shall group for the current interaction of the con-man algorithm (one interaction). The second window is from data no. “window size” to “2 × window size -1. Hence, window i starts at data no. “(i -1) × window size” and ends at data no. “i × window size -1,” inclusive for i ϵ ℕ. The number of data in the window is the “window size.” Figure 19 conceptualizes this scenario.

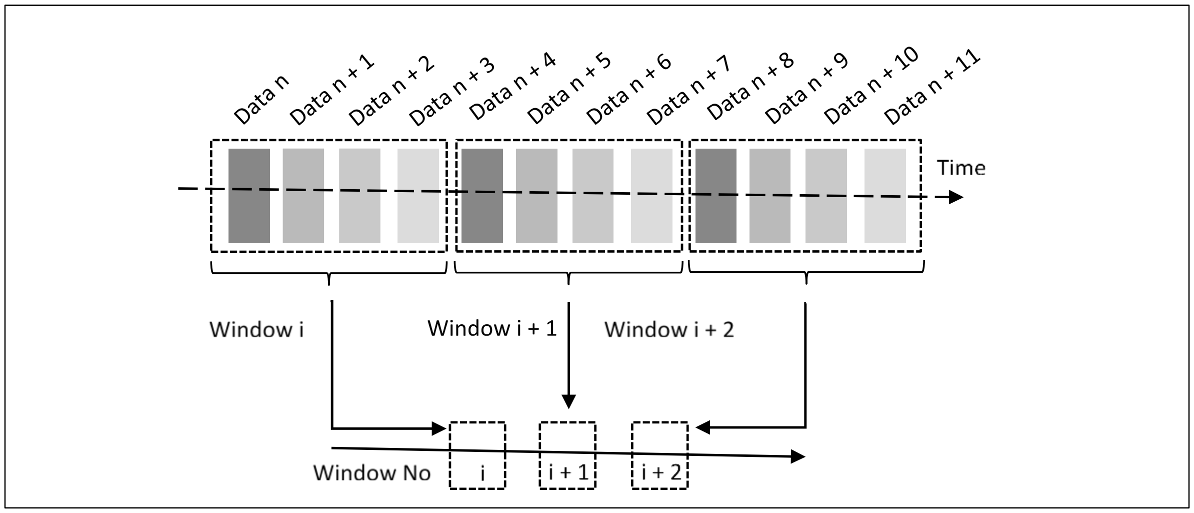


Figure 19. Each Window Considered as Individual Con-Man Datum.

In every window, a collection of data corresponding to each time represents a single interaction. A trust value is calculated for each interaction. The following equation confirms the window size.

$$\mathrm{i} \times \text{window size} - 1 - (\mathrm{i} - 1) \times \text{window size} + 1 = \text{window size}$$

2.7.4. Window-Based, Con-Man-Resistant Trust Algorithm Properties

Two methods are considered to unlock the properties of the window-based con-man algorithm. The first method is inspired from the fluctuating QoS and flexible SLA work [11]. A threshold or an overall value is considered in a time window. The area above and below the threshold is calculated using area integration. If the area above is less than the area below or if any data value inside the window reaches a predefined peak value, then the corresponding interaction for that window is taken as a defection. If none of the conditions are true for that data window, cooperation happens. This peak value is 100%, and the trapezoidal rule of integration is applied in this application. Whenever defection happens, the defection percentage is considered as 100% because only the constant-punishment method is applied for this window-based approach or batch processing of data. Cooperation is indicated by the 0% defection percentage.

In the second method, the defection percentage corresponding with each data value is aggregated. Hence, an aggregation method needs to be selected. The cooperation or defection is determined based on the difference between the aggregated value's average and a threshold (the same threshold value mentioned in the previous paragraph). The simplest aggregation method, the mean of data values inside this window, is chosen. The corresponding time for this window is the time of the last data inside the window. There is a case when both methods result in the same defection percentage for the interaction window. The case is when there is no data inside the window with 100% SLA violation and when the trapezoidal rule of integration is applied for the first method. Only for this case, the following statements hold:

- $Integration\ area\ above ==$
  $Integration\ area\ below\ results\ no\ SLA\ violation \simeq$
  $average\ is\ overallSLA\ results\ no\ SLA\ violation$

- $Integration\ area\ above < Integration\ area\ below\ results\ no\ SLA\ violation \simeq average\ is\ below\ overallSLA\ results\ no\ SLA\ violation$
- $Integration\ area\ above > area\ below\ results\ 100\%\ SLA\ violation \equiv average\ is\ above\ overallSLA\ results\ 100\%\ SLA\ violation.$

2.7.5. Window-Based Algorithm's Property Analysis

2.7.5.1. Trust Measure for the Area Above Threshold Versus the Area Below Threshold

Figure 20 shows how the trust value changes when the defection's presence is recalculated for a batch of data or for each window following the first method of defection detection. In the figure, for the con-man-resistant trust algorithm, the parameter setting used is: $\alpha_0 = 0.05$ and $\beta_0 = -0.5$. All the data represented against time in Figure 20 have the corresponding defection amount as percentages with few time-stamps where no defection happened.

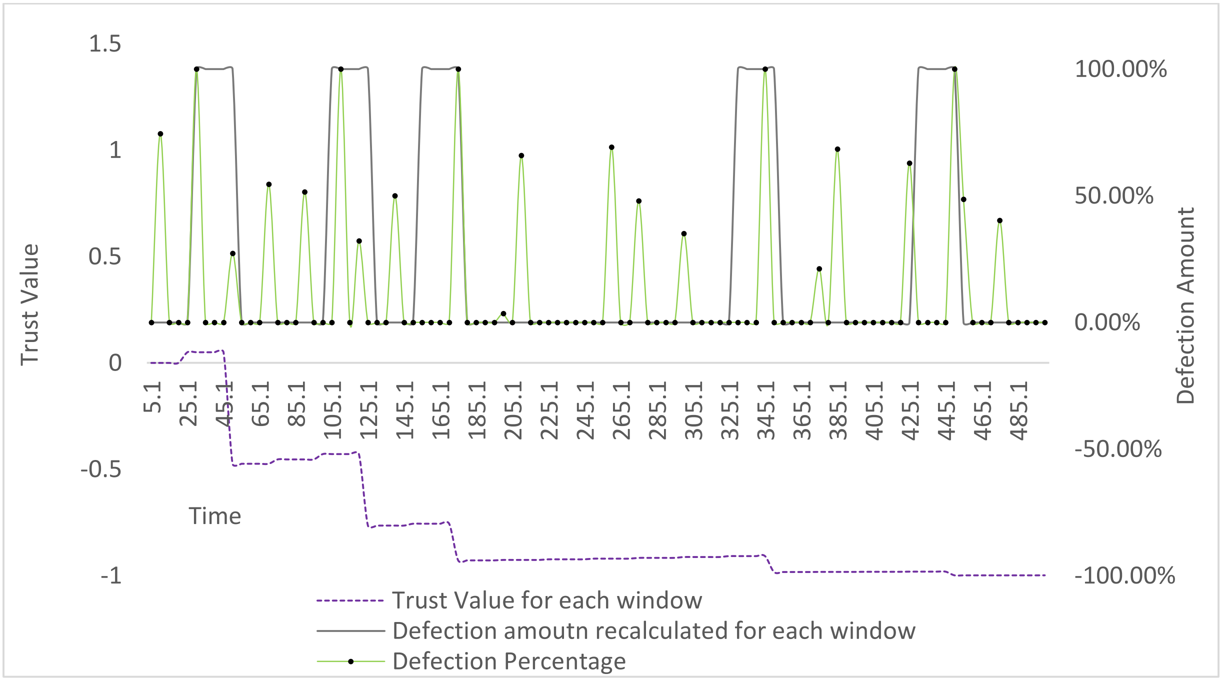


Figure 20. Trust Values over Time with the Area Above Threshold versus the Area Below Threshold as a Measure of the Defection Presence for Each Window (SAAS).

Each window is 5 consecutive data against time. An integration of area above a threshold, e.g., 25% defection percentage, is compared against the area below that threshold, i.e., inside that window.

The two conditions determining the defection presences in a window, for the first or the integration method, are: if for any time inside a window, the defection percentage is 100% or in the window, the integrated area above the threshold is larger than the area below that threshold. The conditions are called the defection-detection conditions. A defection is present in a window if any of the conditions is true. The figure shows that, for five time windows of size 5, at least one defection-detection condition is true. The height of the bars for the defection-presence line represents the defection's existence; 100% indicates the presence, and 0% means the absence of cooperation.

2.7.5.2. Average of Defection or Cooperation Within Each Window as a Measure of Trust

Figure 21 shows how the trust value changes when the defection presence is recalculated for a batch of data or for each window following the second method of defection detection.

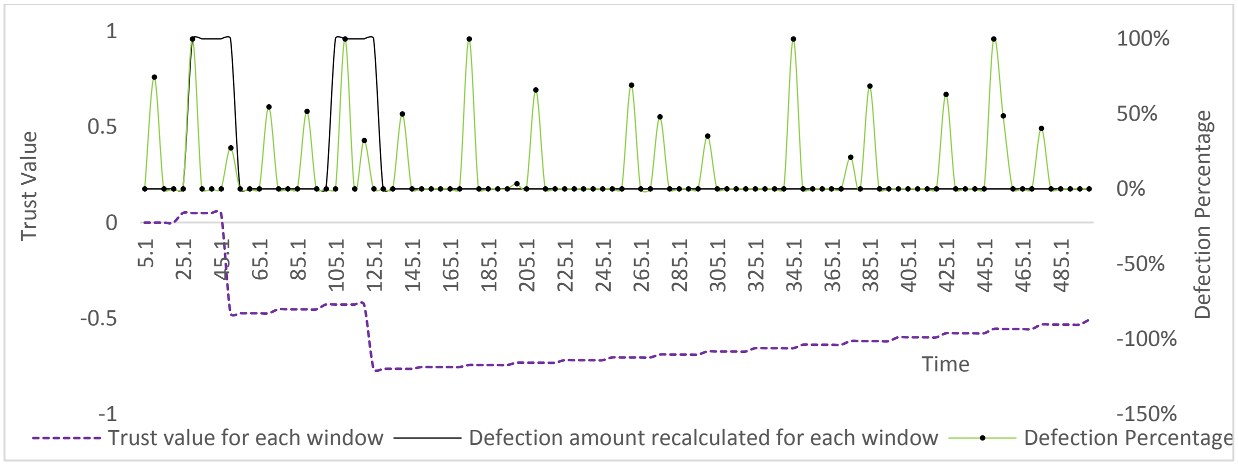


Figure 21. Trust Values over Time with the Average for the Defection or Cooperation Amount Within Each Window as a Measure of the Defection Presence for Each Window.

The defection percentages are averaged using the arithmetic mean for 5 consecutive data points against time (window size 5). If this average is above a threshold, e.g., 25%, then it is considered as a defection. Here, punishment categories are not used; hence, if defection happens in each window, it is 100%. The figure shows that for two windows (windows ending at 50.1 milliseconds and 130.1 milliseconds respectively) the average defection percentage is above the threshold (25%). For rest of the windows the average is below the threshold. Hence only for the two window, the defection is detected and the corresponding con-man-resistant trust value drops at 50.1 milliseconds and 130.1 milliseconds respectively. The con-man-resistant trust value eventually gets higher value over time as for rest of the windows no defection is detected.

2.7.5.3. Trust-Measures Comparison: Average versus Area Integration

Figure 22 shows the defection-presence calculation in a window using the two methods described earlier. The trust-value evolution is also presented.

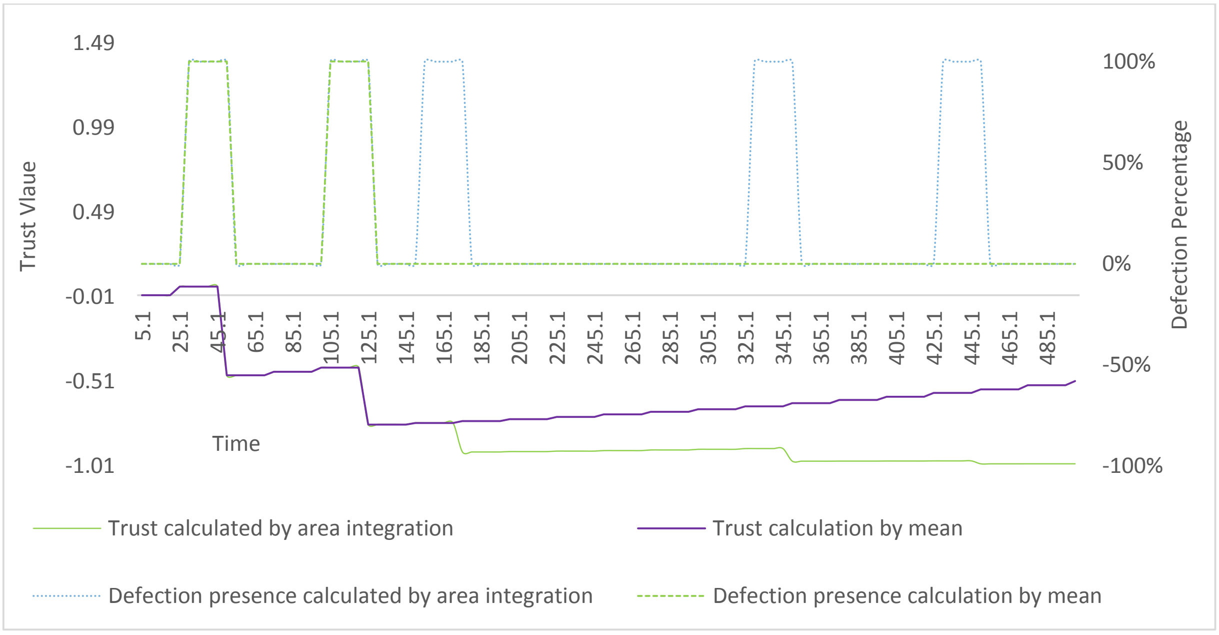


Figure 22. The Trust Values' Evolution for the Average and Area-Integration Method.

The following principle can be concluded from this trust-value evolution:

Principle: Trust converges faster or with an equal number of interactions for the defection-presence calculation process with area integration compared to the mean or average.

Consider that there is at least one data point inside any window for which the defection percentage is equal to 100%. In such a scenario, trust converges faster or with an equal number of interactions for area integration when compared to the mean or average. However, if no defection amount is 100%, then the convergence time, or number of interactions, is identical for these two methods of window-based defection calculation.

2.7.5.4. Trust-Value Evolution Comparisons for Different Window Sizes

Figure 23 shows the trust-value evolution over time with two different window sizes: 5 and 10. Defection-presence calculation with an average is applied with 10% as the threshold.

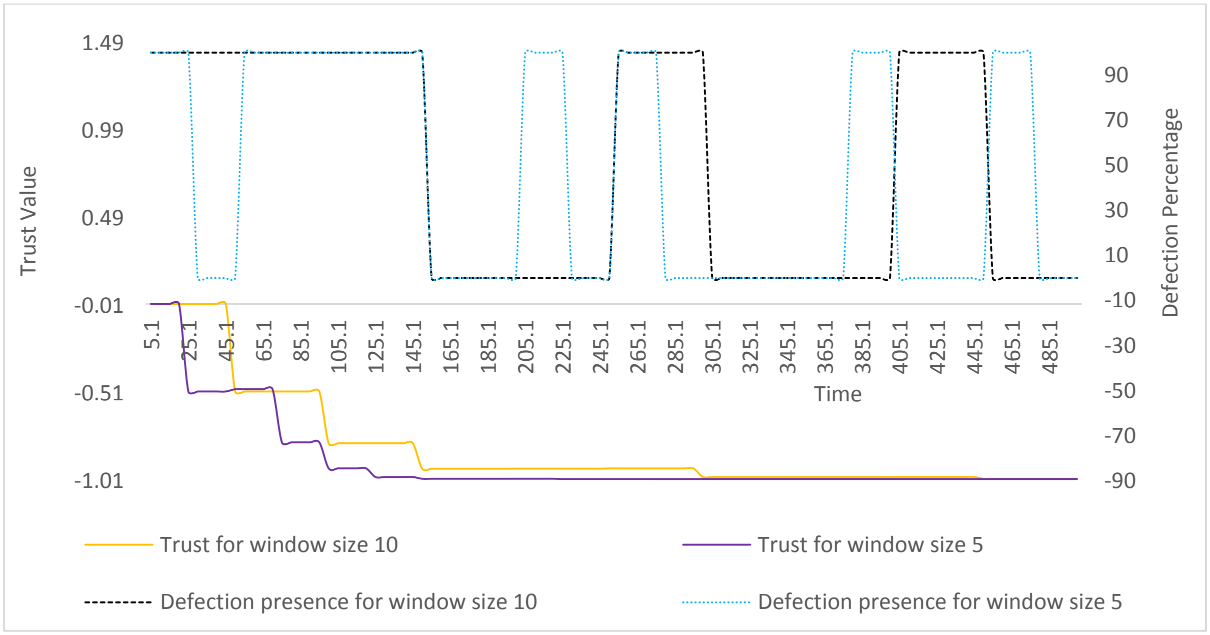


Figure 23. Trust-Value Evolution Comparisons: Window Sizes 5 and 10.

The following observation is established from Figure 23:

Principle: The number of interactions needed to detect untrustworthiness is proportional to the window size; i.e., trust converges to -1 faster or with an equal number of interactions for a smaller window.

2.7.5.5. Trust-Value Evolution Comparisons for Different Thresholds

Figure 24 shows the trust-value evolution over time for two different thresholds during the defection-presence calculation from windows. The window size is 5. The following observation is made from this result:

Principle: The number of interactions needed to detect untrustworthiness is proportional to the threshold; i.e., trust converges to -1 faster for lower thresholds.

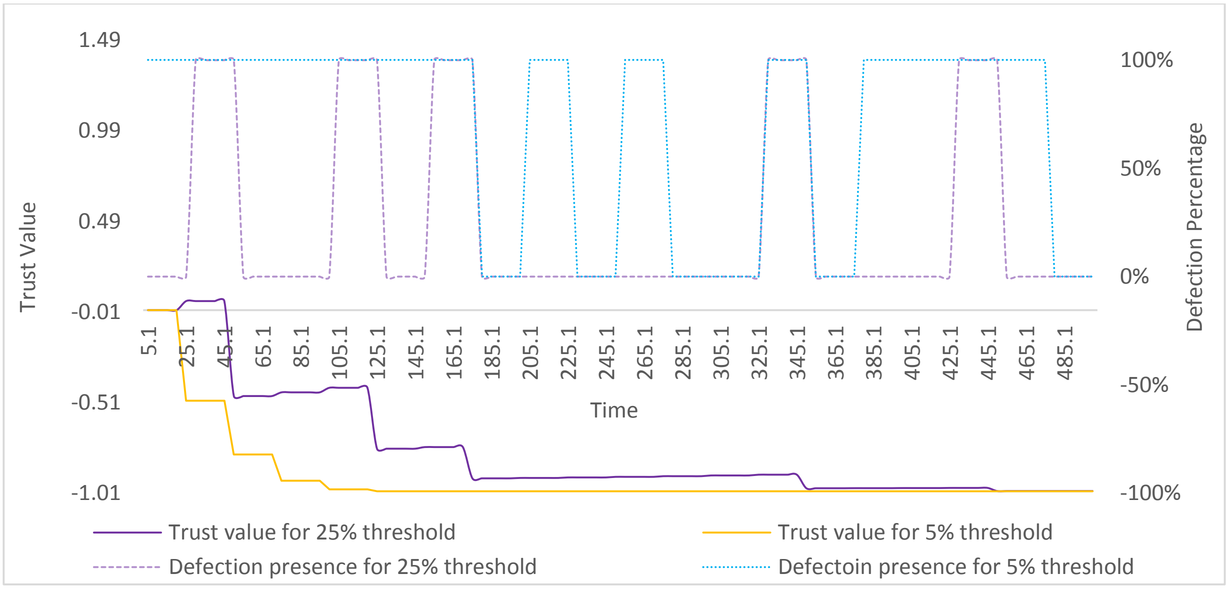


Figure 24. Trust-Value Evolution Comparisons: 5% versus 25% Threshold.

2.8. Recommended Applications for the Implemented Con-Man-Resistant Trust Algorithm

Two attributes of the SLA violation are chosen for this dissertation: SLA-violation magnitude and SLA-violation repetition (frequency) over time. With more repetition, the customer has less satisfaction. Again, the higher the SLA-violation magnitude, the lower the customer’s satisfaction. Such repetition is an example of the con behavior. This repetition of the SLA violation is essentially a cycle of bad and good interactions; this cycle is also interchangeable with instability for a cloud service’s QoS. The con-man-resistant trust algorithm is capable of detecting such a cycle. When the cloud-service provider does not resolve this issue, despite having revenue from the consumer, the bad part of the cycle keeps affecting the consumer. An example of this effect is financial loss. This cyclic behavior is a fraud from the cloud-service provider. The con-man-resistant trust algorithm can detect this fraud.

Besides this fraud-cycle-detection application, the algorithm can be applied for other purposes. The algorithm can detect unstable resources for any service-oriented architecture. This recommended use is inspired from the first application of the con-man-resistant trust algorithm for utility computing, e.g., in a smart electrical grid [3]. This con-man trust model was applied in the smart electrical-grid domain to select stable nodes for a smart power-grid control system. “Stable” means a node where the power frequency does not vary beyond a limit. The less stable the node’s frequency is, the more it affects other components of the grid. This stability issue stresses the grid’s operation. Hence, detecting such a stability issue is crucial. The repetitive unstable behavior is comparable to the con man’s cyclic behavior of good and bad service. Hence, the con-man-resistant trust algorithm can detect this stability issue. The con-man-resistant trust algorithm can be applied to determine the stability of cloud-computing resources following the smart-grid control system’s application of the algorithm.

Table 11 lists the recommended applications for this con-man-resistant trust algorithm in the cloud-computing domain. The recommended applications are those utilizations where stable behavior for the cloud resource is desired.

Table 11. Applications for the Recommended Con-Man-Resistant Trust Algorithm.

Possible Application	Description
Stability-concerned applications	The con-man-resistant algorithm can be applied to determine the stability of cloud-computing resources. This application can replace the existing stability-calculation process for the cloud-computing system described in [111].
Cloud resource management	The con-man trust model in cloud-resources stability can also be applied as an alternative method for cloud-resource allocation and load-balancing applications [112][67]. Only trusted resources, i.e., stable resources, can be selected with SLA-violation frequency as the stability-measurement metric.
Assessing fluctuant QoS behavior	Repetitive SLA violations is an example of an unstable or fluctuating QoS. Hence, the con-man-resistant trust algorithm can be applied to measure the fluctuating QoS behavior of the cloud resources by replacing the static trust model with the fluctuating QoS and flexible SLA [11].
Cloud-service monitoring	Any SLA-violation-based measure that can indicate possible customer dissatisfaction is desirable for the service providers. Moreover, any comprehensive number of parameters' value representing the dissatisfaction is further desirable. The con-man algorithm can catch the dissatisfaction dynamics with a smaller number of parameters. Also, the con-man-resistant trust algorithm can be applied to monitor the QoS for both real-time and historical data, and the algorithm can be embedded with a third-party monitor [113]. Hence, this con-man trust algorithm is of great interest to the cloud-service providers.
Selecting con-man-attack resistant resources	To make a system con-man-attack resistant, resources with the maximum con-man-resistant trust value can be selected. This con-man algorithm can be used in the existing trust-management module [66], as a part of trust calculator inside a trust engine [114], and in a trust-aided evaluation module [115]. The algorithm can be utilized in the first phase of the trust-aided evaluation module: trust formulation. Although the trust-formulation unit computes trust values based on the direct trust values and reputation values, the con-man trust values can be used as a direct trust values. From the perspective of the con-man trust value, any service providers with a trust value of 1 can be on a suggestion or recommendation list.

2.9. Conclusion

Deceptions in cyberspace are becoming quite diverse and unpredictable. Hence, it is important to predict and to model the deceptions. The developed model is useful for resisting the deception. A con-man attack in the cloud services is empirically studied in this chapter by modeling and observing the behavior from the model's results and by applying a resistance algorithm to find the deception. The result shows how quickly the con-man behavior is identified with the con-man-resistant trust value. Additionally, the findings show how well the con-man-resistant trust algorithm works with respect to another algorithm which aims to identify service-quality issues. Like the cloud services, this deception can happen with other service types in cyberspace. The loss associated with the deception is significant. Hence, the work presented in this chapter contributes to cybersecurity by attempting to minimize the con deception in cyberspace.

CHAPTER 3. CON-MAN-RESISTANT TRUST ALGORITHM PROPERTIES STUDY

3.1. Introduction

The con-man-resistant trust algorithm's properties are studied in this chapter. Certain principles and properties emerge from this study, and they serve as a guide for setting parameter values. Depending on the parameter values, the con-behavior-detection velocity varies. The velocity models rate of interactions which are necessary to detect the con behavior when the deceptive behavior prevails. A severe end user may choose a small number of interactions before labeling a con, whereas a more forgiving end user may choose a larger number of interactions. The specific configuration of the parameter values must be chosen for each domain. For example, for cyber-attack-prone cloud applications, a relatively small number of interactions to detect the con behavior might be the best choice, whereas for secondary cloud-storage capacity, a larger number of interactions may be appropriate. Hence, the study presented in this chapter contributes results for choosing the appropriate configurations of the con-man-resistant trust algorithm to proactively detect a con-man attack.

An enhancement for this algorithm is proposed: the recursion depth utilized in the mathematical scheme for the algorithm is reduced. This adjustment can potentially reduce the lag of dependencies between the con-man-resistant trust algorithm's parameters.

The remainder of this chapter is organized as follows: Section 3.2 states the problem. Section 3.3 summarizes this chapter's Contribution. Section 3.4 presents The Con-Man-Resistant Trust Algorithm's Properties with Enhancement. In section 3.4, Sub-section 3.4.1 has the State Diagram of the Con-Man-Resistant Trust Algorithm. Sub-section 3.4.2 explains the algorithm's properties in terms of its parameters. Sub-section 3.4.3 shows The Con-Man-Resistant Trust Algorithm's Trust-Calculation Step Reduction which is a revision for the trust-updating scheme.

Sub-section 3.4.4 elaborates the Study of the Con-Man-Resistant Trust Algorithm's Parameter Properties. Sub-section 3.4.5 explains the observed and evaluated principles from the Simulation Results. Finally, Section 3.5 concludes the chapter by summarizing the presented work.

3.2. Problem Statement

This chapter evaluated several con-man-resistant properties for the trust algorithm during the algorithm's use in cyberspace [4]. The con-man-resistant trust algorithm utilizes a mathematical procedure to calculate trust value. The mathematical scheme's properties depend upon the initial value for several parameters. Certain parameter configurations can result in improved efficiency for the algorithm.

It is important to know the correct parameter configuration and how one parameter affects another one in order to use this algorithm effectively. Hence, it is necessary to evaluate the parameter values' effect on each other by evaluating the dependency chain between the parameters. In the same way, understanding the effect of the parameter ratios, the decimal precision, and the exponential-function-based constants on trust evaluation and evolution is important.

The properties of consecutive defection (a bad outcome from an end user's perspective) without cooperation or the expected outcome can appear in service-oriented architecture's services, e.g., in the cloud services. Similarly, consecutive cooperation (an expected outcome) without defection (a bad outcome) can prevail. Studying the con-man-resistant trust algorithm's properties in the scenario is important.

The core of this con-man-resistant trust algorithm is its mathematical procedure; the scheme defines a recursion that leads to one parameter's dependency on its own or another parameter's prior values. The depth of the recursion also controls the effectiveness of this algorithm along with the parameter configuration. Hence, it is important to study this recursion and to evaluate the reduced calculation steps.

3.3. Contribution

This chapter's contribution is the study of the con-man-resistant trust algorithm's properties and the proposed enhancement. The result of this study is a set of principles that a con-man-resistant trust algorithm follows and a revised trust-updating scheme for this algorithm. These principles and the enhancement guide the con-man-resistant trust algorithm's parameter configuration. The detection pace of the con behavior depends upon this configuration proving that these principles and the enhancement are promising. The con-man resistant trust algorithm is formally modeled by the interaction pattern presented in the following equation in terms of formal language (L) over the alphabet $\sum = \{C, D\}$ [116][117].

$$L = \left\{\left(DC^{\theta_i}\right)^{+} \mid i = 1 \ldots n, \theta_i \epsilon\, \mathbb{N}\right\}$$

The cycle here is defection and cooperation. There are i cycles. The amount of cooperation (C) is θ_i in each cycle i, followed by one defection (D).

The implemented con-man-resistant trust algorithm in this chapter reveals that, if $\theta'_i > \theta_i$, i.e., the number of cooperation interactions between two defections is more than θ_i, then the con-man-resistant trust algorithm cannot detect this con behavior (Figure 25).

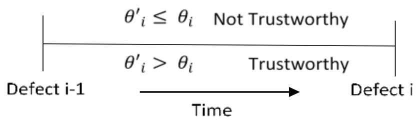

$\theta'_i \leq \theta_i$ Not Trustworthy
$\theta'_i > \theta_i$ Trustworthy
Defect i-1
Defect i
Time

Figure 25. Undetected Con Man.

The algorithm has a trust value as output. If this trust is -1, then the cloud service is untrustworthy; if the trust is +1, then the service is trustworthy [4]. The consumer expects a zero or positive trust value. Negative values are discouraged because they can converge towards -1, or untrustworthiness, faster than positive values. Trust convergence means that the trust value is moving toward either +1 or -1. Again, θ_i results from a fixed α_0, β_0 pair with certain number of interactions. The specific number of interactions implies that the results of prior work [1][2][3][4] were also for a certain number of interactions. However, in the previous research, no such fixed interaction numbers were reported. This chapter reveals that the α_0, β_0 pair and the α_0-to-β_0 ratio affect the evolution of the trust value which is calculated using the con-man-resistant trust algorithm. Also, the trust-value calculation formula, called the “trust-updating scheme” in prior work, has dependencies on the con-man-resistant trust algorithm’s parameter value. The trust-updating scheme is also revised in this chapter.

Trust-value convergence also depends upon the constant C. Prior research presented results for $C = e^{-1}$. However, for $C = e^{-2} \ldots e^{-10}$, the results’ convergence times are different; this finding is shown in the chapter’s result section. Additionally, the decimal precision plays a big role in determining the trust value. Hence, the convergence time of the con-man-resistant trust algorithm towards -1 or +1 is also dependent on the latter two parameters along with the α_0, β_0 pair. The chapter’s contribution is showing the influence of the α_0, β_0 pair; the α_0 to β_0 ratio; C; and the decimal precision for the con-man-resistant trust algorithm’s trust value along with an enhancement for the trust-updating scheme.

3.4. The Con-Man-Resistant Trust Algorithm’s Properties with Enhancement

3.4.1. State Diagram of the Con-Man-Resistant Trust Algorithm

In this chapter, the con-man-resistant trust algorithm’s state diagram is derived. Figure 26 illustrates the state diagram. The trustworthy node has a trust value of +1 for a cloud-service provider to be trusted and, hence, in a trustworthy state. The cooperation arc from the trustworthy node to itself depicts that, when a service provider is trustworthy (+1), further cooperation or good service does not improve the trust value because +1 is the maximum trustworthy value.

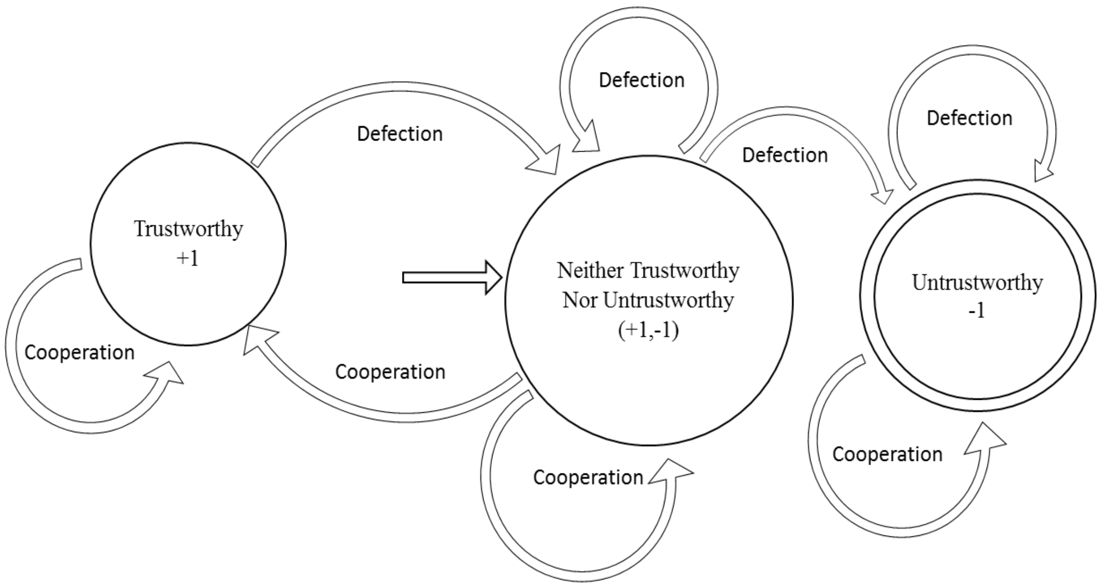


Figure 26. State Diagram of the Con-Man-Resistant Trust Algorithm.

If defection or a lapse in service occurs, the trust value is diminished, and the service provider’s trust value becomes less than +1. When the trust value is less than 1, the service provider is neither trustworthy nor untrustworthy. Hence, another node labeled neither trustworthy nor untrustworthy indicates this service-provider state. When the trust value becomes less than 1, the service provider transitions from the trustworthy state into the “neither trustworthy nor untrustworthy state. This transition is represented by the arc’s defection from the trustworthy node to the neither trustworthy nor untrustworthy node.

The neither trustworthy nor untrustworthy node presents the provider’s state where the service provider achieves a trust value between +1 and -1. The value explicitly indicates that the service provider’s state is neither trustworthy nor untrustworthy. This state is the start-state of the state diagram for the con-man-resistant trust algorithm because, at the very beginning of a service evaluation, the cloud-service provider is neither trustworthy nor untrustworthy. In Figure 26’s diagram, the start arrow explicitly illustrates that this node is the start state. When cooperation or good service continues, the trust value increases. This change in trust value is indicated by the cooperation arc from this node to itself. The increment results in the trust value reaching +1 at some point. When the trust value is +1, the service provider reaches the trustworthy state (transition to the trustworthy node). This transition is indicated by the cooperation arc from the current node to the trustworthy node. Again, when a defection or lapse in service continues in the current state (neither trustworthy nor untrustworthy), the trust value is diminished. This defection, or lapse in service, is indicated by the defection arc from the neither trustworthy nor untrustworthy node to itself. The defection or a lapse in service can result in a trust value of -1. When the -1 value is achieved, the service provider is no longer trustworthy, and the service provider is tagged as a “con man.” Hence, when the -1 value is achieved, a transition from the neither trustworthy nor untrustworthy state to a new state happens and is called the untrustworthy node. This transition is shown by the defection arc from the neither trustworthy nor untrustworthy node to the untrustworthy node.

At the untrustworthy node, the service provider already has a trust value of -1. If further defection or service lapses happen, the trust value does not decrease because the trust has already reached the minimum value. This change in trust value is shown by the defection loop from the untrustworthy node to itself. Once the con man or the cloud-service provider becomes untrustworthy (trust value of -1), it cannot recover; i.e., “once untrustworthy, always is untrustworthy” or “once a con, always a con.” The value of α is $(0, \alpha_0]$, and β is $(-1, \beta_0]$. When defection happens for a significant amount of time, then α and β have their smallest values in the diagram and are at the untrustworthy node. If cooperation or good service continues after the defections or service lapses, the α and β values never increase. Hence, the trust value never changes. This scenario of unchanged trust value explains why, once the trust value reaches -1 or the service provider is identified as untrustworthy, the provider can never become trustworthy. Hence, there is no arc from the untrustworthy node to either of the remaining nodes: neither trustworthy nor untrustworthy and trustworthy. The double circle of the node representation for the untrustworthy state explicitly shows that this node is the dead state.

3.4.2. The Con-Man-Resistant Trust Algorithm's Behavior with Respect to the α_0 to β_0 Ratio, Continuous Cooperation, and Defection

As another contribution to con-man-resistant trust algorithm, a study about the con-man algorithm's behavior when continuous defection or cooperation happens is presented in the result section. The following principles present the findings. Principles 1 and 2 are combined into Principle 3.

- Principle 1: When continuous defection happens without any cooperation, then θ is not dependent on α or α_0.
- Principle 2: When continuous cooperation happens without any defection, then θ is not dependent on β or β_0.
- Principle 3: The α-to-β ratio does not affect θ when consecutive defection happens without any cooperation or when consecutive cooperation happens without any defection.

In the previous research about the con-man-resistant trust algorithm [1][2][3][4], the parameter which influenced the convergence time or the number of interactions necessary for trust to converge towards its extreme values (trustworthiness or untrustworthiness) was the pair

$\{\alpha_0, \beta_0\}$. Hence, in this chapter, only one pair is used for most calculations. Again, following the previous works, the α_0 to β_0 ratios used in this chapter are 1 to 3 [3] and 1 to 10 [1]. A comparison of these ratios is presented in later sections.

3.4.3. The Con-Man-Resistant Trust Algorithm's Trust-Calculation Step Reduction

This algorithm calculates trust recursively. The recursion depth is 4 in the latest work [3]; i.e., if the current trust is for interaction N with a defect, then it is dependent on parameters down to N-3. Consecutive defection is utilized to evaluate the defection parameter's effect on the trust value (defection at N, N-1, N-2, and N-3). The recursion depth of 4 in Table 12 is reduced into 3 in Table 13 and into 2 in Table 14.

Table 12. Recursion Steps for the Con-Man-Defection Formula.

Recursion Depth	Recursion Steps for the Con-Man-Defection Formula		
	Interaction No.	Defection Present	Formula (Recursion)
0	N	Yes	$T_N = T_{N-1} + \beta_{N-1} (1 + T_{N-1})$
1	N-1	Yes	$\beta_{N-1} = \beta_{N-2} – \gamma_{N-2} (1 + \beta_{N-2})$
2	N-2	Yes	$\gamma_{N-2} = C \times T_{N-3}$
3	N-3	Yes or No	$T_{N-3} =$ …. … …

Table 13. Reduction of Recursion Steps for Con-Man-Defection Formula

Recursion Depth	Recursion Steps for the Con-Man-Defection Formula		
	Interaction No.	Defection Present	Formula (Recursion)
0	N	Yes	$T_N = T_{N-1} + \beta_N (1 + T_{N-1})$
			$\beta_N = \beta_{N-1} – \gamma_{N-1} (1 + \beta_{N-1})$
1	N-1	Yes	$\gamma_{N-1} = C \times T_{N-2}$
2	N-2	Yes	$T_{N-2} =$ …. … …

Table 14. Further Reduction of the Recursion Steps for the Con-Man-Defection Formula.

Recursion Depth	Recursion Steps for the Con-Man-Defection Formula		
	Interaction No.	Defection Present	Formula (Recursion)
0	N	Yes	$T_N = T_{N-1} + \beta_N (1 + T_{N-1})$
			$\beta_N = \beta_{N-1} – \gamma_{N-1} (1 + \beta_{N-1})$
1	N-1	Yes	$\gamma_{N-1} = C \times T_{N-1}$
			$T_{N-1} =$ …. … …

3.4.4. Study of the Con-Man-Resistant Trust Algorithm's Parameter Properties

With prior con-man-trick related research, not all parameters that bias the con-man-resistant trust value were studied [1][2][3][4]. These parameters are analyzed in this chapter. When the con man interacts with the victim's expected behavior (expected service quality), then a parameter called the reward parameter (α) is incremented. This parameter keeps track of how well the con man (or the cloud-service provider) meets the expected behavior or service quality from the victim's (the consumer) perspective. A high value for this parameter indicates a high quality for the interactions and vice versa. This parameter value represents the interactions' positivity or the service quality's goodness. Hence, this reward for cooperation (parameter α) is always a positive value.

The more consecutive or repetitive the unexpected behavior of the con man (the cloud-service provider) toward the victim is, the worse the quality of interactions (unexpected service quality). Hence, this negative behavior is tracked by a variable called the defection parameter (β) and is always a negative value. A higher value of |β| or, equivalently, a lower value of β represents a lower-quality interaction (unexpected service quality).

When cooperation happens, the trust value is incremented by an amount dependent on α. Hence, the trust increment is dependent on how many high-quality interactions the victim or consumer experienced. Alternatively, how much of the expected service quality that the victim or consumer experienced is reflected by this trust increment.

Similarly, when defection happens, the trust value is decreased by an amount which is dependent on β. This trust value diminish represents how much unexpected service quality the victim of con behavior or the consumer experienced.

When cooperation happens, α is increased. Therefore, if cooperation happens consecutively, α experiences a series of increments. This consecutive increment is how a high α value represents a series of cooperation.

When defection happens, α is reduced and |β| is increased or β is decreased. If defections happen consecutively, both α and β experience a series of declines. Hence, a low α value and a low β value individually indicate that there is a series of defections or unexpected bad service from the cloud-service provider.

The con-man-resistant trust algorithm does not increment β when cooperation happens. Hence, any subsequent β value with respect to the interaction number is less than or equal to the previous β; i.e., $\beta_N \leq \beta_{N-1} \leq \beta_{N-2} \ldots \leq \beta_0$, where N represents the interaction number. This trend for β is explained for the first time in this chapter by using the following mathematical deductions.

C is a constant with value $e^{-1} < 1$ [1][2][3][4]. The following equation represents a variable biasing the trust decrement upon defection.

$$\gamma_D = C \times |T|$$

Here, T is the trust from a previous interaction. At any time, the con-man-resistant trust algorithm's trust value is between +1 and -1.

$$0 \leqslant |T| \leqslant 1$$

These results give

$$0 \leqslant C \times |T| \leqslant C.$$

Hence,

$$0 \leqslant \gamma_D \leqslant C < 1 \qquad (1).$$

Again,

$$\beta_N = \beta_{N-1} - \gamma_{DN-1} \times (1 + \beta_{N-1}) \quad (2).$$

If $-1 < \beta_{N-1} < 0$ then,

$$0 < (1 + \beta_{N-1}) < 1 \qquad (3).$$

Equations 1 and 3 give

$$0 \leqslant \gamma_{DN-1} \times (1 + \beta_{N-1}) < 1.$$

Hence from Equation 2, the following can be concluded:

$$\beta_N \leqslant \beta_{N-1}$$

Initially, $\alpha = \alpha_0$ and $\beta = \beta_0$ with $\alpha_0 \leq |\beta_0|$. As time goes by, if the defection-cooperation cycle continues, then the α value decreases toward 0, and the β value decreases toward -1. The decrement results, at any time $\alpha_0 \leq \alpha < 0$ and $-1 < \beta \leq \beta_0$ or $|\beta| \geq |\beta_0|$. Hence, $\alpha \in [0, \alpha_0)$ and $\beta \in (-1, \beta_0]$ (Figure 27). These α, β values range result $\alpha \leq |\beta|$ at any time.

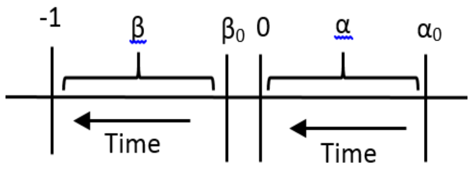


Figure 27. Convergence of α and β.

The α is dependent upon β, and β is dependent upon γ_D. If the drop in trust needs to be slower than what is observed, then |β| needs to be lowered (e.g., β_0 from -0.1 to -0.05). To slow down increasing trust, α needs to be slowed (e.g., α_0 from 0.1 to 0.05). In summary, to achieve this lowering the pair $\{\alpha_0, \beta_0\}$ needs to be lowered.

3.4.5. Simulation Results

Chapter 2 presented the con-man-resistant trust algorithm's application with cloud services in order to catch the cloud-service provider's deception [4]. In that chapter, the cloud services were simulated on top of the Cloudsim simulator, and the service's quality was monitored against the simulated time from the Cloudsim simulator [30][31]. The con-man-resistant trust algorithm for the cloud services evaluated trust by comparing the actual versus the promised service quality for the simulated cloud services. Poisson inter-arrival time (several λ) was used to simulate the repetition intervals of the cloud service's quality degradation. This repetition simulated the con-man behavior in Chapter 2.

The data from Chapter 2 are used in this chapter to study the con-man-resistant trust algorithm parameters' behavior and the enhanced mathematical scheme's effectiveness. The rest of this subsection gives results from the simulation.

3.4.5.1. Effect of the Con-Man Algorithm Parameters' Values on Each Other

Defection repetition affects parameter values in this sequence: previous trust changes previous γ (Figure 28); previous γ changes current β; current β changes current trust; previous β and previous α (Figure 29) change current α. In Figure 28, trust at time t is influenced by the γ_D at time t-3 following the formula mentioned in previous con-man research [1][2][3][4].

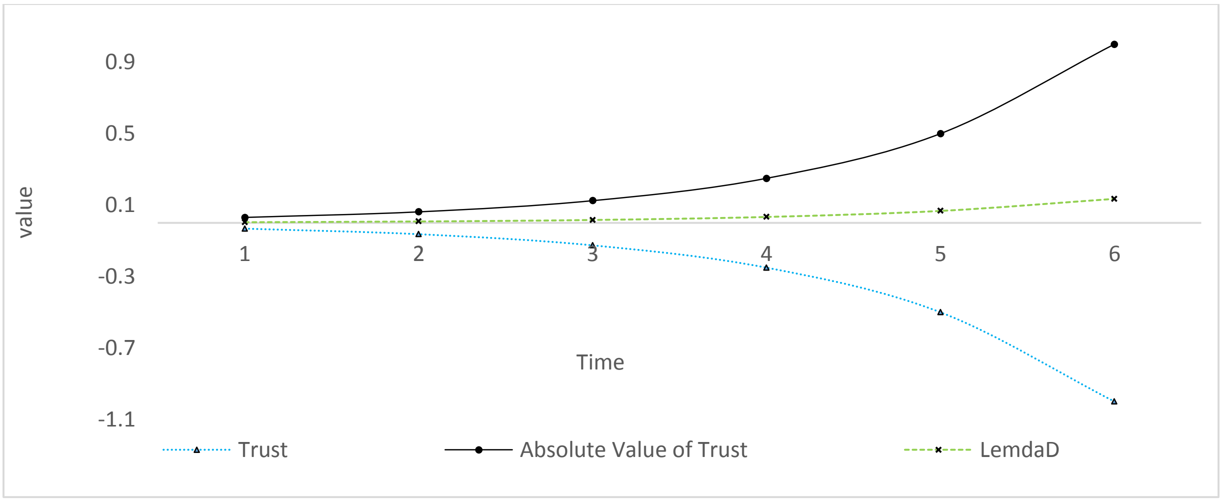


Figure 28. γ_D Changes with Dropping Trust.

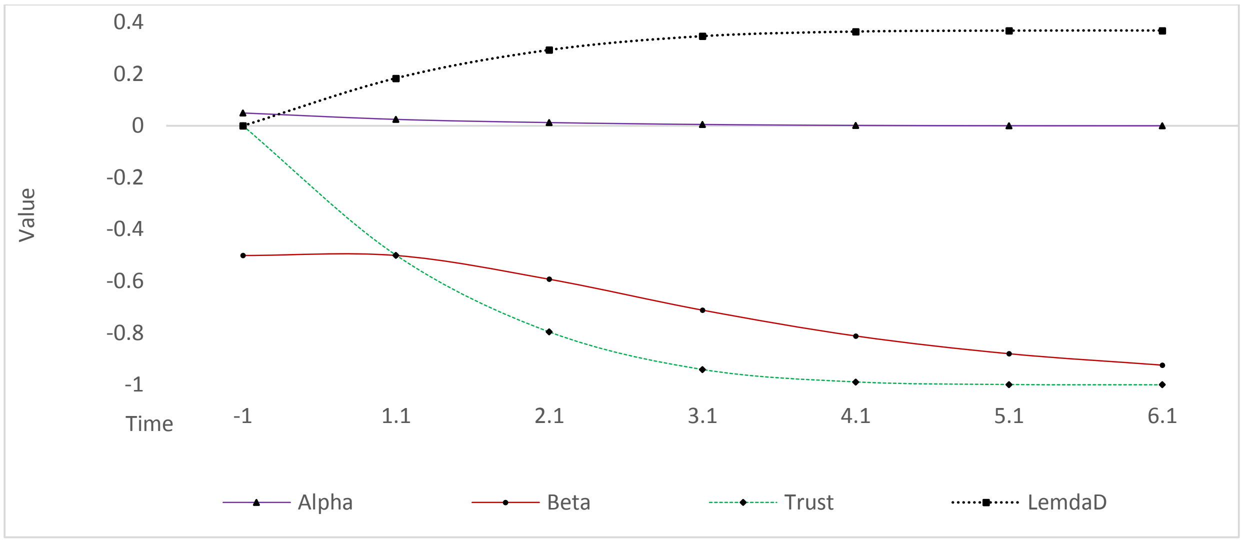


Figure 29. Trust Evolves with α and β Changes, Where α and β Change with the γ_D.

3.4.5.2. The α_0 to β_0 Ratio's Effect on Consecutive Defection or Consecutive Cooperation

Figure 30 shows the proof of principle 1-3. Here, the service-level agreement (SLA) violation percentage represents the service quality's deficiency.

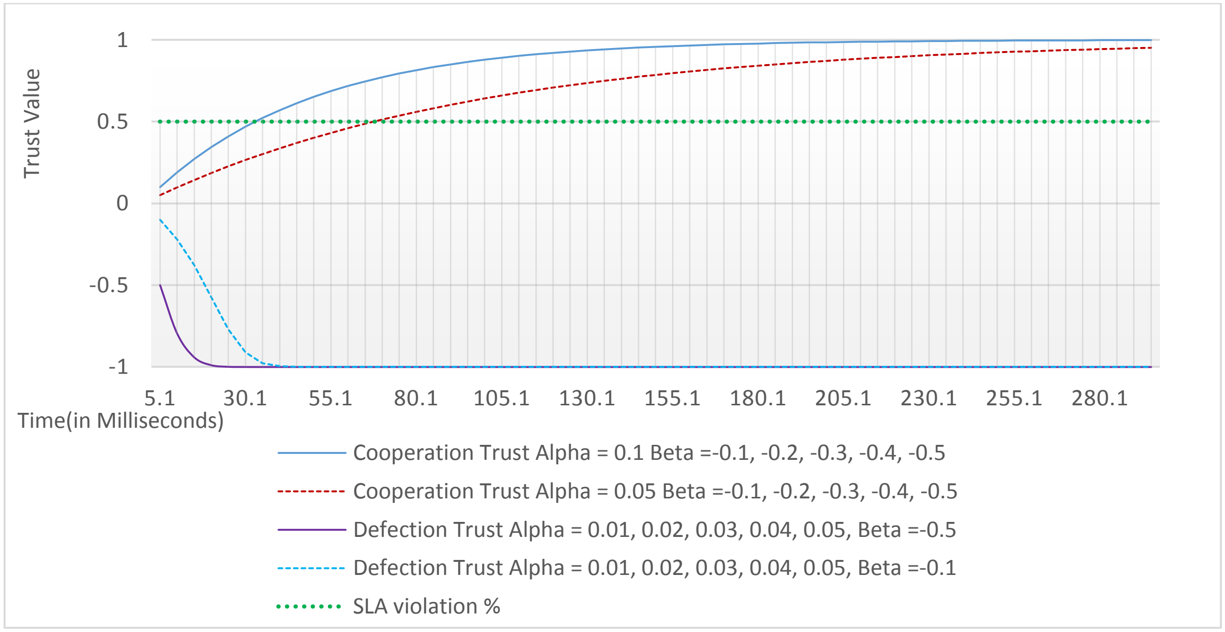


Figure 30. Consecutive Cooperation or Defection is Unaffected by β_0 and α_0, Respectively.

The following principles are evaluated with this figure:

- Principle 1: Consecutive cooperation and the α_0 to β_0 ratio: the number of interactions needed for trust convergence toward +1; i.e., how fast trust converges does not depend upon β_0 and, hence, does not depend upon the α_0 to β_0 ratio.
- Principle 2: Consecutive defection and the α_0 to β_0 ratio: the number of interactions needed for trust convergence toward -1; i.e., how fast trust converges does not depend upon α_0 and, hence, does not depend upon the α_0 to β_0 ratio.
- Principle 3.1: Consecutive defection and $|\beta_0|$: $|\beta_0|$ is inversely proportional to number of interactions needed for trust convergence toward -1; i.e., the higher $|\beta_0|$ is, the faster the convergence or the smaller the number of interactions.
- Principle 3.2: Consecutive cooperation and α_0: α_0 is inversely proportional to the number of interactions needed for trust convergence toward +1; i.e., the higher α_0 is, the faster the convergence or the smaller the number of interactions.

3.4.5.3. Comparison of the α_0 to β_0 Ratio's Values

Figure 31 shows the comparison of two α_0 to β_0 ratios.

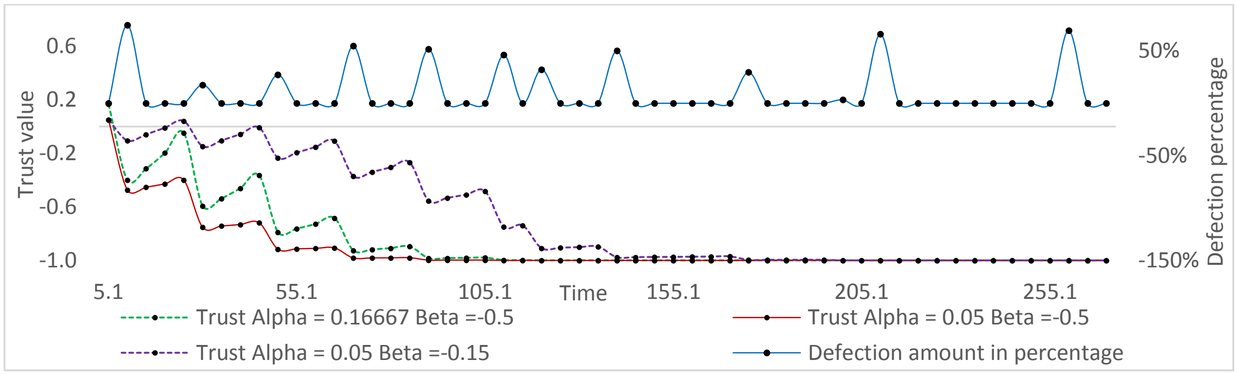


Figure 31. Trust Comparisons for Two Different Parameter Ratios.

The following principles are evaluated with this figure:

- When $|\beta_0|$ is kept the same, the number of interactions for trust convergence toward -1 is proportional to α_0. It means that, the lower the α_0 is, the faster this convergence or the smaller the number of interactions.
- When α_0 is kept the same, the number of interactions for trust convergence toward -1 is inversely proportional to $|\beta_0|$. It means that, the higher the $|\beta_0|$, the faster the convergence or the smaller the number of interactions.

3.4.5.4. Trust-Convergence Interaction Number with Respect to e-1

Figure 32 and Figure 33 show the number of interactions needed to catch the con man's behavior with respect to different C values when bad outcomes or unexpected service quality repeats (for decimal precision = 5).

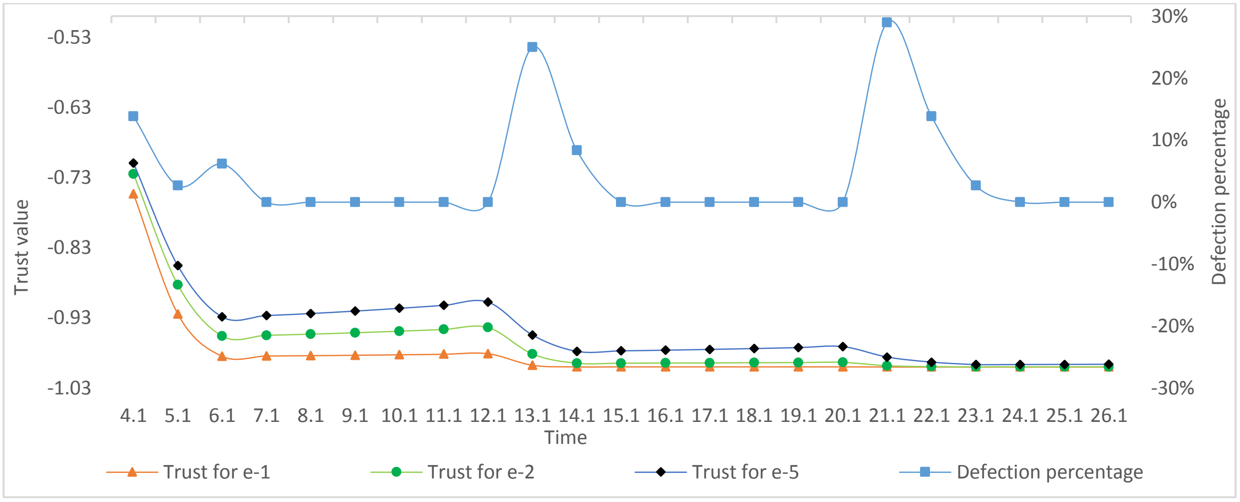


Figure 32. Trust Evolution for Different C = e^{-1}, e^{-2}, e^{-5} Values.

As described before, con-man behavior is detected when the trust value becomes -1 (value indicating untrustworthiness). Figure 32 shows the trust value's evolution toward -1 for decimal precision = 5, $\alpha_0 = 0.05$, and $\beta_0 = -0.5$. Figure 33 shows the number of interactions for 10 different C or exponent values as well as 10 different α_0, β_0.

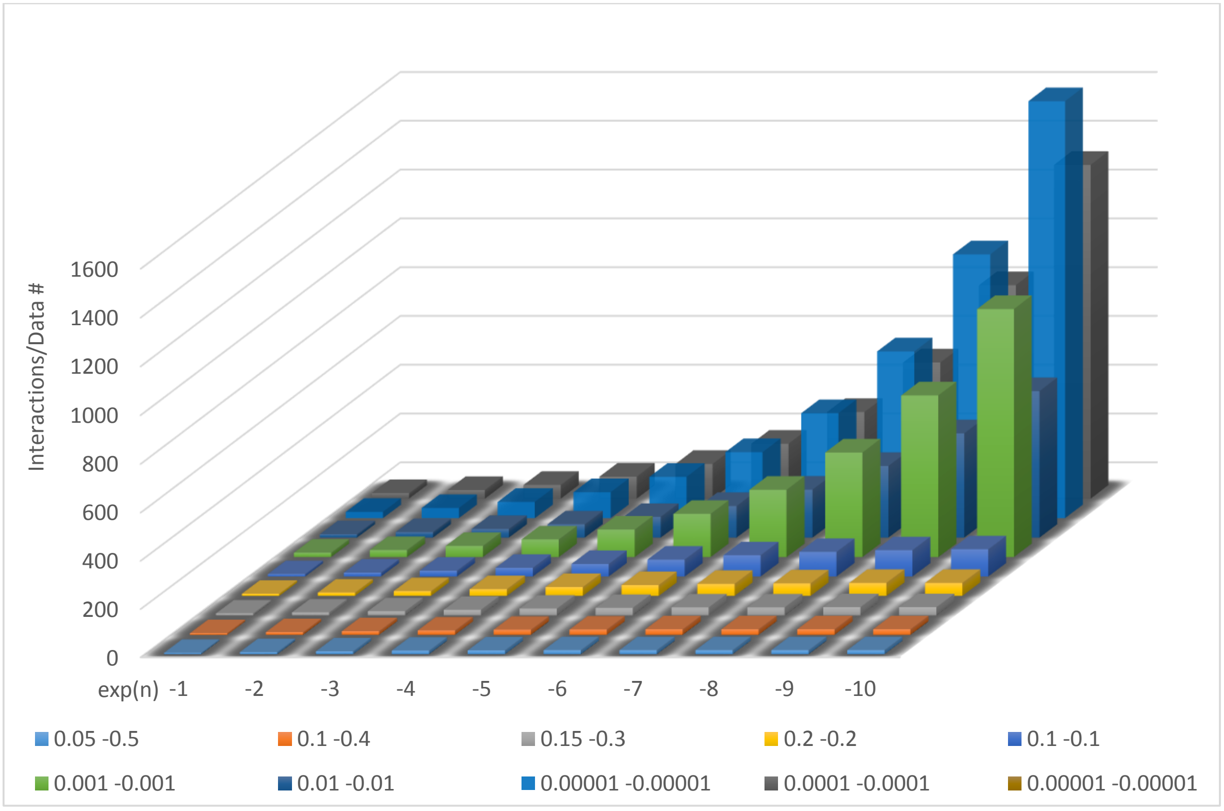


Figure 33. Trust-Convergence Interaction Number with Respect to e^{-n}.

The statement is evaluated with Figure 32 and Figure 33 is, the number of interactions needed for trust convergence toward -1 is proportional to the index of inverse exponential function; i.e., the higher the index, the higher the number of interactions.

3.4.5.5. Reduced Recursion Step Trust Evolves Faster Than Non-Reduced Recursion Step Trust

Figure 34 illustrates the trust-value evolution comparison between reduced recursion steps versus non-reduced recursion steps. It is evident from the figure that the reduced recursion step trust evolves faster than the non-reduced recursion step trust. In short, the recursion step reduces the number of interactions needed to catch con behavior. The trust at time t in this reduced step is dependent on trust parameters from time t-1.

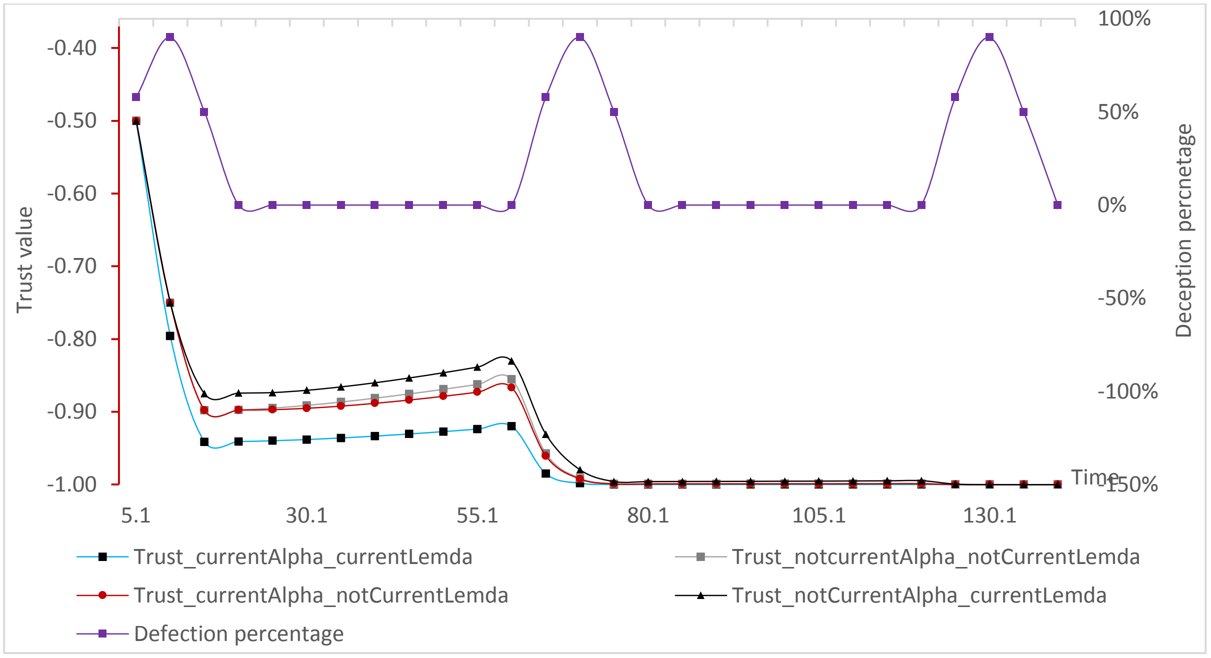


Figure 34. Calculation-Step Reduction.

3.4.5.6. The Number of Interactions Varies for Different Decimal Precision and Different λ

Figure 35 shows the number of interactions need for trust convergence to -1 for two different decimal precisions and two different data series (simulated data following two Poisson inter-arrival times or two λ values).

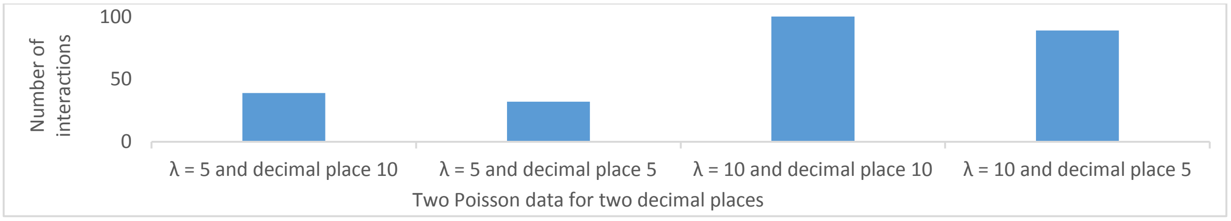


Figure 35. Number of Interactions for Two-Decimal Precision and Two Data Series.

Two principles are evaluated with Figure 35:

- For a fixed data set, the number of trust-convergence interactions is proportional to the decimal precision.
- For a fixed decimal precision when data follow a Poisson distribution, the number of trust-convergence interactions is proportional to λ.

3.5. Conclusion

A con-man trick can appear as a deception in cyberspace. A con-man-resistant trust algorithm can deal with this trick and can be used to find stable resources for service-oriented architecture, e.g., services in cyberspace. Hence, it is important to complete this algorithm by analyzing and studying its properties. This chapter empirically analyzed the properties of the con-man-resistant trust algorithm for cyberspace where the cloud services represented services from cyberspace. This property study completes the con-man-resistant trust algorithm. This completion is accomplished by extending the prior deception-detection procedure in cyberspace. The results presented in this chapter establish the correctness and effectiveness of the derived principles as well as the proposed enhancement of the mathematical scheme.

CHAPTER 4. PERFORMANCE-FOCUSED CON-MAN-RESISTANT TRUST ALGORITHM

4.1. Introduction

A con-man-resistant trust algorithm diminishes or enhances the trust value depending upon the defection-and-cooperation interaction patterns. The earlier works concerning the con-man behavior's trust did not vary the defection magnitude. However, the defection magnitude greatly affects a cloud-consumer victim. A defection is a performance degradation, and cooperation is the expected performance level for a cloud service. This performance-degradation magnitude varies; i.e., a performance fluctuation happens [118][119][120].

For example, consider an expected completion time for a certain web-service call of 5 milliseconds. This web service, hosted on the cloud, had a 6-millisecond response time once and a 1,000-millisecond response time another time. Although the response times degrade, or have a performance lapse, 1,000 milliseconds is worse than 6 milliseconds. Hence, the performance-degradation magnitude needs to be considered when designing the con-man-resistant trust algorithm.

The work in this chapter contributes by addressing this issue and proposes a performance-focused extension to the con-man-resistant trust algorithm's trust-updating scheme. According to this extension, the trust-decrement amount also depends upon the performance degradation or defection amount. Analysis reveals that the relationship between the decrement amount of trust and performance-degradation magnitude follows an exponential growth. Depending upon the type of cloud service, this exponential growth function varies. This work applied three growth functions. The application domain, such as in the cyber-attack prone cloud services or power-aware cloud services, of the functions is also considered. The experimental studies compare these three methods with the prior works' method by using cloud-service data from previous work [4]. The results show that this modification is promising. Result analysis reveals the principles that this performance extension follows. The cloud-service simulation data from Chapter 1 are used to demonstrate the extension's correctness and effectiveness.

The remainder of this chapter is organized as follows: Section 4.2 describes the problem. Section 4.3 presents the chapter's Contribution. Section 4.4 explains why the relationship between performance degradation and the corresponding penalty follows an exponential growth. Three exponential growth functions are used. These functions are utilized for defection or performance-degradation magnitude mapping in the punishment or penalty range. Section 4.5 describes two exponential growth functions, discrete and compound interest law, for the con-man-resistant trust algorithm. Section 6 describes the third exponential growth function, the logistic growth function. Section 4.7 illustrates the configuration of the simulator which was used for the simulation. Section 4.8 presents the Simulation Results that validated the correctness of the con-man-resistant trust algorithm's extension. Section 4.9 concludes this chapter and suggests future research.

4.2. Problem Statement

A con-man trick appears in cyberspace as a deception around the cloud services from the service provider. The con-man-resistant trust algorithm continuously monitors the cloud service's performance to detect this con-man behavior.

A cloud service is identified as a con when a lapse in the service performance repeats by an amount that is greater than the consumer can accept. Like the repetition, performance with a degraded magnitude is another service-performance attribute that affects the consumer's satisfaction. The problem statement includes the dynamism of the performance-degradation magnitude into the con-man-resistant trust algorithm in such a way that the trust evolution also depends upon both the frequency and the magnitude of the cloud service's performance degradation.

4.3. Contribution

The chapter's contribution includes the cloud service's performance-degradation magnitude being added to the con-man-resistant trust algorithm by following various models. The algorithm's mathematical scheme tracks the performance degradation magnitude. This revision evaluates the con behavior in proportion to the performance degradation. This proportional effect follows an exponential growth.

Three alternative exponential growth functions are evaluated. Each function is applicable for a specific cloud service. These functions are compared with each other, and the properties which affect the con-man-resistant trust algorithm are analyzed, resulting in a set of principles. Also, the relationship between performance-degradation magnitudes and the con behavior's detection speed are unlocked, resulting in a principle. The result represents the correctness and effectiveness of the principles.

4.4. Exponential Growth Function as Relationship Between Defection and Penalty

Table 15 and Table 16 show trust-updating schemes from earlier con-man-resistant trust algorithm research [3][118][119].

Table 15. Trust-Value Dependencies.

T	Cooperation	Defection
> 0	$T' = T + \alpha(1 - T)$	$T' = \dfrac{T + \beta}{1 - \min(\lvert T\rvert, \lvert\beta\rvert)}$
< 0	$T' = \dfrac{T + \alpha}{1 - \min(\lvert T\rvert, \lvert\alpha\rvert)}$	$T' = T + \beta(1 - T)$
= 0	α	β

T represents the current interaction's trust, and T′ presents an updated trust value when a defection or cooperation happens. It is clear from the equations in the both tables that trust parameters are dependent on the α, β, and C values.

Table 16. Trust-Parameter Dependencies.

Cooperation	Defection
$\alpha = \min(\alpha + \gamma_C\,(\alpha_0 - \alpha), \alpha_0)$	$\alpha = \alpha \times (1 - \lvert\beta\rvert)$
	$\beta = \beta - \gamma_D \times (1 + \beta)$
$\gamma_C = 1 - \lvert\beta\rvert$	$\gamma_D = C \times \lvert T\rvert$

Reward parameter α and punishment, or penalty, parameter β have initial values of α_0 and β_0, respectively. C (= e^{-1}), α_0, and β_0 were defined by earlier work [1][2][3].

If defection happens, then the reward decreases (α decreases), and the punishment or penalty increases (β decreases or |β| increases). These increases (Δα) and decreases (Δβ) are dependent on C (Table 16) along with their own previous values.

As described earlier, the defection amount varies. The con-man-resistant trust diminish in this work is proposed to be proportionate to the defection amount. Formally, the greater the defection, the more the Δβ is reduced relative to Δα. The increased punishment, or penalty, is achieved by mapping the performance degradation or defection amount into a punishment or penalty range: (0, C], where C is the constant function of mapping with value e^{-1}.

The performance-degradation or defection amount can be normalized into the range [0%,100%]. The assumed defection amounts are s_1, s_2, and s_3 with corresponding punishments of P_{s1}, P_{s2}, and P_{s2}, respectively. Here, $0\% < s1 < s2 < s3 \leq 100\%$. When all other con-man-resistant trust algorithm parameters are unchanged, then this incremental penalty means the following relationship between the defection amount and the corresponding punishment amount, as illustrated in Figure 36.

- $0 \leq Ps1 < Ps2 < Ps3 \leq C$,

  resulting in $0 \leq |\Delta\beta s1| < |\Delta\beta s2| < |\Delta\beta s3| \leq C$

  and $0 \leq |\Delta\, \alpha s3| < |\Delta\, \alpha s2| < |\Delta\, \alpha s1| \leq C$.

- $Ps2 - Ps1 < Ps3 - Ps2$ s,

  where $s2 - s1 \leq s3 - s2$.

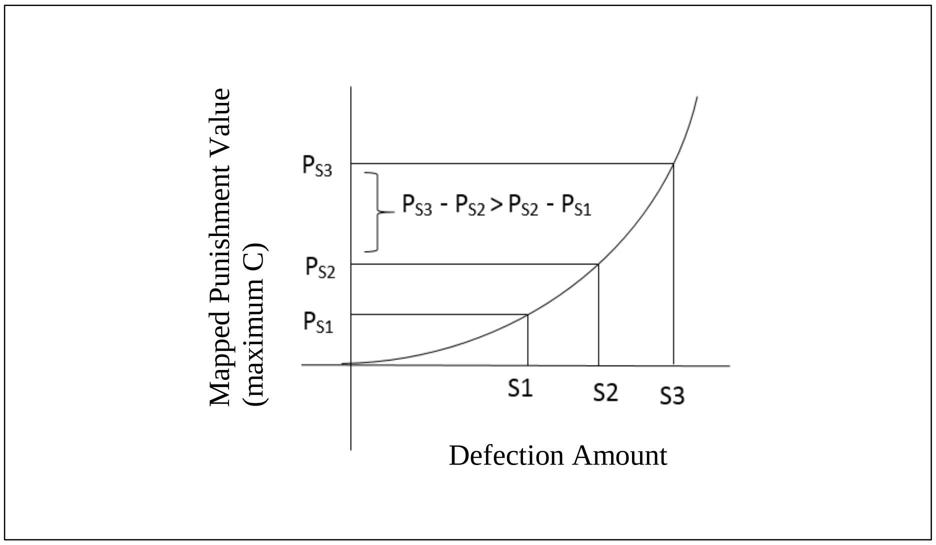


Figure 36. Punishment Increases with a Defection-Amount Increment.

Hence, the greater the defection amount, the greater the punishment. The relationship between defection and the corresponding penalty increment naturally follows an exponential growth function or a polynomial function. However, the exponential growth function is more rapid than polynomial growth [121]. Hence, the exponential growth function is applied to map the performance degradation's magnitude or the defection amount in the penalty range. Applying the exponential growth function to map intense or mild performance issues into higher or lower penalties is not new. The exponential growth function is presented as a suitable function for applications where an intense situation is penalized more than a mild situation, e.g., deceptive delays [122]. The server's delayed response for a requested service is a defense mechanism against a denial-of-service attack. The delay amount can be calculated using the exponential function. The service's expected response time, or processing time, can be used as input for the exponential function. The exponential function's output is how delayed the response for the requested service is. A longer response time refers to a service that needs more computational resources. This request represents an intense situation, i.e., a high possibility for a denial-of-service attack.

Exponential growth is a common phenomenon in the cloud-computing domain. For example, the cloud-computing data centers' growth follows exponential growth [123][124]. The disc storage amount per dollar has increased over time (from 1990 to 2012) [125]. The change follows exponential growth. The growth of data stored in Amazon storage service S3 follows exponential growth [126]. The electronic data traffic's increment follows exponential growth (between 2010 and 2020) [127]. However, these cloud-computing-domain examples might be the result of computing-performance growth characteristics that follow exponential growth [128]. In this chapter, three exponential growth functions are used to accomplish this mapping. The growth functions are discrete compounding (discrete compound-interest law), continuous compounding (continuous compound-interest law), and logistic growth.

4.5. Compound-Interest Law

4.5.1. Compound-Interest Law and Applications

The compound-interest law is defined in [129] as follows: “the rate of change of some quantity is proportional to that quantity itself” (p. 353). This law is phenomenal in nature and has been studied for a long time.

For example, the compound-interest law’s presence in nature was presented in 1919 [129]. Examples are plant growth, the rate at which a hotter body loses heat or cools down, the relationship of air pressure and height above sea level (The higher the elevation is, the lower the air pressure is.), any chemical reaction’s velocity, and the growth of cucumbers.

Another example is the relationship between temperature and a mature butterfly’s size. This relationship follows the compound-interest law [130]. Knowledge growth follows the compound-interest law; i.e., knowledge about the current time is a result of a fixed, proportionate increment of the previous knowledge [131].

Another example is evolutionary stability theory. This theory can interpret the compound-interest law in biology [132]. The plant germinability theories, e.g., seed-bank dynamics, follow the compound-interest law [132]. The number of British banks, known as the “bank population,” has declined since 1810. This decline follows a negative compound-interest law [133].

Moore’s law is combined with the compound-interest law (Compound-Moore’s law) to calculate the depreciation of a cloud resource [134]. This depreciation cost is used to calculate the cloud-service provider’s investment in a duration. This calculation helps the cloud-service provider to make smart business decisions before entering a contract with the consumer. The duration can be divided into the number of years.

Every year, the cloud-service provider can buy a resource. Each subsequent year, the cost of buying a new resource with the same configuration declines due to the depreciation calculated by Compound-Moore's law. However, the service provider may charge the same price to the consumer every year. Over time, the cloud-service provider's investment declines, but revenue from the consumer remains the same. When the investment is deducted from the revenue for the investment's duration, the net profit can be calculated. Therefore, if the investment amount and profit are known, the expected revenue can be calculated by the cloud-service provider. The provider can set the consumer's price for a cloud service by splitting the expected revenue for the contract's duration with the consumer.

4.5.2. Compound-Interest Law for the Con-Man-Resistant Trust Algorithm

The compound-interest law is suitable for specific cloud services, e.g., cyber-attack prone services. With such services, the greater the service-performance degradation, the lower the consumer's trust. Hence, the compound-interest law is chosen to calculate the proposed proportional penalty. On the compound-interest law's debit side, the longer the time left to pay the debt, the larger the balance to be paid is. Hence, the punishment is analogous to this debt side with the defection amount representing to the time that elapses while paying the compound interest.

The defection magnitude can have a continuous value. Hence, a continuous compound-interest law is applied to model the corresponding punishment. There are situations where the defection magnitude can represent intensity levels. Examples include the cloud-service performance's degradation magnitudes. This magnitude can be a continuous value (e.g., 90% performance degradation) or discrete values (e.g., very high, high, medium, low, and very low performance degradation) [64].

To make the presented work adaptable for this latter case, a discrete compound-interest law is used. Moreover, to assign these discrete levels to a defection magnitude, different and disjoint ranges are assigned to the various levels.

With the compound-interest law, interest is paid for each interest period [135]. The interest period is the time when the interest value changes. This period can be finite or infinitesimally small. When a finite interest period is used, the compound-interest law is called discrete compounding. When an infinitesimally small period is used, the compound-interest law is called continuous compounding.

The compound-interest law is given as follows [135]:

$$P_t = \begin{cases} P_0 \times e^{rt}, for\ continuous\ compounding \\ P_0 \times (1+r)^t, for\ discrete\ compounding \end{cases}$$

where P_0 = starting balance

P_t = Balance needs to pay at year t

r = yearly interest rate

t = current time limit

T = Maximum time limit or number of years when the loan needs to pay.

Because this function is exponential, the payments are incremental; i.e., for any time {t, t'} ϵ (0, T] and t < t', the respective payments P_t and $P_{t'}$ are related as $P_t < P_{t'}$. Hence, $\frac{P_{t\prime}}{P_t}$ ϵ (0, 1]. P_t represents the payment that needs to be made after t years and within T years. Hence, P_T outputs the function's maximum value. However, if we want to use the same formula for an exponential-growth-driven penalty calculation, then its output must be mapped into [P_0, C]. This mapping means that P_T must be set to C. The compound-interest law's core is an exponential growth function. This growth function is derived from the following mathematical deduction.

For discrete compounding,

$$\frac{P_t}{P_T} = \frac{P_0 \times (1+r)^t}{P_0 \times (1+r)^T}$$

results in

$$\frac{P_t}{P_T} = (1+r)^{(t-T)}.$$

For discrete compounding,

$$\frac{P_t}{P_T} = \frac{P_0 \times e^{rt}}{P_0 \times e^{rT}}$$

results in

$$\frac{P_t}{P_T} = e^{r\times(t-T)}.$$

For discrete compounding, t corresponds to distinct defection levels. For example, if the performance-degradation magnitude's intensity scale is considered, then the levels are very low, low, medium, high, and very high. Hence, t ϵ {1, 2, 3, 4, 5}, with T = 5, and the intervals between levels is $\frac{100\%}{T}$ (20% for T = 5). Level 0 always means no performance degradation; hence, it is not in the defection levels' example. Also, a 100% performance degradation, or defection magnitude, is assigned a defection level of T. The discrete defection level is calculated as follows from performance-degradation or defection magnitude (0%, 100%]:

$$\text{Defection level} = \begin{cases} \lceil Defection\ amount\ in\ percentile\ \in (0,1] \times Number\ \ of\ defection\ levels \rceil \\ \left\lceil \dfrac{SLA\ violation\ percentile\ \in (0,100] \times Number\ \ of\ defection\ levels}{100} \right\rceil \end{cases}$$

Similarly, for continuous compounding, t corresponds to a value from the range (0,100], with T = 100. Therefore, it is clear what the values for t, T, and P_T can be. The other parameters which need an explanation for this penalty analysis are r and P_0. A formula for r can be derived by simplifying the compound-interest formula mentioned earlier.

$$r = \begin{cases} \frac{1}{T} \times \ln \frac{P_T}{P_0}, for\ continuous\ compounding \\ e^{\frac{\ln \frac{P_T}{P_0}}{T}} - 1, for\ discrete\ compounding \end{cases}$$

Here, r is dependent on the P_T-to-P_0 ratio. Both r and P_0 can be explained by defining this ratio because the P_T value is given. This ratio, r, determines the exponential curve. Choosing this ratio is a challenge. Figure 37 and Figure 38 show punishment values with respect to performance-degradation or defection magnitudes for different ratios, i.e., $\frac{P_T}{P_0}$ values. If this ratio is 100, then for the compound interest during early t values or with a low performance-degradation magnitude, Δt, the corresponding ΔP is very low. Δt is the change in time, defection magnitude, or performance-degradation magnitude; ΔP is the corresponding change in the penalty amount. A ratio value of 5 has almost linear growth, and a value of 10 is close to symmetrical.

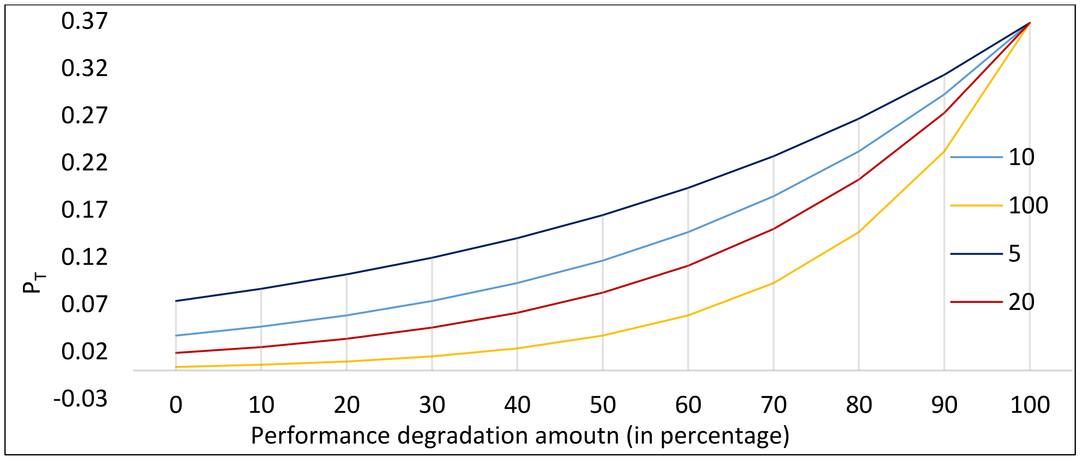


Figure 37. The Continuous Compound-Interest Formula, $P_0 \times e^{rt}$, is Used with r and $P_t = e^{-1}$ and t as the Defection Magnitude.

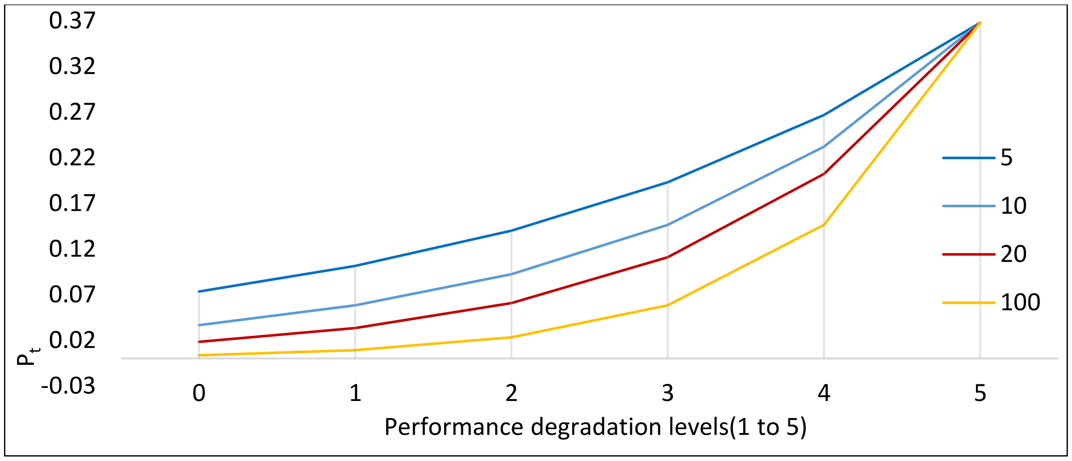


Figure 38. The Discrete Compound-Interest Formula, $P_0 \times (1 + r)^t$, is Used with r and $P_T = e^{-1}$ and t as the Defection Magnitude.

4.6. Logistic Growth Function for the Con-Man Algorithm

4.6.1. Logistic Growth Function and Its Applications

The logistic function is an S-shaped function that slowly increases before a certain value and slowly decreases after that value. The logistic function's growth is slower at the dependent variable's initial and end values. The sigmoid function is a type of logistic function [136]. When A = B = C =1 in a logistic function's formula, then the resulting function is a sigmoid function [137]. The output of the sigmoid function is € [0,1]. The sigmoid function is mentioned here because, in a cloud-computing domain, the sigmoid form of the logistic function is widely applied.

The logistic growth function reflects a naturally occurring behavior. This function has numerous applications; the most popular applications are in statistics and neural networks. This function is the core of the logistic-regression methodology in statistics and has been used to map real values in a bounded interval [138]. The logistic sigmoid function is also applied as the hidden neurons' activation function in a neural network [139]. The logistic growth function's application details for other fields are outside the scope of this chapter because only cloud-computing applications are of interest in this chapter.

In the cloud-computing domain, this logistic function is applied widely. For example, the relationship between the average response time experienced by all users of a cloud service and the number of users follows a sigmoid function [140]. The number of users (x-axis) and the number of request time outs (y-axis) also follow the sigmoid function [140].

The sigmoid function is applied to normalize a cloud service's QoS values (e.g., into [0,1]) [141]. The sigmoid function is also applied as a utility function for the cloud services [142]. This sigmoid function represents the cloud consumer's satisfaction based on the QoS and allocated resources. Inputs for this sigmoid function can be the cloud-service response time and the cost of that service. The same function is reused to define the cloud user's behavior [82]. The consumer-satisfaction issue, i.e., dissatisfaction, is presented in this chapter by the penalty amount that the cloud-service provider pays for a QoS violation. Hence, the logistic function, the parent of the sigmoid function, is used in this chapter to represent the relationship between the cloud-service provider's penalty and the QoS.

In another earlier application, an algorithm was designed to make decisions during an automated web-based SLA negotiation [143]. The decision function was dynamically chosen from an exponential, polynomial, and sigmoid function. The sigmoid function reached the negotiation between the cloud consumer and service provider quicker than the other two functions.

The sigmoid function is used in the cloud-computing domain to include the dynamism of an SLA violation [25]. The sigmoid function represents the relationship between the service provider's penalty and the associated SLA violation. The penalty is dependent on the violation's duration. The sigmoid function is applied to reduce the cloud-service provider's loss when the SLA violation is above a predefined limit. Another function, called SLA satisfaction, which was used in [25], restricts the upper limit of the service provider's penalty. The sigmoid function applied in [25] is also used in [144] to represent the cloud-service provider's penalty percentage with respect to the SLA violation ratio.

The sigmoid function's application in this chapter is inspired by the relationship between the SLA violation's duration and the service provider's penalty. The sigmoid function applied in [25], representing the penalty and SLA violation's duration, is a time-series function because the penalty amount is dependent on the time length. The difference between the sigmoid function applied in [25] and this chapter's sigmoid function is time length (or duration) and the function's parameters. The SLA-violation magnitude or QoS degradation magnitude is used in this chapter's sigmoid function instead of time length. The sigmoid function's parameters are domain specific. Hence, the parameters, e.g., A, B, are different for this chapter's sigmoid function.

In this chapter, the logistic function's application in the cloud service's penalty mapping is inspired by the logistic function's application described in [25], [122], and [142] .

- First, an exponential function is suitable for applications where an intense situation is penalized more than a mild situation [122].
- Second, a special kind of logistic function, i.e., sigmoid function, is used to represent the relationship between the cloud consumer's satisfaction and the QoS [142].
- Third, a logistic function, i.e., a sigmoid function, represents the SLA violation and the corresponding penalty for the service provider's [25].

- Fourth, logistic function limits the upper bound of the service provider’s penalty[25]. By limiting the upper bound, this function reduces the cloud-service provider’s loss when the SLA violation is above a predefined limit.

4.6.2. Logistic Growth Function for the Con-Man-Resistant Trust Algorithm

In cloud-computing research, there is research dealing with the tradeoff between resource management and performance, e.g., between power consumption and performance [22][145][146]. These studies allow a certain performance-degradation magnitude to save energy. If the performance-degradation mapping into the penalty range follows a compound-interest function for such applications, then the power-aware methods result in a high penalty for performance degradation. An asymptotic analysis can bound or limit the punishment curve’s high end.

The logistic growth function’s symmetric, S-shaped curve can serve this purpose. In statistics, the logistic growth function has been applied to map real values into a bounded interval [138]. This function reflects a naturally occurring behavior. Hence, for situations where a certain performance-degradation magnitude is accepted (e.g., energy-aware cloud algorithms), the logistic growth function can be used to represent the relationship between performance-degradation magnitude and cloud service provider’s penalty.

In addition, this function has a legacy in the cloud-computing domain. This function was used with similar applications in the past. Examples are representing the relationship between an SLA violation and the corresponding service-provider penalty when limiting the penalties’ upper bound [25] as well as representing the relationship between consumer satisfaction and QoS [142].

The logistic growth function has the following form:

$$Pt = \frac{C}{1 + Ae^{-B\times t}}$$

P_t has a maximum possible value, C. This formula's output is ϵ (0, C). In this equation, when t = 0, the logistic growth function's initial value needs to be the minimum possible penalty.

$$P_0 = \frac{1}{10^{\#\, of\, decimal\, place}} = \frac{1}{10^5}\ ; [for\ 5\ decimal\ place]$$

However, for t = 0,

$$\frac{1}{10^{\#\, of\, decimal\, place}} = P0 = \frac{C}{1 + Ae^0} = \frac{C}{1 + A}$$

results in,

$$A = C \times 10^{\#\, of\, decimal\, place} - 1.$$

For 5 decimal places and $C = e^{-1}$,

$$A = C \times 10^5 - 1 = 36786.94.$$

B's value would be positive because this logistic growth function always needs to be increasing. The logistic growth function's inflection point is $\left(\frac{lnA}{B}, \frac{C}{2}\right)$. If we want X% punishment to be at $\frac{C}{2}$, then

$$\frac{lnA}{B} = X\% = \frac{X}{100}$$

$$B = \frac{100}{X} \times lnA.$$

For 5 decimal points and when the performance-degradation or defection magnitude is 50%,

$$B = \frac{100}{50} \times lnA = 2 \times lnA = 21.02580.$$

Figure 39 shows this logistic growth function. This function for the con-man-resistant trust algorithm does not result in exact upper- and lower-bound values; hence, its output is (0, C) rather than [0, C]. Again, the initial t = 0 is never applied with the con-man algorithm because there is no penalty for 0% performance degradation or no defection. Hence, t ϵ (0%, 100%].

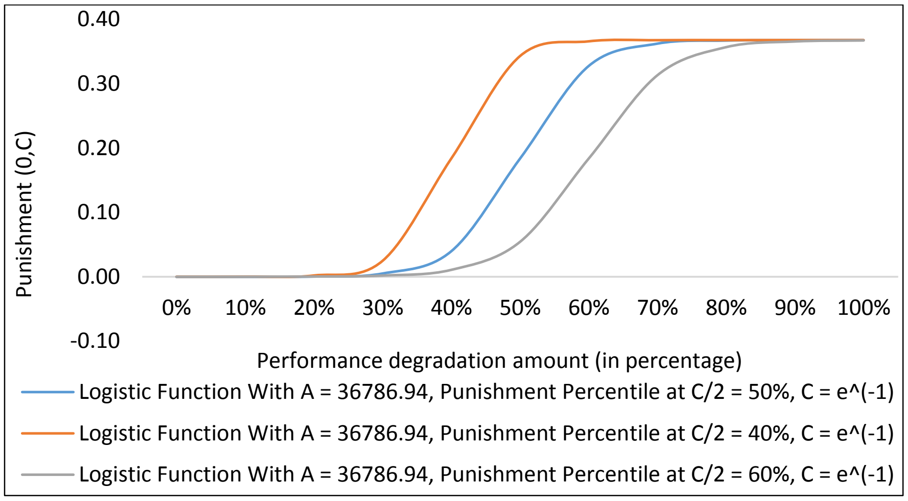


Figure 39. Logistic Growth Function.

The minimum possible value for t is given by following equation.

$$t_{min} = \frac{1}{10^{\#\, of\ decimal\ place}} = \frac{1}{10^5}\ ; [for\ 5\ decimal\ places]$$

Figure 40 shows the utilized exponential growth function’s comparison.

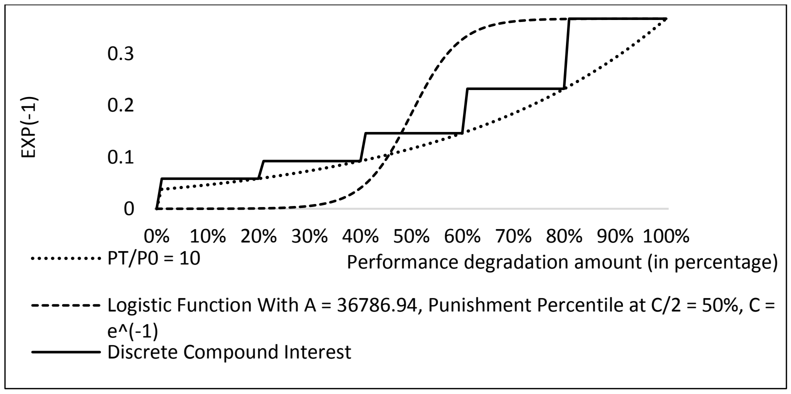


Figure 40. Comparison Between Two Compound-Interest Functions and the Logistic Growth Function for Mapping.

4.7. Simulation Configuration

In prior chapters, two cloud services were simulated on top of the Cloudsim simulator [30][31]. These services are software as a service (SAAS) and infrastructure as a service (IAAS). The appearance of performance degradation follows Poisson arrivals or exponential inter-arrival time. The performance-degradation magnitude follows the normal distribution for IAAS. The con-man-resistant trust algorithm is applied to these data, monitoring the cloud-service providers' con behavior [4]. Data from previous chapters are used for this research to evaluate the effectiveness of the con-man-resistant trust algorithm's proposed extension in this chapter.

4.8. Simulation Results

4.8.1. Discrete Trust: Trust Values from the Same Defection-Level Range Are Identical

Defection, or performance-degradation magnitudes, is mapped into defection levels when a discrete compound-interest formula is used as the mapping function. Figure 41 shows the design's correctness.

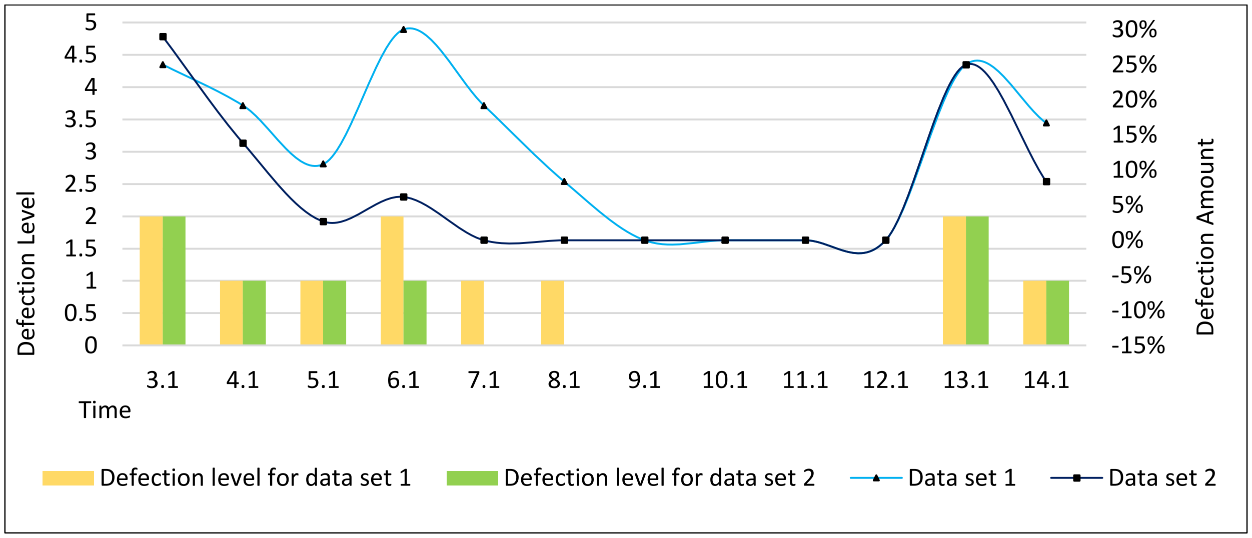


Figure 41. For the Same Defection-Magnitude Range, the Defection Level is Identical for Two Different Data Sets.

The figure shows that different defection magnitudes from a certain range correspond to a specific defection level. Here, the bar's heights show a particular defection level. The figure also shows that, when defection magnitudes are not in the same range, they magnitudes falls into different defection levels. For example, at time 6.1 milliseconds, the defection magnitudes are mapped into two distinct defection levels.

Figure 42 further shows this correctness. It shows that a defection magnitude of 1% versus 20% using the discrete compound-interest formula results in identical trust values. The trust values are identical because both 1% and 20% defections belong to same defection level of 1.

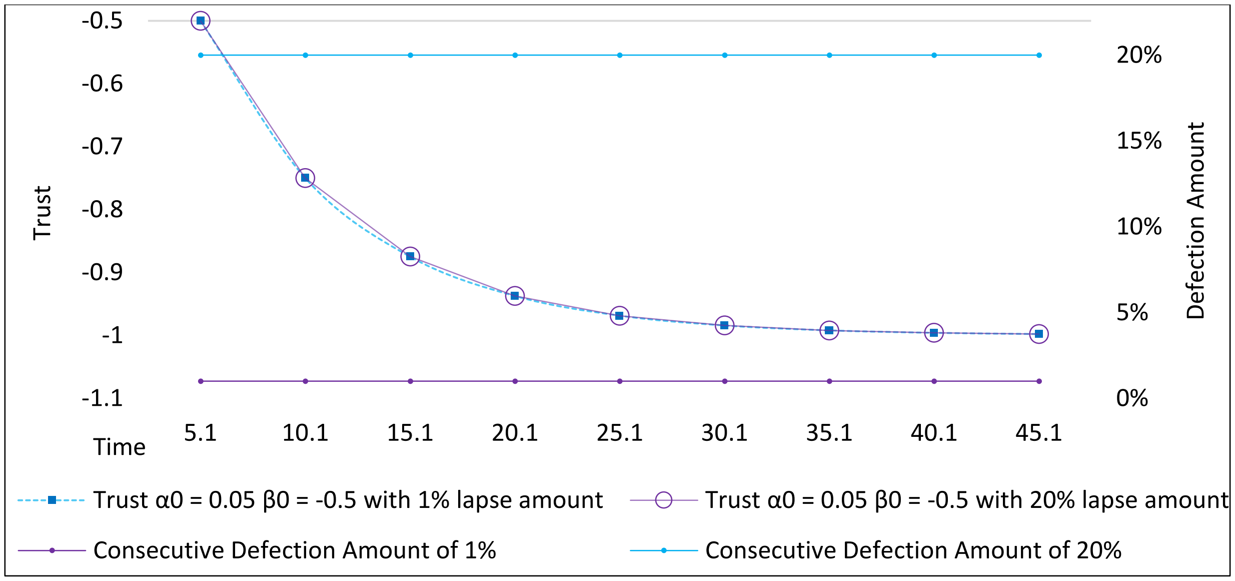


Figure 42. The Trust Values' Evolution with the Discrete Compound-Interest Formula Are Identical for the Same Level.

4.8.2. Principle 1: The Number of Interactions is Inversely Proportional to the Service's Performance-Degradation Magnitude or Defection for Non-Constant Mapping

Figure 43 shows untrustworthiness for a service provider's detection when defection or performance degradation happens consecutively. Untrustworthiness is detected when the trust value has its minimum value (-1). Here, $\alpha_0 = 0.01$; $\beta_0 = -0.1$; there are 5 decimal places; and the continuous compound-interest formula is applied. Principle 1 is derived from this figure:

Principle 1: When non-constant mapping is used, the higher the performance degradation or defection magnitude (percentage) that appears consecutively, the faster the trust convergence is toward untrustworthiness (faster detection of con behavior) and vice versa.

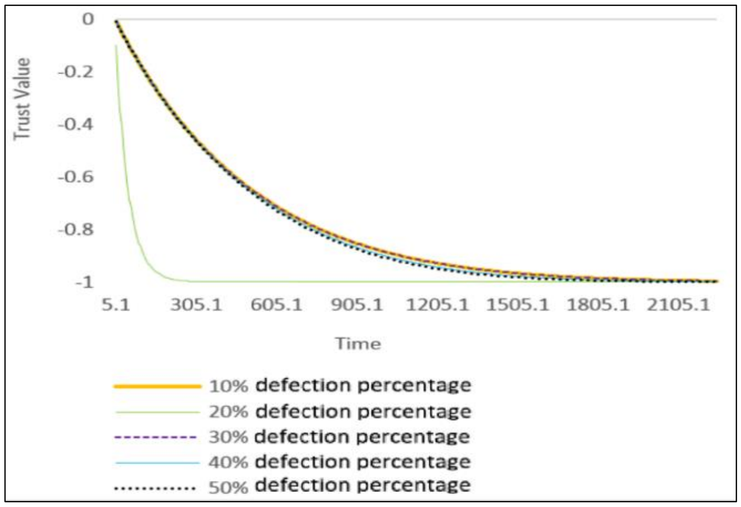


Figure 43. Con-Behavior Detection for Different Consecutive-Lapse Magnitudes.

4.8.3. Principles 2 to 5 from Comparing the Performance-Degradation Magnitude's Mapping Functions

Figure 44 shows how quickly the con behavior is detected, i.e., how quickly the trust value converges to -1 (untrustworthiness) for 4 different defection-magnitude mapping functions when a defection magnitude of 50% appears consecutively. The con-man resistant trust algorithm's parameter configuration is decimal place = 5 and $C = e^{-1}$.

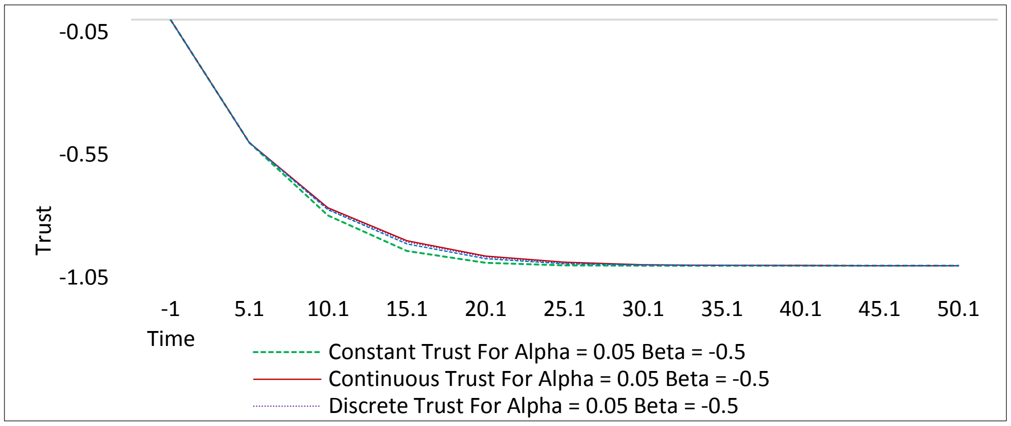


Figure 44. Trust-Convergence Comparison for Different Defection-Magnitude Mapping Functions When the Defections Appear Consecutively.

Figure 45 shows a similar comparison with parameters $\alpha_0 = 0.05$, $\beta_0 = -0.5$, decimal place = 5. Here, the con-man-attack scenario follows Poisson inter-arrival time ($1/\lambda = 5$).

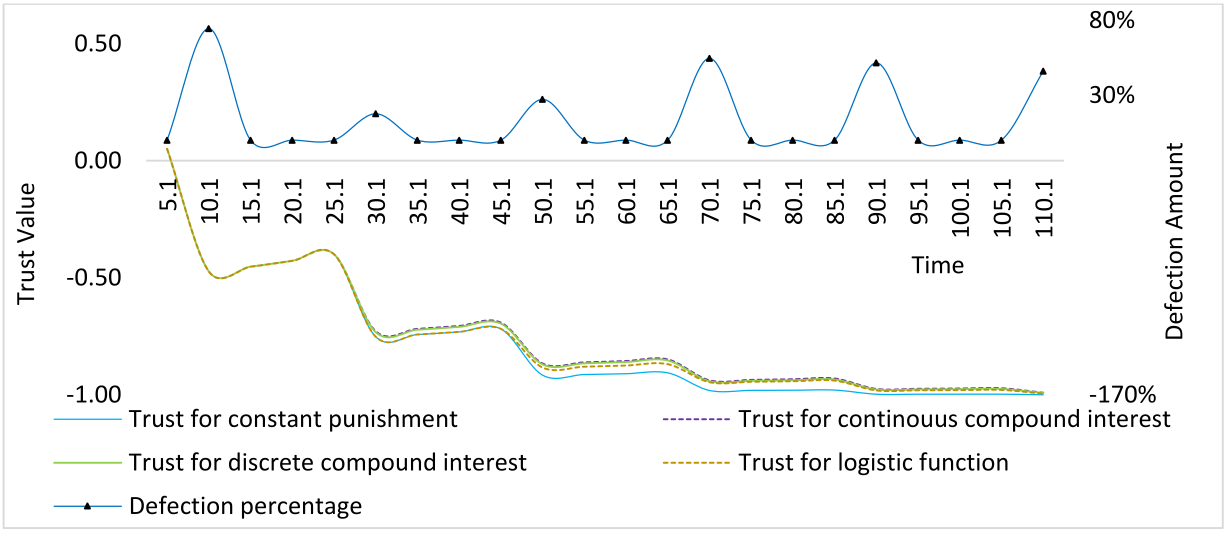


Figure 45. Con-Behavior Detection Time Comparison for Performance-Degradation Magnitude Mapping Functions.

The following principles are derived from Figure 44 and Figure 45:

- Principle 2: The constant function to map the defection magnitude is the fastest for the con-behavior detection. The detection takes fewer interactions for the trust value to converge to -1.
- Principle 3: The logistic function can detect the con behavior faster.
- Principle 4: The discrete-interest law function detects the con behavior slowly.
- Principle 5: The continuous-interest law function is the slowest for the con-behavior detection.

4.9. Conclusion and Future Work

In this chapter, a performance-degradation focused, con-man-resistant trust algorithm's extension for cloud services is developed and evaluated. Three exponential growth functions are added as extensions. These growth functions' properties are studied, and the functions are compared with each other concerning the con behavior's detection. Each growth function is applicable for individual cloud services. In the studied compound-interest function, the performance-degradation magnitude resembles the time length for paying a debt. Hence, the corresponding interest mapped into the penalty value, and, therefore, into the con-behavior amount, is also a natural phenomenon. The logistic growth curve is used to bind the penalty curve's high end and is applicable to certain research scenarios. Additionally, the degradation magnitude's effect on trust convergence is presented as a principle. The principles aid in decision making when the con-man-resistant trust algorithm is configured for a specific domain. These derived principles are this chapter's contribution.

For future work, additional functions which reflect the relationship between the performance-degradation magnitude and the con-behavior amount can be introduced. Like performance-degradation or defection-magnitude mapping in the punishment range, cooperation or the expected performance magnitude can be mapped in the reward range. For example, if a web service hosted in the cloud has a 3-millisecond expected response time, but the actual response time is 1 millisecond, then the service has good performance. This goodness amount (3 milliseconds -1 millisecond = 2 millisecond) can be normalized and can then be mapped in the reward range.

CHAPTER 5. LEARNING ABILITY OF THE CON-MAN-RESISTANT TRUST ALGORITHM

5.1. Introduction

Recently, the global average cost incurred as a result of cybercrimes has dramatically increased. Hence, organizations are investing to deal with “cybercrimes.” This investment is more on “cybercrime-detection” activity. The machine-learning technologies for cybercrime detection are cost effective. The con-man attack is a deception in cyberspace, hence falling under the umbrella of cybercrime. The con-man-resistant trust algorithm detects this cybercrime. The machine-learning algorithm can be applied to tune the con-man-resistant trust algorithm in such a way that con-man-resistant trust algorithm gives its optimal performance.

The con-man deception’s victim is the consumer. As a result, the algorithm that detects this deception needs to be consumer centric. The algorithm needs to learn the consumer’s perspective about the deception. A learning ability is added to the con-man-resistant trust algorithm in this chapter. This ability is added by empirically studying a machine-learning algorithm for the con-man-resistant trust algorithm to learns about a particular consumer’s perspective, about a service from cyberspace, from that the consumer’s historical data using that service. The study’s results are a set of principles that the con-man-resistant trust algorithm follows and a machine-learning algorithm that learns the con-man-resistant trust algorithm’s parameter settings for a particular consumer. Finding the parameter configuration for a particular consumer is both an algorithm-configuration and parameter-tuning problem.

The assumption for this algorithm-configuration problem’s solution is that there is a collection of a particular cloud service’s time-series data (historical data) for a particular consumer. The consumer identified the data as either trustworthy or not trustworthy. Only trustworthy data are selected from the data collection. The con-man-resistant algorithm is tuned for the trustworthy data. Hence, a specific parameter configuration is determined for all the trustworthy data. This tuning in a set of parameter configurations (for the collection of cloud-service data). Only one configuration is chosen from this set of parameter configuration. The specific parameter configuration is determined for the trustworthy data by iteration. The con-man-resistant algorithm’s performance is evaluated at each iteration. If the performance test fails, then the search space shrinks, and the iteration continues with the new, smaller search space. This iteration continues until the search space is the smallest that it can be.

The remainder of this chapter is organized as follows: Section 5.2 describes the problem. Section 5.3 presents the Contribution of this chapter. Section 5.4 describes the Related Concepts, e.g., algorithm-configuration problem, parameter versus hyperparameter, machine-learning details, and cross-validation details. Section 5.5 is a summary of the overall learning process that uses a machine-learning algorithm. Section 5.6 presents the simulated cloud-resource configuration. Section 5.7 is the solution for sub problem 1, i.e., selecting the con-man-resistant trust algorithm’s bias parameter that needs tuning. Section 5.8 presents sub problem 2, i.e., analysis of the selected parameter’s properties. Section 5.9 describes sub problem, 3 i.e., training, testing, and applying the machine-learning algorithm. Section 5.2 elaborates the con-man-resistant trust algorithm’s limitations in learning. Finally, Section 5.3 ends this chapter with suggested future work.

5.2. Problem Statement

The con-man-resistant trust algorithm monitors the cloud-service provider’s con-man-attack deception. The con-man-attack deception is an oscillation of good and bad service. The deception-detection pace depends on the trust model’s parameter tuning, known as an algorithm optimization problem. A specific consumer is susceptible to a certain pattern of oscillation. Hence, for different consumers, this optimal parameter setting varies. Manual adjustment of the parameters by trial and error during the algorithm's optimization, with the goal of parameter tuning for a particular consumer, is time consuming. A machine-learning algorithm can learn this parameter configuration by reading the consumer's historical-usage data for that cloud service. Hence, the problem statement is optimizing the con-man-resistant trust algorithm for a particular consumer.

This problem can be divided into the following sub problems:

- Sub Problem 1: Selecting the con-man-resistant trust algorithm parameters to be tuned.
- Sub Problem 2: Studying the selected parameters' properties.
- Sub Problem 3: Training, testing, and applying the Machine Learning version of the con-man-resistant trust algorithm.

5.3. Contribution

The efficiency of the con-man-resistant trust algorithm depends on how well it can reflect the consumer's perspective of trust for the service. The reflection further depends upon how aligned the algorithm's parameters are with the consumer's trust perspective. Hence, the efficiency of the con-man-resistant trust algorithm can be enhanced by tuning, either manually or automatically, its parameter values. This area of study is known as algorithm configuration. Manual parameter tuning means to follow an exhaustive trial-and-error process. Automatically means following some kind of machine-learning algorithm in a way that the algorithm can learn from experiences and can tune itself. However, there is no such technique for the con-man-resistant trust algorithm. The chapter's goal is to optimize the con-man-resistant trust algorithm for a particular consumer.

More elaborately, the goal is to propose a machine-learning algorithm where the con-man-resistant trust algorithm learns about the consumer's choice from that consumer's historical use of any service. Upon learning, this con-man-resistant trust algorithm can classify real-time data as trustworthy or untrustworthy from that consumer's perspective. In short, the learning enables this algorithm to discern the consumer's perspective.

The chapter's contribution is to unlock additional properties for the con-man-resistant trust algorithm and to introduce the algorithm's learning capability to optimize the algorithm for a particular consumer.

5.4. Related Concepts

5.4.1. Algorithm-Configuration Problem

An algorithm-configuration or parameter-tuning problem is finding the set of parameters for which the algorithm provides its optimal output. Generally, the optimal performance for any algorithm is effective in terms of performance or cost. Hence, from the definition, there is no doubt why the tuning or configuration is desired. The algorithm-configuration, or parameter-tuning, problem's definitions from different authors are presented in Table 17.

Table 17. Algorithm-Configuration, or Parameter-Tuning, Problem Definitions by Authors.

Definition by	Definition
[147]	"Given • An algorithm, A, with parameters, $p_1 .... p_k$, that affect its behavior. • A space, C, of configurations (i.e., parameter settings), where each configuration, $c \in C$, specifies values for A's parameters such that A's behavior for a given problem instance is completely specified. • A set of problem instances, I. • A performance metric, m, that measures the performance of A on instance set I for a given configuration, c. Find a configuration, $c * \in C$, that results in optimal performance of A on I according to metric m" (p. 38).
[148]	"Given an algorithm, a set of parameters for the algorithm, and a set of input data, find the parameter values under which the algorithm achieves the best-possible performance on the input data" (p. 269).

The algorithm-configuration problem can be defined in terms of set theory. According to [147], the algorithm's configuration problem is finding a set of configurations such that the algorithm results in an optimal performance for certain inputs and for these configurations. The optimal performance is measured using a pre-defined metric.

The con-man-resistant trust algorithm's parameter-configuration problem can be defined in terms of the definition given by [147] which is presented in Table 18. According to [147], "algorithm configuration" and "parameter tuning" are interchangeable; hence in this dissertation, they both refer to the same concept.

Table 18. Mapping the Con-Man-Resistant Trust Algorithm's Terminologies with the Algorithm Configuration's Definitions.

Algorithm-Configuration Definition Terms	Con-Man-Resistant Trust Algorithm's Terms
A is an algorithm where its parameters affect its behavior. These parameters are $p_1 \ldots p_K$.	A is the con-man-resistant trust algorithm. The α_0-to-β_0 ratio and α_0, β_0 (Parameter C is excluded, and the reason is explained later in this chapter.) are the parameters which affect A's behavior.
C is the set of parameter configurations or settings. Algorithm A's behavior, or output, is well known for each element of C, or for each such configuration.	C is the combinations of the parameter values (α_0-to-β_0 ratio; α_0, β_0; and e^{-1}), where each parameter belongs to its own range.
I is the set of problem instances.	I corresponds to the set of cloud-service-quality historical data for a certain service and for a certain consumer that is identified as trustworthy by the consumer.
The performance of A on input data set I for an element of C, or for a configuration, is measured by the performance metric, m.	m is trustworthiness.

The algorithm configuration's definition that is presented in [147] is restated in this dissertation as follows, for the con-man-resistant trust algorithm:

> The algorithm configuration problem, in terms of con man resistant trust algorithm, is finding the values of α_0 to β_0 ratio, α_0, β_0 in such a way that A outputs trustworthy for all elements of I i.e., for all trusted historical data.

The algorithm configuration, is achievable in both a manual and an automated way. However, the automatic configuration, or tuning, is always preferred [149]. The automated iterative process to find an algorithm's optimal parameter setting is described in [150]. At each iteration, the algorithm's performance is evaluated. If the evaluation process fails the performance test, then the iteration continues until the evaluation passes. The work presented in this chapter follows this concept of iteration where a parameter space, or value range, shrinks at each iteration until a range is found where the condition is met. For future work, the iterations can be parallelized [151]. One drawback with algorithm configuration is it needs numerous run of the algorithm [151]. Hence, this process is computationally very costly [151]. This cost sensitivity proves that, the smaller number of algorithm runs need for the con-man-resistant trust algorithm configuration, is expected. Table 19 presents automated algorithm-configuration examples where machine-learning algorithms are applied for automated algorithm configuration.

Table 19. Automatic Algorithm-Configuration Examples.

Algorithm-Configuration Example	Description
Optimizing MaltOptimizer	MaltOptimizer is an implemented machine learning that is based on the parameter-tuning algorithm [152]. This learning algorithm can tune some specific parameters of an algorithm automatically in such a way that the tuned configurations result in optimal outputs for the algorithm. The MaltParser algorithm's parameters are tuned using MaltOptimizer. The input to this MaltOptimizer is a training set, and the output is an optimized configuration of MaltParser.
Tuning support-vector-machine parameters	An example of algorithm configuration is the use of an iterative algorithm to tune the pattern-recognition support vector machine's (SVM's) parameter [153]. The learning algorithm minimizes the SVM's generalization error over specific parameters. This minimization is done by minimizing the error in the SVM classifier's generalization ability for a set of particular SVM parameters, i.e., by minimizing the generalization error with a gradient-descent algorithm.
Evolutionary computing as machine learning to tune parameters	The choice function's weights, or parameters, are tuned by genetic-algorithm and particle-swarm optimization [154]. Both algorithms train the choice function at the sampling phase where the search continues until a criterion, e.g., number of steps, is met. Hence, these algorithms are applied as machine-learning techniques to tune the choice function [155].
Population-based algorithms as machine learning to tune parameters	Two machine-learning algorithms, random forest and genetic algorithm, are applied to build a model that is applied for automatic algorithm configuration [156]. This model works by predicting the parameter space's high-performance regions.
Bayesian optimization in parameter tuning	Bayesian optimization is applied for automated parameter tuning of a Markov Chain Monte Carlo, without human intervention [149].
Machine-learning-based genetic algorithm and particle-swarm optimization	A machine-learning-based genetic algorithm and particle-swarm optimization are applied to find the optimum model parameter for a probabilistic neural-network classifier [157].
Random forests in algorithm configuration	An algorithm called sequential model-based optimization is applied to solve algorithm-configuration problems [158]. This algorithm is based on a machine-learning technique called random forests.
Supervised learning for algorithm configuration	F-Race is a supervised learning technique that is applied for a highly parameterized algorithm's configuration [159].
Machine learning to tune metaheuristics	F-Race is applied to tune metaheuristics [160]. A machine-learning-based genetic algorithm is also applied to tune metaheuristics. A genetic algorithm and simulated annealing are applied to tune the support vector machine's parameter automatically [161].

5.4.2. Hyperparameter versus Parameter

Model hyperparameter is an algorithm parameter that cannot be estimated from data, e.g., from the training phase of any learning algorithm. On the other hand, the model parameter can be estimated from data. The configuration of model parameter affects the algorithm's efficiency and correctness.

Table 19 presented the model parameter tuning, or optimization, using machine-learning algorithms. Similarly, machine-learning algorithms, e.g., support vector machine, can be applied to hyperparameter optimization[150][162][163]. In a con-man-resistant trust algorithm, the ratio of α_0 to β_0 is hyperparameter, where C, α_0, and β_0 are model parameters.

5.4.3. Machine Learning

The concept of machine learning was first introduced in 1959 [164]. This concept was first applied in the game of checkers to determine whether a machine-learning algorithm outperforms a human. An IBM 704 computer was used. The algorithm learned how to improve its game playing when rules and information about the game were given.

Machine-learning algorithms can deal with black-box optimization problems [165]. The goal of machine learning is to add learning capability to an algorithm. An algorithm can count its experiences as its input and can then train itself. Machine-learning algorithms can unlock the dependencies between a system's input and output efficiencies [93]. The training part of machine learning makes the algorithm adaptive to its computational environment because the algorithm is learning from the feedback. Machine-learning algorithms are applicable when human intervention to solve a problem is expensive [166]. These algorithms can discover the complex nature of the input data [93]. However, there are problem areas where machine-learning algorithms should not be used, e.g., areas where a completely error-free outcome is expected [166].

The machine-learning algorithms that possess characteristics which are significantly different from other optimization problems are presented in [165]. The two characteristics mentioned are as follows: the algorithms can run in parallel, and the run time for an individual algorithm can vary. These two characteristics enable the algorithm to run with a large amount of computational resources, e.g., using the cloud resources. Deploying the algorithm into multiple cloud resources reduces the cost. In the reminder of this subsection, machine-learning algorithm types, machine-learning technology in cybercrime detection, and machine-learning applications are presented.

5.4.3.1. Unsupervised Learning

A machine-learning algorithm can be unsupervised. Unsupervised learning is described in [167] as “learning without a teacher” (p. 486). The goal of unsupervised learning is defined in [167] as inferring the characteristics of a given observation collection where the correct characteristics for each observation are not given. Unsupervised learning infers hidden properties that are common in a given collection of the unlabeled data. These data possess the same characteristics and are clustered together [32]. These clusters are the output for the learning process. There is no way to evaluate the accuracy of an unsupervised learning algorithm due to the lack of data labels. This learning method discovers the patterns, properties, or structures from the observations. Unsupervised learning can be applied in the classification process but in an indirect way. This classification process clusters the data items, or observations, based on the common properties. Afterword, this learning algorithm assigns class labels to these clusters. This classification is suggested in [168] to classify internet traffic data using the unsupervised learning. This process is applied in the intrusion detection, e.g., in determining if a user belongs to an authorized users' cluster or not [32].

Data mining is subfield of machine learning that deals with data analysis and falls under the umbrella of unsupervised learning. Pattern recognition is an example of an unsupervised-learning and data-mining application.

5.4.3.2. Supervised Learning

A machine-learning algorithm can be supervised. There are two steps in supervised learning: training and testing [169]. The algorithm is trained to learn the items' properties. The items can be considered as classes. The algorithm is then tested by to identify unknown items that have the features of the known items or classes. Hence, supervised learning is applied for classification purposes [170]. This learning ascertains the class labels from training and then maps these class labels to new items during the testing phase. Supervised machine-learning algorithms have been applied in classification problems to map known class labels into items where the labels are not known (for example, in internet traffic classification) [168][169]. A supervised learning algorithm with artificial neural networks is applied for anomaly detection in IAAS [171].

The supervised machine learning algorithms involve steps. In the first step, features of the internet traffic are defined (traffic-flow duration, packet lengths, etc.). In the second step, the machine-leaning algorithm is trained to co-relate these features with the existing, known traffic's class labels. In the third step, the algorithm is applied to internet traffic where the classes are not known. The third step's goal is to map the class labels from step 2 into step 3's traffic. The classifier's accuracy is evaluated by splitting the known, labeled data set into two: the training and test data sets [168]. The model is built using the training data set, and the classification accuracy is measured using the test data set.

5.4.3.3. Semi-Supervised Learning

A machine-learning algorithm is semi-supervised when the input is a collection of labeled data with many unlabeled data [172]. This learning algorithm is a subclass of the supervised-learning method. This method is useful when labeling the data is expensive.

5.4.3.4. Reinforcement Learning

Reinforcement learning is another machine-learning technique [173]. This learning tunes the underlying model’s actions in such a way to maximize a reward [174]. The difference between reinforcement learning and supervised learning is the pre-classified or pre-labeled input data. In terms of reinforcement learning, the difference is that the optimal output samples are not given [174].

5.4.3.5. Machine-Learning Technology in Cybercrime Detection

The global average cost incurred by cybercrime has dramatically increased recently [175]. A study surveyed 254 global companies’ investment in cybersecurity [175]. The companies’ cost for cybercrime was \$7.2 million in 2013, \$7.6 million in 2014, \$7.7 million in 2015, \$9.5 million in 2016, and \$11.7 million in 2017. Hence, organizations are investing to deal with cybercrimes. Among the investments, the companies are spending more on cybercrime detection, e.g., 35% for detection activities during 2017. Hence, the cybercrime detection is of great importance. According to the study presented in [175], information-technology security layers have six components: network layer, application layer, data layer, human layer, physical layer, and host layer. The organizations’ investments for cybersecurity had the largest increase from 2015 to 2017 at the application layer (4% increment) and the data layer (4% increment).

Hence, the security technologies for these layers shall be given a great importance. The machine-learning technologies for the cybercrime detection can reside on the application layer. The companies' investment to employ security-enabled technologies versus the return of investment is also calculated in [175]. The arithmetic reveals that the organizations have not widely accepted automation, orchestration, and machine-learning technology as their security solution. Hence, this technology is ranked the lowest among the enterprise-wide deployment technologies for security (nine technologies represented in [175]).

However, when the return on investment for a technology is calculated, the automation, orchestration, and machine-learning technology is ranked 3rd; i.e., the return on investment is 17.1%. This return-on-investment cost is calculated by dividing the gains from the investment by the cost of the individual technologies' application. The average return on the investment for all nine studied security technologies is 14.1%, which is lower than 17.1%. Hence, machine-learning technology for cybercrime detection can be considered when the cost savings come into play.

5.4.3.6. Machine-Learning Applications

Machine-learning algorithms have been applied for algorithm optimization [156]; computer vision, e.g., in corner detection [176]; pattern recognition [153][174][177]; data mining [170][178][179]; distributed artificial intelligent systems or multi-agent systems [180]; classification [153][181][182][183]; and many other diverse areas. Table 20 presents examples of the machine-learning applications.

Machine-learning classifiers have been successfully applied in security-concerned areas [184]. For example, game theory and machine-learning concepts, together, can take advantage of each other's methods to detect malicious and fraudulent behavior [184].

Table 20. Machine-Learning Applications.

Machine-Learning Application	Example
Parallel random forest algorithm in classification	A wide range of machine-learning algorithms are evaluated in [183] for classification. The authors concluded that the parallel random forest algorithm gave the best result.
Cyber-intrusion detection	Machine learning is applied in the cyber-intrusion-detection domain, e.g., unsupervised anomaly detection [32]. If a user belongs to an authorized-user cluster, then the user may not be an anomalous user.
Hybrid support-vector-machine-based decision tree	The support vector machine is a widely used supervised machine-learning tool in the classification domain. A hybrid support-vector-machine-based decision tree is a fast version of the support vector machine used for classification [177].
Internet-traffic classification	Two supervised machine-learning algorithms, Kiss and Abacus, are used as the behavior classifiers that categorize internet traffic based on the traffic's features [185]. These algorithms are studied and compared, focusing on the memory consumption and the computational cost.
Data mining	Machine learning is used to extract information from data [170]. A machine-learning algorithm can be used as the tool or technique to discover, explain, and unlock existing patterns in data. The technical side of data mining is the machine learning techniques [170].
Multi-agent systems	In multi-agent systems, machine learning is used to learn the agent's behavior for different scenarios [180]. As a machine-learning application on multi-agent systems, robotic soccer is used as a test bed [180].
Cloud security and attack classification	The machine-learning techniques are used in cloud security to monitor cyber-attacks on cloud resources [186]. The cyber-attack threats are identified by analyzing the virtual machine's performance data, e.g., CPU speed, disc-space usage, network data, and memory usage. Predefined attack classes help to classify the attacks using machine learning. These learning techniques can be provided as a third-party tool and can help the consumer to avoid losses from cyber-attacks. Among the employed techniques, the support vector machine has the best performance on the top threats' identification.
Netflix recommendation system	Netflix uses machine learning to predict a consumer's choice [187]. This machine-learning-based recommendation system first matches the consumer features with already-known consumer's feature. Based on known consumer choices, the technique predicts the current consumer's choice.
Walmart	Walmart bought data storage from Hewlett Packard to store the purchase records for all its customers [187]. The company is using machine-learning tools and techniques to analyze those data in order to improve business efficiency.
Supervised learning feed-forward network	Artificial neural networks, a machine-learning approach, are used for anomaly detection in IAAS with statistical pattern recognition [171]. A supervised learning algorithm is used with a feed-forward network.
Machine-learning forensics	Machine learning can be used for data mining. The algorithms can find interesting structures or information from large data sets. An example of such information mining can be risk discovery. The algorithms can analyze the pattern of criminal activities. This specific area of machine learning that deals with predicting criminal behavior and crimes is known as machine-learning forensics [188].
Intrusion detection: Cloud-based malware detection	A machine-learning algorithm is used to classify malicious code and botnet traffic automatically. This machine-learning algorithm has a significantly lower classification error during such intrusion detection with classification. This algorithm can be implemented in MapReduce and, hence, can run parallel to cloud resources, reducing cost. Therefore, the problem that this algorithm addressed is called cloud-based malware detection [189].
Botnet detection	Botnet is malicious software or compromised client software that resides on a client's computer. A machine-learning algorithm is used to extract the features of both malicious and non-malicious clients during the training phase [190]. Then, this trained algorithm is used on real-time network data to detect malicious activities.
Machine-learning vulnerability: Adversarial examples	Adversarial examples are the inputs for which a machine-learning algorithm gives an incorrect result. An attacker can replace the existing machine-learning model with a substitute model. The substitute model works perfectly as the original machine-learning algorithm for all inputs except the adversarial examples. Therefore, if a machine-learning algorithm is replaced with this malicious entity, then the algorithm can incur losses. The effect of a malicious substitute model, or entity, is demonstrated using the commercial machine-learning classifiers from Amazon and Google.
Malicious-code identification	Decision tree, naïve bayes, bayesian network, and artificial neural networks are used to learn the behavioral patterns of malicious code samples [139]. The learned algorithm can be used to identify malicious codes.
Modeling the nonlinear behavior of a cloud resource	Machine-learning techniques can be used to model the nonlinear behavior of a cloud resource, e.g., cloud data center, when trained with real-life workload traces [93]. The trace used was two years of records produced by DJM software that ran at the Los Alamos National Lab. Each job that ran on the lab's servers had 18 features; 6 features were used by the machine-learning algorithm: average CPU time, allocated processor number, memory usage, the job's run time, the job's wait time, and queue number. The wait time is considered as the latency and represented the disk's read-write wait time and network latency.
Big-data classification problem	The accumulated collection of internet traffic data became a big-data problem [191]. Hence, introducing machine-learning techniques to classify these traffic data with the goal of network-intrusion prediction is essentially a big-data classification problem. This prediction problem was resolved with big-data classification and by integrating multiple technologies together, e.g., Hadoop distributed file systems, cloud technologies, the representation learning technique, and support vector machines.

The goal of the work presented in this dissertation is to identify given time-series data as trustworthy or not trustworthy by using the concepts of both game theory and machine learning. The assumption is that the consumer of a specific cloud service classifies the service as trustworthy or not depending upon their satisfaction about the service. As described before, the con-man-resistant trust algorithm uses game theory to identify a cloud-service provider's oscillating service-performance issue.

The goal of the machine-learning algorithm implemented in this chapter is to find a parameter setting for the con-man resistant-trust algorithm so that the parameter value best describes these classes (form the consumer's given trustworthy and untrustworthy data). This parameter value reflects the consumer's perspective of trustworthy and not trustworthy. Hence, this problem is a classification problem. The algorithm is a supervised machine-learning algorithm because an algorithm needs to train itself with pre-classified data (trustworthy and non-trustworthy) to meet the goal. This supervised machine-learning algorithm trains itself by learning the con-man-resistant trust algorithm's parameter value and then uses this parameter setting to classify a cloud service's execution as trustworthy or not.

5.4.4. Cross-Validation for the Accuracy-Estimation Method and Parameter Tuning

A model is accurate when its output is correct. A model's accuracy estimation validates the model's correctness. A model can have the goal of classifying data. Example classes can be good or bad, trustable or not trustable, etc. The accuracy estimation is often used to validate such binary classifiers. The accuracy-estimation methods split the data set into two mutually exclusive sets: the training set and the test set. A classifier learns from the training set, and the accuracy is tested with this trained classifier's application on the test data set. According to [179], a classification algorithm trains itself from the training data set; hence, the training data set has indirect effects on the accuracy estimation. The test data set is used to see how well the trained algorithm is performing. The test data set directly affects the measurement for accuracy estimation because the measurement is taken against the test data set.

The importance of an accuracy estimation for a supervised learning algorithm with goals of both estimating the classification accuracy and selecting a classifier from a classifier list is described in [192]. The two most widely accepted methods to assess classifier accuracy are cross-validation and bootstraping [192]. Additional accuracy estimation methods, presented in [192], are holdout and random subsampling. For real-world data sets, cross-validation, e.g., tenfold cross-validation, provides the best output when the goal is to choose an accuracy-estimation method with low bias and low variance [192]. The leave-one-out method is very expensive with respect to the cross-validation, although it is not biased [161].

The repeated versions of cross-validation and bootstrapping are also compared in [193]. However, the result of the comparison is the opposite of the result presented in [192] because the findings presented in [193] consider a larger sample size. The cross-validation estimator outperforms with a smaller sample size; for a larger sample size, the bootstrapping method performs better.

The characteristics of the cross-validation and bootstrapping methods are also studied in [194]. Bootstrapping is biased for a small sample size. Both algorithms behave like an asymptotically optimal algorithm with a very large data size. Both techniques have a low mean-square error.

In a holdout method, two-thirds of the data are used for the training and one-third of the data for the testing. The method's issue is its high variance [192]. Dividing the set into two has a high performance bias for this method [192]. Another issue with this method is that the one-third of the data used during the test is a large number in real life, making the method inefficient [192]. Random subsampling is a form of the holdout method. This method is repeated a fixed number of times. The estimated accuracy is the mean of the accuracy estimate from each run. This estimator is good for a large sample size. The issue with this method is the bias. If a class is overrepresented by the training partition, then the other class suffers a misclassification [192].

In the k-fold cross-validation method, the data set is divided into k mutually exclusive subsets. The training and test phases are run k times. At each time, one particular subset is used as the test set, and the rest (k-1) of the subsets are the training set. This process is repeated until all k subsets are used as the test set. The k-fold cross-validation provides an accurate performance estimate [179]. The goal of any k-fold cross-validation is the replicability of results. Replicability is a measure of stability, meaning how consistent the statistical result is for samples from the same population [195]. The goal of the work presented in this dissertation is to validate a supervised learning-based binary-classification method to give the data trustworthy and not trustworthy labels. Both the consistency and the correctness of this binary classification are of equal importance. The k-fold cross-validation can also be used successfully in the performance estimation of a supervised learning algorithm [179].

For the cross-validation, $k = 10$ is a widely accepted value for the accuracy estimator [193]. This estimator is comparatively unbiased and has a high variance with a smaller k [193]. A higher k value results with a smaller test data set size, reducing the bias [194].

In a stratified, k-fold cross-validation, each fold contains the same proportion of every class label (for the classification problem). Each fold needs to be a good representation of the given data set. Rearranging the data can enable this property [179]. For a binary classification, 50% of the data from each fold shall represent one class label [179]. Hence, in the binary classification presented in this dissertation, the trustworthy class label represents 50% of the data.

Another special cross-validation case is the leave-one-out method where one subset is used as the test data and rest as the training data [194]. This process is repeated for N - 1 subsets, where N is the number of observations. This method is computationally expensive because it needs to run N times. When k = N, the k-fold cross-validation is a leave-one-out method.

In the bootstrap method, data are randomly selected from the data set N times with replacement. Replacement means that the same data can be selected more than once because the already-selected data are still in the selection pool. Bootstrapping has a smaller standard deviation, i.e., a low variance, for a small sample size [193]. However, the bootstrap method is computationally expensive with respect to cross-validation [193]. Bootstraping can also be used for testing the results' replicability, but it has some other drawbacks with respect to cross-validation. The 0.632 bootstrap is biased to a small sample size compared to the cross-validation method [194]. The training set is two-thirds, or 0.632, of the total data in the 0.632 bootstrap.

The work presented in this chapter has a small sample size; therefore, the 0.632 bootstrap is not a good choice with respect to the cross-validation. The cross-validation has three applications: performance estimation, model selection, and parameter tuning [179]. Cross-validation is used in this chapter to estimate the accuracy of the binary classifier.

The cross-validation application is recommended in [179] for the parameter-tuning application. The authors mentioned that, when the labeled, or pre-classified, data are limited, then cross-validation is the best option. They recommended splitting the training data from the supervised learning algorithm to perform the cross-validation. A part of the supervised learning algorithm's training data needs to be the training data for the cross-validation, and the rest shall be the test data. The split of the supervised learning algorithm's training data is repeated k-1 times because the k-fold cross-validation is used. The classifier in this dissertation work is a supervised learning algorithm that is trained using labeled data. The training data are split to apply the k-fold cross-validation.

5.5. Learning-Process Summary

There is no previous work on the con-man trust domain that incorporated historical data. All previous studies focused on real-time data. However, historical data are utilized in this chapter. The idea of trust generation from historical records came from [11]. However, the learning idea is empirical. Trust from real-time data $A_f\left(P_i, P_j\right)$ can be addressed as a short-term trust factor, whereas a long-term trust factor is trust from historical data, $H_f\left(P_i, P_j\right)$ [11]. According to the terms of Xiaoyong's work, this long-term trust is a learned behavior from historical data using the machine-learning algorithm [60]. Applying this machine-learning algorithm on simulated, real-time data is short-term trust factor. Hence, the application of machine learning can follow Xiaoyong's notation, where P_i is the consumer and P_j is the service provider.

$$D = \left(H_f\left(P_i, P_j\right), A_f\left(P_i, P_j\right)\right)$$

The con-man-resistant trust algorithm's parameter values are tuned from this learning. These parameter settings are fed into the con-man-resistant trust algorithm (as the initial parameters) to monitor unstable cloud services in real time.

5.6. Simulation Configuration

Supervised learning is used in this empirical study. The learning needs pre-classified data [196]. Therefore, the assumption is that a collection of historical records is identified as either trustworthy or untrustworthy. Rather than choosing a mix of both types, only the trustworthy records are chosen.

For one preselected con-man algorithm parameter, β_0, a certain β_0 value identifies one individual record as trustworthy. The maximum β_0 (minimum $|\beta_0|$) is chosen from these β_0s. This β_0 is used to determine whether the real-time data for that service provider and consumer pair (P_i, P_j) are trustworthy.

The simulated historical records contain 144 fixed-interval simulation times, from 0.1 millisecond to 143.1 milliseconds. The Poisson inter-arrival time used is 5 milliseconds, and the number of requests at each Poisson time is between 5 and 10 (normal distribution). This number of 5 to 10 is determined from the trial-and-error process. For example, with a number between 0 and 5 (normal distribution), the resource shortage does not happen, resulting in no QoS degradation. The QoS violation happens for more than two consecutive data when the number is between 10 and 15.

5.7. Sub Problem 1: Selecting Bias Parameters to be Tuned

The con-man-resistant trust algorithm's trust evolution over time depends upon certain algorithm parameters which were described, in detail, with their properties in previous chapters. A particular amount for these parameters is chosen in this chapter, and then, the parameters are tuned using a machine-learning algorithm. In this subsection, a particular parameter is determined.

Table 21 and Table 22 are recap from previous chapters. Table 21 shows the trust-value dependencies, and Table 22 is the trust-parameter dependencies. Each time a consumer encounters a cloud service, the current trust value, T′, is updated (Table 21). In all the cases for Table 21, the current trust depends upon previous trust, T, and parameters α and β.

Table 21. Trust-Value Dependencies.

T	Cooperation	Defection
> 0	$T' = T + \alpha(1 - T)$	$T' = \frac{T + \beta}{1 - min\,(\lvert T\rvert, \lvert\beta\rvert)}$
< 0	$T' = \frac{T + \alpha}{1 - min\,(\lvert T\rvert, \lvert\alpha\rvert)}$	$T' = T + \beta(1 - T)$
= 0	$T' = \alpha$	$T' = \beta$

The α and β values change each time the consumer receives the cloud service. Table 22 illustrates that this change depends upon initial values α_0 and β_0; previous α, β, and C; and previous trust value, T.

Table 22. Trust-Parameter Dependencies.

Cooperation	Defection
$\alpha' = \min(\alpha + \gamma_C(\alpha_0 - \alpha), \alpha_0)$	$\alpha' = \alpha \times (1 - \lvert\beta\rvert)$
	$\beta' = \beta - \gamma_D \times (1 + \beta)$
$\gamma'_C = 1 - \lvert\beta\rvert$	$\gamma'_D = C \times \lvert T\rvert$

The constants here are initial values α_0, β_0, and C. In previous con-man-resistant trust algorithm work, the ratio between α_0 and β_0, with a value of C, is recommended [1][2]. Also, these recommended values are successfully utilized in the con-man-resistant trust algorithm's application for stable node selection in a smart electrical grid [3].

Hence, the ratio and C values are excluded. The only parameters left are α_0 and β_0. The α_0s and β_0s are derivable from each other because their ratio is defined or recommended. Either of these values can be considered as the selected variable to be tuned. In this chapter, β_0 is selected as the parameter to be tuned using the machine-learning algorithm. Therefore, the selection process is complete.

There are multiple values for the α_0-to-β_0 ratio recommended by previous work. Multiple C values are recommended in a previous chapter of this dissertation [4]. It can be argued that there is a relationship between the consumer type and the recommended parameter values. In this chapter, it is hypothesized that there can be two consumer categories: flexible and conservative (not flexible).

For the flexible consumer type, the consumer allows frequent QoS degradation. This consumer will identify the frequent QoS degradation-appearing data as trustworthy. The SAAS data in the presented work, with a very small inter-arrival time of QoS degradation appearance (λ, e.g., 3, 4, 5, 6, 7) are an example. The consumer has more trust reward for cooperation (good service) and less trust punishment for the defection (bad service). Hence, to learn from or to train the machine-learning algorithm with this consumer's identified trustworthy data, the parameter values can be flexible. For example, C = e^{-n}, with n more than 1 and the α_0-to-β_0 ratio as 1:3 (indicates a smaller punishment value with respect to the reward value). This scenario is tested by using different values for these parameters.

A conservative consumer can be defined as one who does not like frequent QoS degradation. The QoS degradation's appearance in the data needs to be sparse. This property of the empirical-analysis data used in this work is presented by a large inter-arrival time (e.g., λ = 50, 60, 70, etc.) of QoS degradation. This consumer has more trust punishment for the defection than trust reward for cooperation. Hence, to train the machine-learning algorithm using this consumer's identified trustworthy data, the parameter values need to be conservative. The conservative values imply indicates a larger punishment value for a defection with respect to the reward value for cooperation. The conservative values are C = e^{-1} and the α_0-to-β_0 ratio as 1:10. This scenario is also tested by utilizing different values for the parameters.

5.8. Sub Problem 2: Selected Parameter's Property Study

The selection of a particular parameter for the con-man-resistant trust algorithm is presented in the previous subsection. The parameter's characteristics and the learning process for this parameter value are studied in this section. The assumption is that there is a list of data which were classified by the consumer as trustworthy and not trustworthy. Trustworthy means that the data did not have con behavior (from the consumer's perspective). In this chapter, the relationship between the selected parameter values for trustworthy and not-trustworthy data is unlocked. For certain data, if the β_0 for which these data are identified as trustworthy can be derived, then the β_0 for which these data are not trustworthy can also be obtained with the earlier β_0. In the second step, the algorithm to find such β_0 for certain data identified as trustworthy is described.

5.8.1. Selecting Historically Trustworthy and Untrustworthy Data

In the con-man-resistant trust algorithm, for a continuous defection-cooperation cycle, higher β_0 values yield an earlier convergence rate of trust toward -1 (untrustworthy value). This relationship between β_0 values and convergence rate implies that, if a record is untrustworthy, there must be a β_0 for which the con-man trust model identifies the record as trustworthy. This relationship also implies that, if the record is trustworthy, there must be a β_0' for which the con-man trust model identifies this record as untrustworthy. Hence, the first two principles are equivalent. They result in the third principle.

- Principle 1: If the historical record is identified as trustworthy, there is a β_0', maximum trustworthy β_0, for which any $\beta_0 \le \beta_0' \xrightarrow{\text{yields}}$ Trustworthy with any $\beta_0'' > \beta_0' \xrightarrow{\text{yields}}$ Untrustworthy.

- Principle 2: If the historical record is identified as untrustworthy, there is a β_0', minimum untrustworthy β_0, for which any $\beta_0'' \geq \beta_0' \xrightarrow{\text{yields}}$ Untrustworthy with any $\beta_0 < \beta_0' \xrightarrow{\text{yields}}$ Trustworthy.
- Principle 3: For any data with fixed length, there is a β_0' for which any $\beta_0'' \geq \beta_0'$ $\xrightarrow{\text{yields}}$ Untrustworthy with any $\beta_0 < \beta_0' \xrightarrow{\text{yields}}$ Trustworthy. The difference between β_0' and the maximum value of such β_0s and is $10^{-decimal\ place}$.
- Summary of principles 1-3: In short, *above* $\beta_0' \xrightarrow{\text{yields}}$ Untrustworthy and *below* β_0' $\xrightarrow{\text{yields}}$ Trustworthy.

The above principle 3 is a conclusion from this project's results. The difference between $\beta_0$and β_0' , which are representative of the trustworthy and untrustworthy parameter values, respectively, in this project, is always at the least significant digit (for any constant decimal place throughout the project). This difference is always $10^{-decimal\ place}$. Figure 46 shows this scenario.

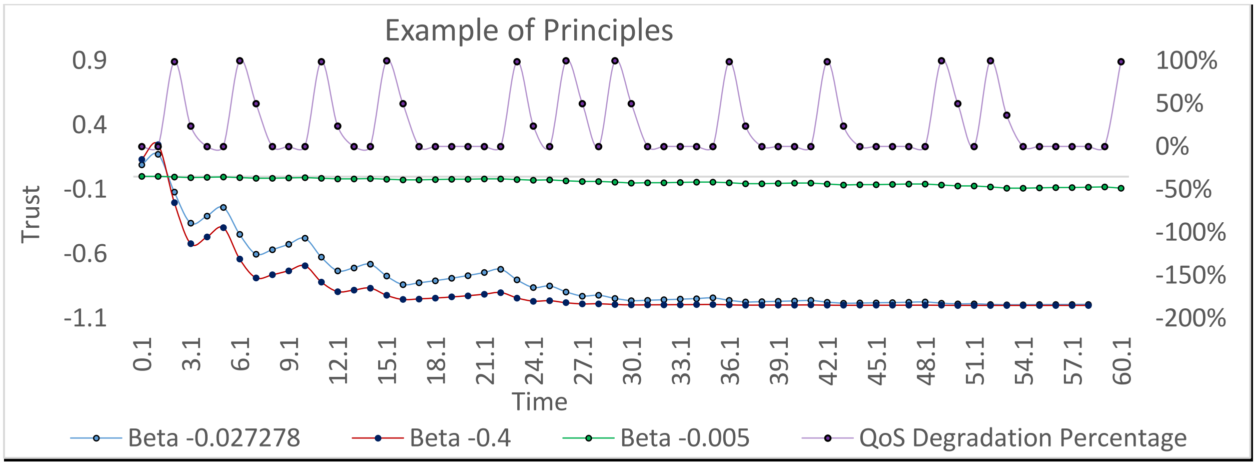


Figure 46. Maximum Trustworthy β_0 and Minimum Untrustworthy β_0.

For example, for 5 decimal places, if a trustworthy β_0 = -0.49999, then the untrustworthy β_0 = -0.50000. As another example, for a trustworthy β_0 = -0.27278, any β'_0 = - 0.4 > β_0 results in trust converging to -1 or untrustworthiness. Similarly, for an untrustworthy β_0 = -0.27278 (trust converges to -1 for this β_0.), any β'_0 = - 0.005 < β_0 results in trustworthiness (trust d not converge to -1).

It is clear that the trustworthy parameter can be derived from the untrustworthy one and vice versa. Hence, considering only the trustworthy records or only the untrustworthy records, rather than a mix of them, produces interchangeable results. Only the trustworthy records are chosen in this project's training phase to find β_0 value(s) for a certain service provider.

5.8.2. Finding Trustworthy β_0

The β_0 can take a value from $[-(1 - 10^{-\text{decimal place}}, -10^{-\text{decimal place}})]$ for any one historical record. For example, for 5 decimal places, $\beta_0 \in$ [- 0.99999, - 0.00001]. The maximum number of possible distinct values that β_0 can take in this β_0 range is $m = 10^{\text{decimal place}} - 2$. The developed algorithm to find trustworthy β_0, i.e., the maximum β_0 for which the historical record is trustworthy, has a complexity of $\theta(\lceil \log_2 m \rceil)$. The θ is 17 for decimal place = 5. Improvement can be made in future work by dividing the problem to find trustworthy β_0 into 10 sub problems rather than 2 (current application). This split reduces the complexity into $\theta(\lceil \log_{10} m \rceil)$. The reason for using 10 is that m is dependent on decimal precision where the base is 10.

As described in previous sections, the {α_0, β_0} pair is selected for the training phase of the machine-learning algorithm. Also, the α_0-to-β_0 ratios used are 1:3 and 1:10. Hence, only the trustworthy β_0 value (the β_0 for which certain service data are identified as trustworthy) needs to be determined from the training records. This β_0 is also dependent on the length of an individual record, so the β_0 for a longer record could be different. Algorithm 3 determines the β_0 for a certain historical record.

```
Algorithm  Find Maximum Trustworthy $\beta_0$
  procedure Find_Maximum_Trust_Worthy_$\beta_0$
      $Minimum|\beta_0| \leftarrow -10^{-Decimalplace}$
      $Maximum|\beta_0| \leftarrow -1 + 10^{-Decimalplace}$
      $Numberofiterations \leftarrow 0$
      if Trust converges for Minimum $|\beta_0|$ then
          $Numberofiterations++$
          return 0
      else if Trust does not converge for Maximum $|\beta_0|$ then
          $Numberofiterations++$
          return -1
      else
          return ITERATE($Minimum|\beta_0|$,$Maximum|\beta_0|$)
  procedure ITERATE($Minimum|\beta_0|$,$Maximum|\beta_0|$)
      $Mid\beta_0 \leftarrow \frac{Minimum|\beta_0|+Maximum|\beta_0|}{2}$
      if Trust converges for $Mid\beta_0$ then
          $Numberofiterations++$
          $Maximum|\beta_0| \leftarrow Mid\beta_0$
      else
          $Minimum|\beta_0| \leftarrow Mid\beta_0$
      if $Maximum|\beta_0| - Minimum|\beta_0| \leq 10^{-Decimalplace}$ then
          return $-Minimum|\beta_0|$
      else
          return ITERATE ($Minimum|\beta_0|$,$Maximum|\beta_0|$)
```

Algorithm 3. Finding the Minimum Trustworthy β_0 for a Certain Historical Record.

Figure 47 shows that, for β_0 = -0.00265, trust converges to -1 and that, for β_0 = - 0.00264, it does not converge until the simulation time of 3015.10.The number of times that the con-man-resistant trust algorithm ran is 11.

The α_0-to-β_0 ratio used for these β_0 is 1:10, and the con man's constant C = e^{-1}. Here, $m = 10^5 - 2 = 99998$; hence, complexity is $\theta(\lceil \log_2 99998 \rceil)$ or $17 \geq 11$. The same difference of 1 at the least-significant digit is found for the α_0-to-β_0 ratio of 1:3.

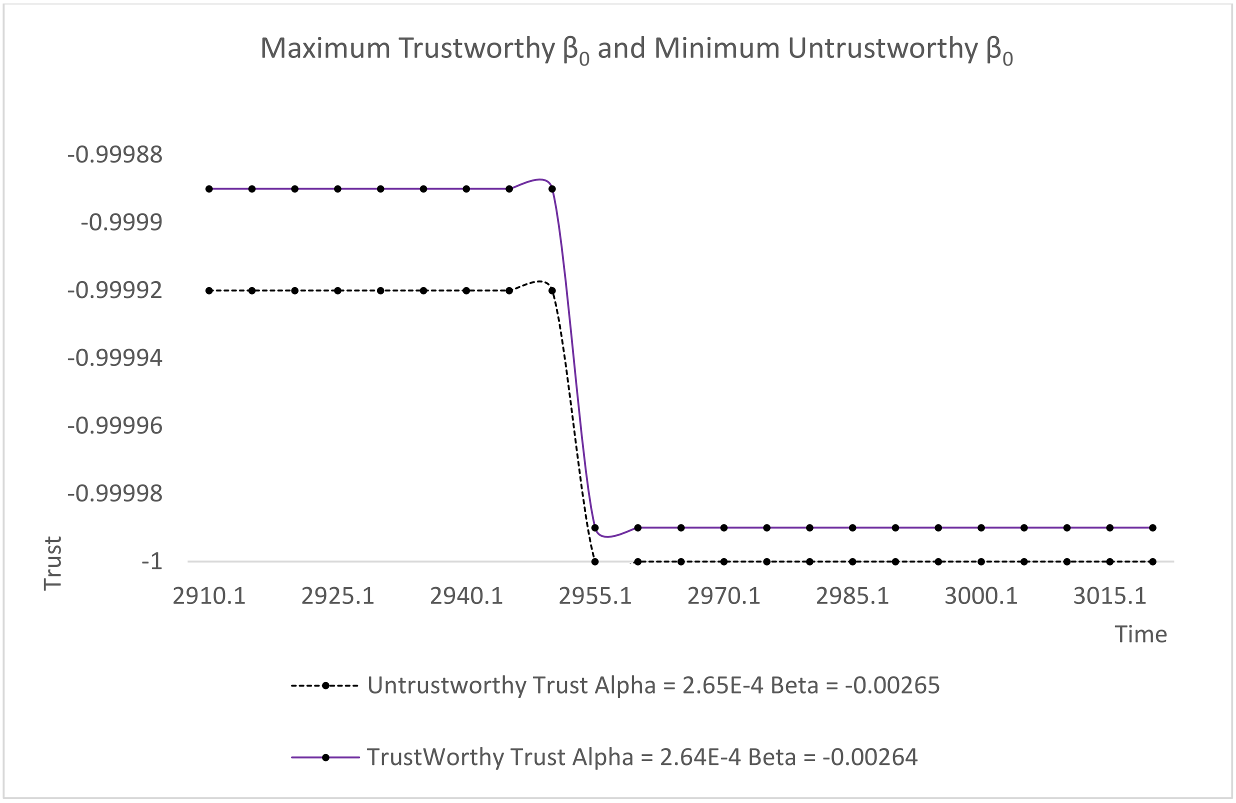


Figure 47. Maximum Trustworthy β_0 and Minimum Untrustworthy β_0.

Table 23 summarizes more data from the presented experiment for C = e^{-1} and a α_0-to-β_0 ratio of 1:3.

Table 23. Maximum Trustworthy β_0 and Minimum Untrustworthy β_0.

λ	Trust Converges for β_0 (Untrustworthy β_0)	Trust Does Not Converge for β_0 (Trustworthy β_0)	Number of Times that the Con-Man-Resistant Trust Algorithm Ran and Found β_0
5	- 0.32626	- 0.32625	9
5	-0.29893	-0.29892	10
5	-0.33778	-0.33777	7
6	-0.28510	-0.28509	8
6	-0.31416	-0.31415	10
7	-0.30253	-0.30252	8
7	-0.32555	-0.32554	8
7	-0.33060	-0.33059	10
8	-0.34567	-0.34566	8
8	-0.30336	-0.30335	9
8	-0.29491	-0.29490	6

In the experiments summarized in Table 23 and shown in Figure 47, a certain parameter setting for this con-man-resistant trust algorithm is used. However, it is necessary to know more about the parameter settings. There is a need for the con-man-resistant algorithm's parameter analysis because the algorithm is still in its evolving era. Recent works analyzed the parameter setting [6]. The parameters which affect the proposed machine-learning algorithm's performance and accuracy are listed in this chapter.

5.9. Sub Problem 3: Training, Testing, and Applying Machine Learning

The training data are adequate for the current application. This adequacy is known as the training-data coverage criteria. The training data are used to tune the selected parameter. During this training phase, the goal is to choose the maximum β_0 or minimum $|\beta_0|$. In addition, cross-validation is used to find how many training data are misidentified as not trustworthy for this β_0. The optimized algorithm is applied on simulated real-time data after training and cross-validation.

5.9.1. Training-Data Coverage Criteria

Higher QoS degradation (violation magnitude and QoS-degradation frequency) results in an earlier convergence of trust, resulting in a lower β_0. Similarly, lower QoS-degradation values result in a later convergence of trust, resulting in a higher β_0. There is a β_0 for the highest QoS degradation valued historical record is trustworthy. This β_0 is the lowest among all the β_0s from rest of the records. Hence, choosing the minimum β_0 among all the β_0s for that the consumer's historical record determines the specific consumer's allowed, or acceptable, QoS values. The |minimum β_0| serves as an optimal minimum-dominating set among all β_0s because there is only one such β_0 or |minimum β_0| = 1.

Figure 48 shows the overlapping area for different historical data (simulated) and their trustable β_0 values. There is a β_0 range above which any value of the trustable β_0 is not trustworthy, at least for one historical datum Below this range, the β_0 is trustable for all historical data used in the training. In the previously given example of decimal place = m, if there are n historical records, then the algorithm to find the concerned β_0 for a certain customer has the complexity of $\theta(n \times \lceil \log_2 m \rceil)$. The β_0 constitutes the optimal-minimum dominating set. This complexity is the optimality burden for this machine-learning version of con-man algorithm.

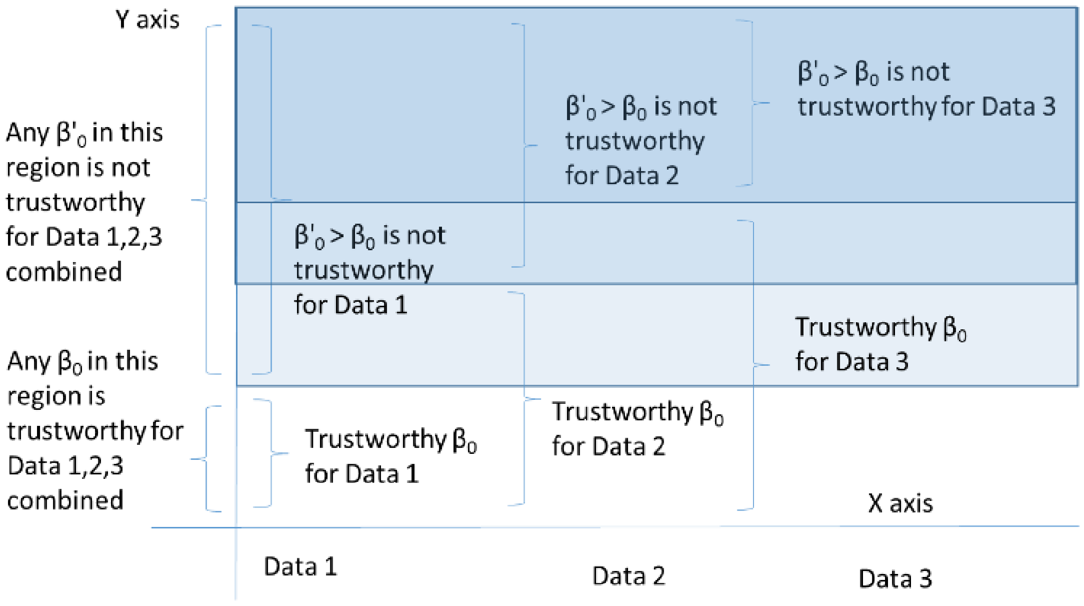

Y axis
Any β'0 in this region is not trustworthy for Data 1,2,3 combined
Any β0 in this region is trustworthy for Data 1,2,3 combined
β'0 > β0 is not trustworthy for Data 1
β'0 > β0 is not trustworthy for Data 2
β'0 > β0 is not trustworthy for Data 3
Trustworthy β0 for Data 1
Trustworthy β0 for Data 2
Trustworthy β0 for Data 3
X axis
Data 1
Data 2
Data 3

Figure 48. Choosing the Lowest β_0.

If the historical record does not represent a case which is trustable, the corresponding β_0 value is supposed to be unrevealed. Hence, the training-set coverage criteria could be collecting a sample of all trustworthy historical records that collectively represents all the trusted cases. However, such brute-force coverage criteria are eliminated by representing all the trustworthy cases with the highest-possible QoS degradation record or the record with minimum β_0.

The customer-satisfaction level related to the QoS degradation varies. Some consumers may not be dissatisfied with a certain percentile of QoS declination and its repetition above a certain time interval. The same values, however, can be unacceptable to another consumer. Hence, the declination of trust related to the mentioned values is consumer dependent. This dependency implies that, the con-man algorithm's parameters, e.g., a β_0 above which the service provider is not trustworthy, are dependent on a specific consumer. Each trustworthy β_0 implies the allowed QoS degradations or the allowed service instability for a certain consumer and a certain service-provider pair.

For multiple trusted historical records of the consumer receiving service from that specific service provider, the trustworthy β_0s can be found, one β_0 for each record. Finding the minimum number of historical records and the corresponding β_0 that represents the customer's satisfaction from the collection of historical records is essentially a set-coverage problem. If a consumer is satisfied with a certain frequency and value of QoS degradation, then that customer will most likely accept any lower frequencies and values. Hence, the lowest frequencies and values serve as a dominating set. Hence, the β_0 for which the highest QoS-degradation-containing historical records are identified as trustworthy is a representative of the consumer's acceptable QoS values.

5.9.2. Training and Cross-Validation

For this part, the con-man-resistant trust algorithm's parameter setting is $C = e^{-5}$; the α_0-to-β_0 ratio is 1:3; and the decimal precision is 5. Fivefold cross-validation is used to validate this machine-learning method [12]. Each $\lambda \in \{5, 6, 7, 8\}$ took 10 historical data, using a total of $10 \times 4 = 40$ time-series data. These data are divided into 5 same-size subsets in the next step. Each subset has 8 data points.

These subsets are non-overlapping because they are distinct from each other. Although, for the same λ, the data are conceptually similar, they cannot be the same (due to the randomness of Poisson-data generation).

In each 5 cases of cross-validation, shown in Table 24, 32 data are used for training (from 4 subsets), and the remaining 8 data (the remaining subset) are used for the test phase.

Table 25 shows the statistical summary of the unlocked β_0 from the training data set using the first subset.

Table 24. Fivefold Cross-Validation Summary.

Number of Folds	Each Fold Contains
5 folds	32 training data 8 test data

Table 25. The Training Data's Statistics for the First Subset.

Mean β_0	Maximum β_0	Minimum β_0	Standard Deviation	Variance
-0.31725	-0.35845	-0.28509	0.02025	0.00041

Table 26 shows the result from the first subset of the fivefold cross-validation. The table illustrates that, in 75% of the cases, the result is correct; i.e., the historical record is correctly identified as trustworthy.

Table 26. Sample Results for the Fivefold Cross-Validation (First Subset).

λ	Untrustworthy?	Trustworthy?	Trust Converged to -1 at Time
5	No	Yes	
5	Yes	No	143.1
6	No	Yes	
6	No	Yes	
7	Yes	No	141.1
7	No	Yes	
8	No	Yes	
8	No	Yes	

The result of Table 26 is very close for the incorrect part (two cases); i.e., the time until the data are treated as trustworthy is close to 143.1 milliseconds.

Table 27 shows a basic statistical summary for the other four subsets. The minimum $|\beta_0|$ corresponding to a subset is used for cross-validation. This minimum value is used as the $|\beta_0|$ for the con-man-resistant trust algorithm, and then, the algorithm is applied to the corresponding subset.

Table 27. Training-Data Summary for Four Subsets.

Subset No.	Mean β_0	Maximum $\|\beta_0\|$	Minimum $\|\beta_0\|$	Standard Deviation	Variance
2	-0.31745	-0.35845	-0.27278	0.02005	0.0004
3	-0.31607	-0.35845	-0.27278	0.02254	0.00051
4	-0.30934	-0.35441	-0.27278	0.01876	0.00035
5	-0.31425	-0.35845	-0.27278	0.02224	0.00049

Like Table 26, Table 28 shows the results from the second subset of the fivefold cross-validation. Table 28 illustrates that, in 100% of the cases, the result is correct; i.e., the historical record is correctly identified as trustworthy.

Table 28. Sample Result for the Fivefold Cross-Validation (Second Subset).

λ	Untrustworthy?	Trustworthy?
5	No	Yes
5	No	Yes
6	No	Yes
6	No	Yes
7	No	Yes
7	No	Yes
8	No	Yes
8	No	Yes

Table 29 shows the summary for this fivefold cross-validation. The table illustrates that, except for the first subset, all the other subsets have 100% correct results.

Table 29. Fivefold Cross-Validation Summary.

Fold Number	Number of Records	True Positive	False Negative	Correct Result %
1	8	6	2	75%
2	8	8	0	100%
3	8	8	0	100%
4	8	8	0	100%
5	8	8	0	100%

5.1. Application on Simulated, Real-Time Data

For SAAS simulation data with λ = 3, the applied parameter setting for the con-man-resistant trust algorithm is the learned $\beta_0 = -0.27278$ from the training phase (Figure 49). The expectation is that these data will be identified as untrustworthy, and that result happened at 139.1 milliseconds.

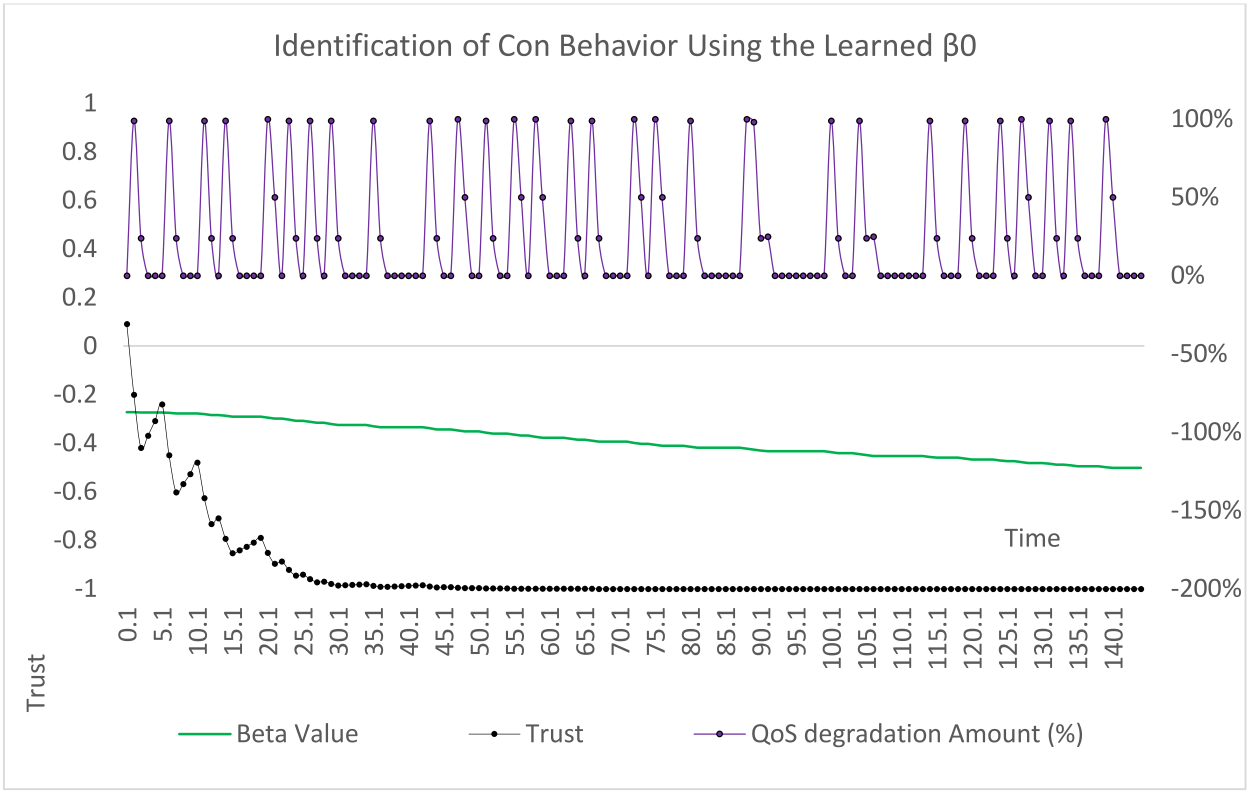


Figure 49. Con-Behavior Detection Using the Learned β_0 Value.

5.2. Con-Man-Resistant Trust Algorithm's Machine-Learning Version Limitations

It is not possible to find the minimum untrustworthy β_0 from a certain consumer's historical data all the time. This unattainable situation happens in two opposite, extreme cases as shown in Table 30.

Table 30. Cases with an Inability to Calculate the Minimum Untrustworthy β_0.

Case 1	Case 2
Compact performance-degradation appearance in data	For sparse performance-degradation appearance in data
Higher C value (e.g., C = e^{-1})	Lower C values
Higher α_0-to-β_0 ratio (e.g., 1:10)	Lower α_0-to-β_0 ratio (1:3)

5.3. Conclusion and Future Work

In this empirical study, a machine-learning process is presented to learn about user's preference by using the consumer's historical data. Some principles that the con-man-resistant algorithm follows are derived from this study. The project's assumption is the arrival of requests following the Poisson inter-arrival time. An algorithm is presented to determine the parameter settings for the con-man-resistant trust algorithm which reflect the consumer's preference. The algorithm's output is cross-validated after training. The contribution is to unlock properties the con-man-resistant trust algorithm's properties and to take the first step to learn.

Some enhancements for this learning process can be recommended for future research. First, the QoS-degradation magnitude can be recalculated with respect to a consumer's given tolerable value. There are cases where the consumer chooses or mentions the allowed, or tolerable, QoS-degradation magnitude. Hence, as a future study, the QoS degradations in a consumer's record can be further recalculated with respect to these tolerable values. However, the recalculation will change the distribution for both the generated QoS-degradation data and the violations' inter-arrival time. For example, if the consumer can tolerate a 50% QoS degradation and the QoS-degradation magnitude is 49%, then the recalculated QoS degradation (with respect to 50%) is 0%, resulting in no violation for that Poisson interval time. This recalculation changes the pattern of QoS-degradation repetition. The con-man-resistant trust algorithm can be run on the recalculated data in real time in order to identify the con deception.

Finally, the historical records' diversity can be maintained by following the research mentioned in [11]. The records can be split into collections where each collection has its own service-level agreement (SLA). The con-man algorithm can be run for each historical record, keeping in mind that the new QoS-degradation percentage needs to be recalculated with the appropriate QoS degradation. After running the con-man-resistant trust algorithm on historical records, there is one value for each con-man trust parameters for each record: α, β, γ, and trust. These values can, first, be aggregated following the aggregation process of [11]. In the second step, the aggregated values for each parameter (α, β, γ, trust) are used as initial trust parameters for the con-man trust calculation with real-time data.

CHAPTER 6. CONCLUSION

Cloud-computing models are becoming targets for exploitation. The cloud-computing model also inherits misuse from the distributed-computing and utility-computing models. The exploitation ranges from consumer-side fraudulent behavior to service-provider-side deceptive behavior. The loss associated with the deception is significant and is often incurred by the deception's victim. Hence, it is important to detect deceptions and to protect assets.

The work presented in this dissertation contributes by modeling a cloud-service provider's specific type of deception: the con-man attack. The deception exploits consumers with repeated service shortfalls. The repetition may or may not dissatisfy the contract between the consumer and the cloud-service provider. The cases of breaking the total-service-availability contract and not breaking the contract have a long-term effect on the consumer's business. When the cloud-service provider delivers service shortfalls, the provider has a short-term benefit: an operating-cost reduction and less investment. Hence, cloud-service providers that utilize con-man behaviors may do so to maximize profits or to minimize costs. If the cloud-service provider does not resolve this repetition issue, despite having revenue from the consumer, then the bad part of the cycle still affects the consumer. Examples of this effect are a loss for the consumer and also a long-term loss for the service provider. The incurred losses may be directly financial or subtle, such as the loss of reputation, customer base, or good will. Hence, this cyclic behavior is a fraud by the cloud-service provider. The mitigation algorithm for this fraud is the con-man-resistant trust algorithm.

The work is motivated from the presence and the effect of repeated service deficiencies as a deception. The study presents the sources of the con deception: the cloud-service provider predicting the consumer's service-usage pattern and allocating resource accordingly, rather than allocating all the time; the provider’s repeated service-shortfall behavior in a peer-to-peer network; an oscillating reputation in peer-to-peer network where a peer exhibits good service to achieve a good reputation, followed by acceptable service until the reputation touches the lower bound, with the goal of profit maximization; and the cyclic pattern of good and bad transactions from the service provider in a distributed network (oscillating pattern, increasing and decreasing pattern, and random pattern). The dissertation also presents the effect of this oscillating behavior as a declining number of consumers; e.g., softlayer.com lost its customers; some users of the Animato site left permanently; and the social-networking and gaming site friendster.com failed because its customers gradually left due to repeated slow responses. The described examples and the effect on businesses improve the consumer’s and the service provider’s awareness.

The dissertation contributes by applying the resistance algorithm to the con deception. The result shows how quickly the con-man behavior is identified with the con-man-resistant trust value. The findings also illustrate how well the con-man-resistant trust algorithm works with respect to another algorithm that aims to identify the service-quality issues.

Another contribution is completing the con-man-resistant trust algorithm by empirically analyzing its properties and the individual parameter’s properties, studying one parameter’s effect on another parameter, aggregating multiple transactions, describing the trust-model behavior using the state diagram, analyzing continuous cooperation-defection behavior, and proposing enhancements for the trust-calculation process. The study and analysis derived many useful principles.

An additional contribution is the development and evaluation of a performance-degradation-focused, con-man-resistant trust algorithm extension for the cloud services. Three exponential growth functions are added, and the functions are compared with each other concerning the con-behavior detection along with studying their properties. A study of the cloud service's performance-degradation magnitude's effect on con-man-resistant trust convergence is presented as principles. These rules and properties aid in decision making during the algorithm-configuration process of the con-man-resistant trust algorithm for a specific domain.

The work also contributes by recommending applications for the exponential growth functions. For example, the logistic function is recommended for use in resource-management and performance-tradeoff situations, e.g., in energy-aware cloud algorithms when there is a tradeoff between resource management and performance (e.g., between power consumption and performance). The compound-interest law is suitable for specific cloud services, e.g., cyber-attack prone services where the penalty needs to be an increasing value like the interest's debt side. The QoS-degradation magnitude can be expressed either as a continuous value or as intensity levels.

The con-man-trust algorithm's automated configuration is another contribution, i.e., enhancing the con-man-resistant trust algorithm's efficiency by automatically tuning its parameter values to reflect a consumer's perspective of trust for a cloud service. A machine-learning algorithm is designed and implemented to configure the con-man-resistant trust algorithm's parameters to reflect the consumer's preference. This algorithm configuration with machine learning unlocked a few properties of the con-man-resistant trust algorithm. The first step to add a learning capability to the con-man-resistant trust algorithm is designing and implementing this machine-learning algorithm. The algorithm is context sensitive after the learning process.

The dissertation research improves the efficiency of the con-man-resistant trust algorithm's application to detect the cloud-service provider's con behavior as a deception.

Identifying the con deception helps the consumer take action, e.g., leaving the service provider or selecting the service provider wisely, to avoid further financial losses from the con deceit. The identification also helps the cloud-service provider to dynamically evaluate its service deficiencies and to take action, e.g., sufficient investment, to overcome them. The presented work also recommends different applications for the con-man-resistant trust algorithm in a cloud-computing environment.

In conclusion, this dissertation is a significant research work that proposes a new context-sensitive evaluation method for cloud services. The evaluation states that, the less stable the cloud-service quality is, the more it affects the service's consumer. The significance of this study is instrumenting the consumer or the service provider to detect the cloud service's deficiency by identifying an unstable cloud service as a con deception. The service provider can review its investment decisions by evaluating the business service's score that was given by the con-man-resistant trust algorithm for cloud services. The con behavior has an economic effect because it incurs loss, directly or indirectly. Hence, the presented work has significance for cloud economics because the con-man-resistant trust algorithm for the cloud prevents the loss with early detection of the con deception. The con-man-resistant trust algorithm's parameter setting depends upon the consumer type. The consumer or provider can tune the algorithm's parameter depending upon the flexibility choice (e.g., a flexible consumer versus a conservative consumer). If the consumer or provider wants the deficiency tolerance to be reflected in the service-deficiency detection, they can apply the recommended machine-learning version of the con-man-resistant trust algorithm.

REFERENCES

[1] A. Salehi-Abari and T. White, “Towards con-resistant trust models for distributed agent systems,” in *International Joint Conference on Artificial Intelligence*, Pasadena, CA, 2009, pp. 272–277.

[2] A. Salehi Abari and T. White, “Trust Models and Con-Man Agents: From Mathematical to Empirical Analysis,” in *The Twenty-Fourth AAAI Conference on Artificial Intelligence*, Atlanta, Georgia, Jul. 2010, pp. 842–847.

[3] C. Shipman, K. Hopkinson, and J. Lopez, “Con-resistant trust for improved reliability in a smart-grid special protection system,” *IEEE Trans. Power Deliv.*, vol. 30, no. 1, pp. 455–462, Feb. 2015.

[4] M. Chowdhury and K. Nygard, “Deception in Cyberspace: An Empirical Study on a Con Man Attack,” in *2017 IEEE International Conference on Electro Information Technology (EIT),* Lincoln, Nebrasca, May 2017, pp. 410–415.

[5] M. Chowdhury, K. Nygard, K. Kambhampaty, and M. Alruwaythi, “Deception in Cyberspace: Performance Focused Con Resistant Trust Algorithm,” in *the 4th Annual Conference on Computational Science & Computational Intelligence*, Las Vegas, Nevada, Dec. 2017.

[6] M. Chowdhury and K. Nygard, “An Empirical Study on a Con Resistant Trust Algorithm for Cyberspace,” in *Proceedings of the 2017 International Conference on Internet Computing and Internet of Things, Las Vegas, Nevada*, Jul. 17-20, 2017, pp. 32–37.

[7] M. Chowdhury and K. Nygard, “Machine Learning within a Con Resistant Trust Model,” in *the 33rd International Conference on Computers and Their Applications (CATA 2018),* Las Vegas, Nevada, Mar. 19-21, 2018, pp. 9–14.

[8] “softlayer.com cloud computing repeated downtime - read only mode | Web Hosting Talk.” [Online]. Available: http://www.webhostingtalk.com/showthread.php?t=1067657. [Accessed: 19-Apr-2018].

[9] M. Armbrust *et al.*, “Above the Clouds: A Berkeley View of Cloud Computing,” in UC Berkeley Technical Report, Feb. 2009.

[10] M. Armbrust *et al.*, “A view of cloud computing,” *Commun. ACM*, vol. 53, no. 4, p. 50, Apr. 2010.

[11] L. Qi, W. Dou, J. Ni, X. Xia, C. Ma, and J. Liu, “A Trust Evaluation Method for Cloud Service with Fluctuant QoS and Flexible SLA,” in the *2014 IEEE International Conference on Web Services*, Anchorage, Alaska, Jan. 2014, pp. 345–352.

[12] K. Nygard, M. Chowdhury, and P. Kotala, “Trust and Purpose in Computing,” in *32nd International Conference on Computers and Their Applications*, Honolulu, Hawaii, Mar. 2017, pp. 161–166.

[13] K. Nygard, M. Chowdhury, A. Bugalwi, and P. Kotala, “People and Intelligent Machines in Decision Making,” *Int. J. Comput. Their Appl.*, 2017.

[14] C. Dong, Y. Wang, A. Aldweesh, P. Mccorry, and A. Van Moorsel, “Betrayal, Distrust, and Rationality: Smart Counter-Collusion Contracts for Verifiable Cloud Computing,” *in Proceedings of the 2017 ACM SIGSAC Conf. Comput. Commun. Secur.*, Dallas, Texas, pp. 211–227, 2017.

[15] M. Walfish and A. Blumberg, “Verifying computations without reexecuting them,” *Commun. ACM*, vol. 58, no. 2, pp. 74–84, Jan. 2015.

[16] H. Galanxhi and F. Nah, “Deception in cyberspace: A comparison of text-only vs. avatar-supported medium,” *Int. J. Hum. Comput. Stud.*, vol. 65, no. 9, pp. 770–783, Sept. 2007.

[17] D. Buller, K. Strzyzewski, and J. Comstock, “Interpersonal deception: I. deceivers’ reactions to receivers’ suspicions and probing,” *Commun. Monogr.*, vol. 58, no. 1, pp. 1–24, Mar. 1991.

[18] J. Burgoon and J. Nunamaker, “Toward computer-aided support for the detection of deception,” *Gr. Decis. Negot.*, vol. 13, no. 1, pp. 1–4, 2004.

[19] Z. Zhang, Y. C. Hu, and S. P. Midkiff, “CycleMeter: Detecting Fraudulent Peers in Internet Cycle Sharing,” in SC '06: *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, Tampa, FL, Nov. 2006.

[20] A. Bardsiri and S. Hashemi, “QoS Metrics for Cloud Computing Services Evaluation,” *Int. J. Intell. Syst. Appl.*, vol. 6, no. 12, pp. 27–33, 2014.

[21] S. Garg, S. Versteeg, and R. Buyya, “A framework for ranking of cloud computing services,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 4, pp. 1012–1023, 2013.

[22] A. Beloglazov and R. Buyya, “Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers,” in *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, Bangalore, India, pp. 1–6, Nov. 2010.

[23] A. Beloglazov, J. Abawajy, and R. Buyya, “Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing,” *Futur. Gener. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, May 2012.

[24] A. Beloglazov and R. Buyya, “Energy Efficient Resource Management in Virtualized Cloud Data Centers,” *in 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, Melbourne, VIC, Australia, 2010.

[25] J. Comellas, I. Parallel, and J. Fernandez, “SLA-driven Elastic Cloud Hosting Provider,” *the 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, Pisa, Italy, pp. 111–118, 2010.

[26] M. Hussain and M. Al-Mourad, “Effective Third Party Auditing in Cloud Computing,” in *2014 the 28th International Conference on Advanced Information Networking and Applications Workshops,* Victoria, BC, Canada, May 2014.

[27] M. Hussain and M. Al-Mourad, “A crowdsource model for quality assurance in cloud computing,” *Int. J. Grid Util. Comput.*, vol. 7, no. 3, p. 177, 2016.

[28] L. Xiong and L. Liu, “PeerTrust: Supporting reputation-based trust for peer-to-peer electronic communities,” *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 7, pp. 843–857, Jul. 2004.

[29] F. Mármol and G. Pérez, “Security threats scenarios in trust and reputation models for distributed systems,” *Comput. Secur.*, vol. 28, no. 7, pp. 545–556, 2009.

[30] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya, “CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Softw. - Pract. Exp.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[31] R. Calheiros, R. Ranjan, C. De Rose, and R. Buyya, “CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services,” Software—Practice Exp., vol. 41, no. 1, pp. 23–50, Mar. 2009.

[32] L. Janczewski and A. Colarik, *Cyber Warfare and Cyber Terrorism*. Information Science Reference (IGI Global), 2007.

[33] H. Zhang, L. Ye, J. Shi, X. Du, and M. Guizani, “Verifying cloud Service Level Agreement,” in *2012 IEEE Global Communications Conference (GLOBECOM)*, Anaheim, CA, Dec. 2012, pp. 777–782.

[34] H. Zhang, L. Ye, J. Shi, X. Du, and M. Guizani, “Verifying cloud service-level agreement by a third-party auditor,” *Secur. Commun. Networks*, vol. 7, no. 3, pp. 492–502, Mar. 2014.

[35] A. Juels and A. Oprea, “New approaches to security and availability for cloud data,” *Commun. ACM*, vol. 56, no. 2, p. 64, Feb. 2013.

[36] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione, “Smart Cloud Storage Service Selection Based on Fuzzy Logic, Theory of Evidence and Game Theory,” *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2348–2362, Aug. 2016.

[37] L. Wei *et al.*, “Security and privacy for storage and computation in cloud computing,” *Inf. Sci. (Ny).*, vol. 258, pp. 371–386, 2014.

[38] Q. Huang, L. Ye, X. Liu, and X. Du, “Auditing CPU performance in public cloud,” in *Proceedings of the 2013 IEEE 9th World Congress on Services*, Santa Clara, CA, June 2013, pp. 286–289.

[39] J. Xu, Q. Wen, W. Li, and Z. Jin, “Circuit Ciphertext-Policy Attribute-Based Hybrid Encryption with Verifiable Delegation in Cloud Computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 1, pp. 119–129, Jan. 2016.

[40] S. Bouchenak, G. Chockler, H. Chockler, G. Gheorghe, N. Santos, and A. Shraer, “Verifying cloud services,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 47, no. 2, p. 6, Jul. 2013.

[41] P. Wieder, J. Butler, W. Theilmann, and R. Yahyapour, *Service Level agreements for Cloud Computing*, Illustrate. Springer Science & Business Media, 2011.

[42] P. Patel, A. Ranabahu, and A. Sheth, “Service Level Agreement in Cloud Computing,” *Kno.e.sis Publ.*, vol. 14, no. 6, Jan. 2009.

[43] B. Kandukuri, R. Paturi, and A. Rakshit, “Cloud Security Issues,” in *2009 IEEE International Conference on Services Computing*, Bangalore, India, Oct. 2009, pp. 517–520.

[44] Cloud Standards Customer Council, “Practical Guide to Cloud Service Level Agreements, version 2.0,” the Cloud Standards Customer Council, Apr. 2015.

[45] M. Alhamad, T. Dillon, and E. Chang, “Conceptual SLA framework for cloud computing,” in *4th IEEE International Conference on Digital Ecosystems and Technologies*, Dubai, Apr. 2010, pp. 606–610.

[46] I. Chana and S. Singh, “Quality of Service and Service Level Agreements for Cloud Environments: Issues and Challenges,” in *Cloud Computing. Computer Communications and Networks,* Springer, Cham, 2014.

[47] Norwegian Telecommunications Regulatory Authority, “Model for Service Level Agreement (SLA) Included Quality of Service (QoS) Descriptions,” Jan. 2005.

[48] Telecommunication Standardization Sector of International Telecommunication Union, “Terms and Definitions Related to Quality of Service and Network Performance Including Dependability,” International Telecommunication Union, Recommendation E.800, Aug. 1994.

[49] I. Villy, “Teletraffic engineering handbook,” *Itu-D Sg*, vol. 2, no. January, p. 16, 2001.

[50] A. Mani and A. Nagarajan, “Understanding Quality of Service for Web services,” 2002. [Online]. Available: https://www.ibm.com/developerworks/library/ws-quality/index.html. [Accessed: 11-Apr-2018].

[51] S. Marsh, “Formalising Trust as a Computational Concept,” Department of Computer Science, University of Stirling, Scotland, 1994.

[52] J. Lansing and A. Sunyaev, “Trust in Cloud Computing,” *ACM SIGMIS Database Database Adv. Inf. Syst.*, vol. 47, no. 2, pp. 58–96, 2016.

[53] E. Chang, T. Dillon, and F. Hussain, “Trust and reputation relationships in service-oriented environments,” in *Proceedings of the 3rd International Conference on Information Technology and Applications*, Sydney, New South Wales, Jul. 2005, pp. 4–14.

[54] A. Halberstadt, L. Mui, M. Mohtashemi, C. Ang, and P. Szolovits, “Ratings in Distributed Systems : A Bayesian Approach,” in *Proceedings of the Workshop on Information Technologies and Systems*, New Orleans, 2001, pp. 1–7.

[55] R. Albuquerque, L. Villalba, A. Orozco, R. Sousa, and T. Kim, “Leveraging information security and computational trust for cybersecurity,” *J. Supercomput.*, vol. 72, no. 10, pp. 3729–3763, Oct. 2016.

[56] D. Romano, “The Nature of Trust: Conceptual and Operational Clarification,” Louisiana State University and Agricultural and Mechanical College, 2003.

[57] D. Gambetta, “Can we trust trust?,” in *Trust: Making and Breaking Cooperative Relations*, Electronic., The Bodleian Library, University of Oxford, 2000, pp. 213–237.

[58] A. Josang, C. Keser, and T. Dimitrakos, “Can We Manage Trust?,” in *Proceedings of the 3rd International Conference on Trust Management (iTrust 2005)*, Paris, France, May 2005, pp. 93–107.

[59] A. Salehi-Abari, “The Exploitation-Resistant Trust (ERT) Model for Open Distributed Systems,” School of Computer Science, Carleton University, 2009.

[60] X. Li, F. Zhou, and X. Yang, “A multi-dimensional trust evaluation model for large-scale P2P computing,” *J. Parallel Distrib. Comput.*, vol. 71, no. 6, pp. 837–847, 2011.

[61] Q. Zhang, T. Yu, and K. Irwin, “A classification scheme for trust functions in reputation-based trust management,” *IProceedings 2004 Int. Conf. Trust. Secur. Reput. Semant. Web* 2004, vol. 127, pp. 52–61.

[62] E. Blasch, Y. Al-Nashif, and S. Hariri, “Static versus dynamic data information fusion analysis using DDDAS for cyber security trust,” in *Procedia Computer Science*, 2014, vol. 29, pp. 1299–1313.

[63] R. Levien, “Attack Resistant Trust Metrics.” in *Computing with Social Trust. Human–Computer Interaction Series. Springer, London,* 2009.

[64] E. Canedo, R. Albuquerque, and R. Sousa, “Trust Model for File Sharing in Cloud Computing,” in *the Second International Conference on Cloud Computing, GRIDs, and Virtualization*, Rome, Italy, Sept. 2011, pp. 66–73.

[65] S. Chakraborty and K. Roy, “An SLA-based framework for estimating trustworthiness of a cloud,” in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, Liverpool, Sept. 2012, pp. 937–942.

[66] M. Alhamad, T. Dillon, and E. Chang, “SLA-Based Trust Model for Cloud Computing,” in *2010 13th International Conference on Network-Based Information Systems*, Takayama, Japan, Sept. 2010, pp. 321–324.

[67] P. Gupta, M. K. Goyal, P. Kumar, and A. Aggarwal, “Trust and reliability based scheduling algorithm for cloud IaaS,” in *Lecture Notes of Electrical Engineering*, 2013, vol. 150, pp. 603–607.

[68] V. Kumar and M. Aramudhan, “Trust Based Resource Selection and List Scheduling in Cloud Computing,” *Int. J. Adv. Eng. Technol.*, vol. 6, no. 6, pp. 2455–2463, 2014.

[69] P. S. Pawar, M. Rajarajan, S. K. Nair, and A. Zisman, “Trust model for optimized cloud services,” in *IFIP Advances in Information and Communication Technology*, 2012, vol. 374, pp. 97–112.

[70] S. Habib, S. Ries, and M. Muhlhauser, “Towards a Trust Management System for Cloud Computing,” in *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, Changsha, China, Nov. 2011, pp. 933–939.

[71] E. Chang, F. Hussain, and T. Dillon, “Fuzzy nature of trust and dynamic trust modeling in service oriented environments,” in *Proceedings of the 2005 workshop on Secure web services*, Fairfax, VA, Nov. 2005, p. 75-83.

[72] B. Barber and D. Gambetta, “Trust: Making and Breaking Cooperative Relations.,” *Contemp. Sociol.*, vol. 21, no. 3, p. 401, 1992.

[73] G. Lu, J. Lu, S. Yao, and J. Yip, “A Review on Computational Trust Models for Multi-agent Systems,” *Open Inf. Sci. J.*, vol. 2, no. 2, pp. 18–25, 2009.

[74] Forrester Research, “Developing a Framework to Improve Critical Infrastructure Cybersecurity,” the National Institute of Science and Technology within the Department of Commerce, Apr. 2017.

[75] H. Sato, A. Kanai, and S. Tanimoto, “A Cloud Trust Model in A Security Aware Cloud,” in *Proceedings of the 2010 10th Annual International Symposium on Applications and the Internet*, Seoul, South Korea, Jul. 2010, pp. 121–124.

[76] L. Y. Njilla, N. Pissinou, and K. Makki, “Game Theoretic Modeling of Security and Trust Relationship in Cyberspace,” *Int. J. Commun. Syst.*, vol. 29, no. 9, pp. 1500–1512, Jun. 2016.

[77] J. Zou, “Accountability in Cloud Services,” Macquarie University, 2016.

[78] M. Urbano, “A Situation Aware and Social Computational Trust Model,” Department of Informatics Engineering, University of Porto (Portugal), May 2013.

[79] F. Moyano, C. Fernandez-Gago, and J. Lopez, “Service-Oriented Trust and Reputation Architecture”, in Proceedings of the Doctoral Symposium of the International Symposium on Engineering Secure Software and Systems (ESSoS-DS 2012), CEUR-WS vol. 834, pp. 41-46, 2012.

[80] N. B. Truong, H. Lee, B. Askwith, and G. M. Lee, “Toward a Trust Evaluation Mechanism in the Social Internet of Things,” *Sensors (Switzerland)*, vol. 17, no. 6, pp. 1–24, Jun 2017.

[81] D. Ross, “Game Theory,” *The Stanford Encyclopedia of Philosophy (Winter 2016 Edition)*. Edward N. Zalta (ed.), 2016.

[82] G. Baranwal and D. Vidyarthi, “Admission control in cloud computing using game theory,” *J. Supercomput.*, vol. 72, no. 1, pp. 317–346, 2016.

[83] S. Hart, “Robert Aumann’s game and economic theory,” *Scand. J. Econ.*, vol. 108, no. 2, pp. 185–211, 2006.

[84] T. Wilhelm and J. Andress, *Ninja Hacking: Unconventional Penetration Testing Tactics and Techniques*. Syngress, Sept. 2010.

[85] R. C. Newman, *Computer forensics : evidence collection and management*. Auerbach Publications, 2007.

[86] C. Wang, Q. Wang, K. Ren, and W. Lou, “Privacy-preserving public auditing for data storage security in cloud computing,” in *Proceedings of the IEEE INFOCOM*, San Diego, CA, USA, May 2010, pp. 1–9.

[87] C. Wang, S. Chow, Q. Wang, K. Ren, and W. Lou, “Privacy-preserving public auditing for secure cloud storage,” *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 362–375, Feb. 2013.

[88] S. Han and J. Xing, “Ensuring data storage security through a novel third party auditor scheme in cloud computing,” in *Proceedings of the 2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, Beijing, China, Sept. 2011, pp. 264–268.

[89] S. Lins, H. Teigeler, and A. Sunyaev, “Towards A Bright Future: Enhancing Diffusion of Continuous Cloud Service Auditing by Third Parties,” in *24th European Conference on Information Systems*, Istanbul, Turkey, Jun. 2016, pp. 1–17.

[90] S. Lins, P. Grochol, S. Schneider, and A. Sunyaev, “Dynamic Certification of Cloud Services: Trust, but Verify!,” *IEEE Secur. Priv.*, vol. 14, no. 2, pp. 66–71, Mar. 2016.

[91] D. He, S. Zeadally, and L. Wu, “Certificateless Public Auditing Scheme for Cloud-Assisted Wireless Body Area Networks,” *IEEE Syst. J.*, vol. 12, no. 1, pp. 64–73, Mar. 2018.

[92] Y. Xiaoyong, L. Ying, J. Tong, L. Tiancheng, and W. Zhonghai, “An Analysis on Availability Commitment and Penalty in Cloud SLA,” in *2015 IEEE 39th Annual Computer Software and Applications Conference*, Taichung, Taiwan, Jul. 2015, pp. 914–919.

[93] N. Elprince, “Autonomous resource provision in virtual data centers,” in 2*013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013),* Ghent, Belgium, May 2013.

[94] E. Aljoumah, F. Al-mousawi, I. Ahmad, and M. Al-shammri, “SLA in Cloud Computing Architectures : A Comprehensive Study,” *Int. J. Grid Distrib. Comput.*, vol. 8, no. 5, pp. 7–32, 2015.

[95] “The Father of Social Media, Jonathan Abrams | Eyerys.” [Online]. Available: https://www.eyerys.com/articles/people/father-social-media-jonathan-abrams#3. [Accessed: 20-Apr-2018].

[96] P. Guttorp and T. Thorarinsdottir, “What Happened to Discrete Chaos, the Quenouille Process, and the Sharp Markov Property? Some History of Stochastic Point Processes,” *Int. Stat. Rev.*, vol. 80, no. 2, pp. 253–268, Aug. 2012.

[97] Xi. Sun, G. Chang, and F. Li, “A Trust Management Model to Enhance Security of Cloud Computing Environments,” in *2011 Second International Conference on Networking and Distributed Computing*, Beijing, China, Sept. 2011, pp. 244–248.

[98] R. Yu, Y. Zhang, S. Gjessing, W. Xia, and K. Yang, “Toward Cloud-based vehicular networks with efficient resource management,” *IEEE Netw.*, vol. 27, no. 5, pp. 48–55, Sept.-Oct. 2013.

[99] J. Lee, J. Lee, D. Cheun, and S. Kim, “A Quality Model for Evaluating Software-as-a-Service in Cloud Computing,” in Proceedings of the *2009 Seventh ACIS International Conference on Software Engineering Research, Management and Applications*, Haikou, China, Dec. 2009, pp. 261–266.

[100] P. Manuel, S. Selvi, and M. Barr, “A Novel Trust Management System for cloud Computing - Iaas Provider,” *J. Comb. Math. Comb. Comput.*, vol. 79, no. 1, pp. 3–22, 2011.

[101] H. Chen, H. Wu, J. Hu, and C. Gao, "Agent-Based Trust Management Model for Wireless Sensor Networks," in *2008 International Conference on Multimedia and Ubiquitous Engineering (mue 2008)*, Busan, South Korea, Apr. 2008, pp. 150–154.

[102] H. Chen, "Task-based trust management for wireless sensor networks," *Int. J. Secur. its Appl.*, vol. 3, no. 2, pp. 21–26, 2009.

[103] T. A. Babbitt and B. K. Szymanski, "Trust metric integration in resource constrained networks via data fusion," *2015 18th Int. Conf. Inf. Fusion*, Washington, DC, USA, pp. 582–589, Jul. 2015.

[104] F. Ishmanov, S. Kim, and S. Nam, "A robust trust establishment scheme for wireless sensor networks," *Sensors (Switzerland)*, vol. 15, no. 3, pp. 7040–7061, Mar. 2015.

[105] G. Han, L. Liu, J. Jiang, L. Shu, and J. Rodrigues, "A Collaborative Secure Localization Algorithm Based on Trust Model in Underwater Wireless Sensor Networks," *Sensors(Basel)*, vol. 16, no. 2, p. 229, Feb. 2016.

[106] Z. Ye, T. Wen, Z. Liu, X. Song, and C. Fu, "An Efficient Dynamic Trust Evaluation Model for Wireless Sensor Networks," *J. Sensors*, vol. 2017, pp. 1–16, Oct. 2017.

[107] J. Zhang, R. Shankaran, M. Orgun, V. Varadharajan, and A. Sattar, "A Dynamic Trust Establishment and Management Framework for Wireless Sensor Networks," in *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, Hong Kong, China, Dec. 2010.

[108] W. Li and Z. Chunjian, "An Optimized and Improved Network Trust Model Based on P2P." Journal of Networks, vol. 8, no. 10, 2013, p. 2293.

[109] M. Morid and M. Shajari, "An enhanced e-commerce trust model for community based centralized systems," *Electron. Commer. Res.*, vol. 12, no. 4, pp. 409–427, 2012.

[110] R. Shaikh, H. Jameel, B. D'Auriol, H. Lee, S. Lee, and Y. Song, "Group-based trust management scheme for clustered wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 11, pp. 1698–1712, Nov. 2009.

[111] S. Ps, R. Pasumarthy, and H. Kruthika, "Stability Analysis of Cloud Computing Systems under Uncertain Time delays," in the 21st International Symposium on Mathmatical Theory of Netowrks and Systems, 2014, pp. 1657–1664.

[112] C. Mastroianni, M. Meo, and G. Papuzzo, "Probabilistic Consolidation of Virtual Machines in Self-Organizing Cloud Data Centers," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 215–228, 2013.

[113] A. Hammadi and O. Hussain, "A framework for SLA assurance in cloud computing," in *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2012*, Fukuoka, Japan, mar. 2012, pp. 393–398.

[114] J. Huang and D. M. Nicol, "Trust mechanisms for cloud computing," *J. Cloud Comput.*, vol. 2, no. 1, p. 9, Oct. 2013.

[115] Z. Yang, X. Qin, Y. Yang, and T. Yagink, "A hybrid trust service architecture for cloud computing," in *Proceedings of the 2013 International Conference on Computer Sciences and Applications, CSA 2013*, Wuhan, China, Dec. 2013, pp. 674–680.

[116] B. Koops and M. Goodwin, "Cyberspace, the Cloud, and Cross-Border Criminal Investigation. The Limits and Possibilities of International Law," Tilburg Law School Research Paper No. 5/2016, *Tilburg Institute for Law, Technology, and Society CTLD – Center for Transboundary Legal Development*, Dec. 2014.

[117] M. Graham, “Cloud Collaboration: Peer-Production and the Engineering of the internet,” in *Engineering Earth*, Dordrecht: Springer Netherlands, 2011, pp. 67–83.

[118] B. P. Rimal, E. Choi, and I. Lumb, “A taxonomy and survey of cloud computing systems,” in *5th International Joint Conference on INC, IMS, and IDC,* Seoul, South Korea, Aug. 2009, pp. 44–51.

[119] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, “Performance analysis of cloud computing services for many-tasks scientific computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 6, pp. 931–945, Jun. 2011.

[120] K. R. Jackson *et al.*, “Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud,” in *IEEE Second International Conference on Cloud Computing Technology and Science*, Indianapolis, IN, USA, Dec. 2010, pp. 159–168.

[121] R. Larson, *Notetaking Guide for Larson’s Precalculus With Limits A Graphing Approach, Texas Edition.* Cengage Learning, 2014.

[122] N. Rowe, “Deception in Defense of Computer Systems from Cyber Attack,” in *Cyber Warfare and Cyber Terrorism*, Monterey, California. Naval Postgraduate School, 2007, pp. 97–104.

[123] G. Briscoe and A. Marinos, “Digital ecosystems in the clouds: Towards community cloud computing,” in *3rd IEEE International Conference on Digital Ecosystems and Technologies, DEST ’09*, Istanbul, Turkey, June 2009, pp. 103–108.

[124] A. Marinos and G. Briscoe, “Community cloud computing,” in (eds) *Cloud Computing. CloudCom 2009. Lecture Notes in Computer Science, vol 5931. Springer, Berlin, Heidelberg.*

[125] L. Stein, "The case for cloud computing in genome informatics," *Genome Biology*, vol. 11, no. 5. BioMed Central, p. 207, May 2010.

[126] D. Reed and J. Dongarra, "Scientific discovery and engineering innovation requires unifying traditionally separated high- performance computing and big data analytics," *Commun. ACM*, vol. 58, no. 7, 2015.

[127] J. Zander and P. Mahonen, "Riding the data tsunami in the cloud: Myths and challenges in future wireless access," *IEEE Commun. Mag.*, vol. 51, no. 3, pp. 145–151, Mar. 2013.

[128] G. A. Pratt, "Is a Cambrian Explosion Coming for Robotics?," *J. Econ. Perspect.*, vol. 29, no. 3, pp. 51–60, Aug. 2015.

[129] V. H. BLACKMAN, "The Compound Interest Law and Plant Growth," *Ann. Bot.*, vol. os-33, no. 3, pp. 353–360, Jul. 1919.

[130] K. Fischer and K. Fiedler, "Reaction norms for age and size at maturity in response to temperature: A test of the compound interest hypothesis," *Evol. Ecol.*, vol. 16, no. 4, pp. 333–349, 2002.

[131] J. Tague, J. Beheshti, and L. Rees-Potter, "The Law of Exponential Growth: Evidence, Implications and Forecasts," *Libr. Trends*, vol. 30, no. 1, pp. 125–149, 1981.

[132] Z. Han, T. Liu, Q. Sun, R. Li, J. Xie, and B. Li, "Application of compound interest laws in biology: Reunification of existing models to develop seed bank dynamics model of annual plants," *Ecol. Modell.*, vol. 278, pp. 67–73, 2014.

[133] J. Bissell, "The decline in the British bank population since 1810 obeys a law of negative compound interest," *Bus. Hist.*, vol. 59, no. 5, pp. 802–813, 2017.

[134] B. Sharma, R. Thulasiram, P. Thulasiraman, and R. Buyya, “Clabacus: A Risk-Adjusted Cloud Resources Pricing Model Using Financial Option Theory,” *IEEE Trans. Cloud Comput.*, vol. 3, no. 3, pp. 332–344, 2015.

[135] T. Au and T. Au, "*Engineering Economics for Capital Investment Analysis" in the Engineering Economist, vol. 38, no. 1,* 1992.

[136] T. Kocak, “Sigmoid Functions and Their Usage in Artificial Neural Networks,” in *Lecture Notes of Applications of Calculus II: Inverse Functions, School of Electrical Engineering and Computer Science University of Central Florida*. p. 29, 2015.

[137] “Exponential and Logistic Functions.” in *Lecture Notes* of Ottawa Hills Jr./Sr. School, 2009. [Online]. Available: http://www.ohschools.k12.oh.us/userfiles/225/Classes/73/6per3-1oct16.pdf. [Accessed: 19-Apr-2018].

[138] M. Jordan, “Why the logistic function? A tutorial discussion on probabilities and neural networks,” Technical Report 9503, Massachusetts Institute of Technology, 1995.

[139] P. Kennis, “Method and system for detecting malicious behavioral patterns in a computer, using machine learning,” U.S. Patent US20120198549A1, Jan. 2008.

[140] S. Suakanto, S. Supangkat, and R. Saragih, “Performance Measurement of Cloud Computing Services,” *Int. J. Cloud Comput. Serv. Archit.*, vol. 2, no. 2, 2012.

[141] P. Zhang and Z. Yan, “A QoS-aware system for mobile cloud computing,” in *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, Beijing, China, Sept. 2011.

[142] N. Roy, S. Das, K. Basu, and M. Kumar, “Enhancing Availability of Grid Computational Services to Ubiquitous Computing Applications,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 7, pp. 953–967, January 2009.

[143] F. Zulkernine and P. Martin, “An adaptive and intelligent SLA negotiation system for web services,” *IEEE Trans. Serv. Comput.*, vol. 4, no. 1, pp. 31–43, 2011.

[144] I. Goiri, F. Julia, J. Fito, M. Maclas, and J. Guitart, “Supporting CPU-based guarantees in cloud SLAs via resource-level QoS metrics,” *Futur. Gener. Comput. Syst.*, vol. 28, no. 8, pp. 1295–1302, 2012.

[145] H. Van, F. Tran, and J. Menaud, “Performance and Power Management for Cloud Infrastructures,” in *IEEE 3rd International Conference on Cloud Computing*, Miami, Florida, USA, July 2010, pp. 329–336.

[146] J. Dong, H. Wang, and S. Cheng, “Energy-performance tradeoffs in IaaS cloud with virtual machine scheduling,” *China Commun.*, vol. 12, no. 2, pp. 155–166, Feb. 2015.

[147] Holger H. Hoos, “Automated Algorithm Configuration and Parameter Tuning,” in *Autonomous Search*, Springer, Berlin, Heidelberg, 2011, pp. 37–71.

[148] F. Hutter, H. Hoos, and T. Stuetzle, “ParamILS: An Automatic Algorithm Configuration Framework Kevin Leyton-Brown,” *J. Artif. Intell. Res.*, vol. 36, pp. 267–306, 2009.

[149] N. Mahendran, Z. Wang, F. Hamze, and N. Freitas, “Adaptive MCMC with Bayesian Optimization.” *J. Mach. Learn. Res.*, vol. 22, pp. 751-760, 2012.

[150] M. Lindauer, H. Hoos, F. Hutter, and T. Schaub, “AutoFolio: Algorithm Configuration for Algorithm Selection.” in *29th AAAI Workshops on Artificial Intelligence,* Austin, Texas, USA, January 2015.

[151] F. Hutter, H. Hoos, and K. Leyton-Brown, “Parallel Algorithm Configuration.” in *Proceedings of the 6th international conference on Learning and Intelligent Optimization,* Paris, France, January 2012, vol. 7219, pp. 55–70.

[152] M. Ballesteros and J. Nivre, “MaltOptimizer: An Optimization Tool for MaltParser,” in *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics,* Avignon, France, April 2012, pp. 58-62.

[153] O. Chapelle, O. Bousquet, and S. Mukherjee, “Choosing Multiple Parameters for Support Vector Machines,” *Mach. Learn.*, vol. 46, no. 1–3, pp. 131–159, 2002.

[154] B. Crawford, R. Soto, E. Monfroy, W. Palma, C. Castro, and F. Paredes, “Parameter tuning of a choice-function based hyperheuristic using Particle Swarm Optimization,” *Expert Syst. Appl.*, vol. 40, no. 5, pp. 1690–1695, Apr. 2013.

[155] J. Zhang *et al.*, “Evolutionary computation meets machine learning: A survey,” *IEEE Computational Intelligence Magazine*, vol. 6, no. 4. pp. 68–75, Nov-2011.

[156] C. Ansotegui, Y. Malitsky, H. Samulowitz, M. Sellmann, and K. Tierney, “Model-based Genetic Algorithms for Algorithm Configuration.” in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence* (IJCAI 2015), Buenos Aires, Argentina, July 2015, pp. 733-739.

[157] M. Mookiah *et al.*, “Evolutionary algorithm based classifier parameter tuning for automatic diabetic retinopathy grading: A hybrid feature extraction approach,” *Knowl. Based Syst.*, vol. 39, pp. 9–22, 2013.

[158] F. Hutter, H. Hoos, and K. Leyton-Brown, “Sequential model-based optimization for general algorithm configuration,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011, vol. 6683, pp. 507–523.

[159] C. Fawcett and H. Hoos, “Analysing differences between algorithm configurations through ablation,” *J. Heuristics*, vol. 22, no. 4, pp. 431–458, August 2016.

[160] M. Birattari, *Tuning metaheuristics: a machine learning perspective*, Illustrate. Springer, 2009.

[161] F. Imbault and K. Lebart, “A stochastic optimization approach for parameter tuning of support vector machines,” in *Proceedings of the 17th International Conference on Pattern Recognition*, Cambridge, UK, Aug. 2004, vol. 4, pp. 597–600.

[162] J. Bergstra, D. Yamins, and D. Cox, “Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures,” in *Proceedings of the 30th International Conference on International Conference on Machine Learning*, Atlanta, Georgia, USA, June 2013, vol. 28, pp. 115-123.

[163] J. Bergstra and Y. Bengio, “Random Search for Hyper-Parameter Optimization,” *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 281–305, Feb. 2012.

[164] A. Samuel, “Some Studies in Machine Learning Using the Game of Checkers,” *IBM J. Res. Dev.*, vol. 3, no. 3, pp. 210–229, Jul. 1959.

[165] J. Snoek, H. Larochelle, and R. Adams, “Practical Bayesian Optimization of Machine Learning Algorithms.” in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, Lake Tahoe, Nevada, USA, Dec. 2012, vol. 2, pp. 2951–2959.

[166] T. Mitchell, *Machine Learning*, 1st Edition. McGraw-Hill, 1997.

[167] T. Hastie, R. Tibshirani, and J. Friedman, "Unsupervised Learning," in *The Elements of Statistical Learning*, Springer, New York, NY, 2009, pp. 485–585.

[168] M. Soysal and E. Schmidt, "Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison," *Perform. Eval.*, vol. 67, no. 6, pp. 451–467, Jun. 2010.

[169] T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Commun. Surv. Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[170] Ian Witten and E. Frank, *Data Mining Practical Machine Learning Tools and Techniques with JAVA Implementations*, Second Edition. Morgan Kaufmann, 2005.

[171] F. Doelitzscher, M. Knahl, C. Reich, and N. Clarke, "Anomaly detection in IaaS Clouds," in *Proceedings of the International Conference on Cloud Computing Technology and Science*, Bristol, UK, Dec. 2013, vol. 1, pp. 387–394.

[172] O. Chapelle, B. Scholkopf, and E. Zien, "Semi-Supervised Learning," *IEEE Trans. Neural Networks*, vol. 20, no. 3, pp. 542–542, Mar. 2009.

[173] M. Peters, W. Ketter, M. Saar-Tsechansky, and J. Collins, "A reinforcement learning approach to autonomous decision-making in smart electricity markets," *Mach. Learn.*, vol. 92, no. 1, pp. 5–39, Jul. 2013.

[174] C. Bishop, *Pattern recognition and machine learning*, vol. 4. Springer-Verlag New York, 2006.

[175] Ponemon Institute LLC and Accenture, "2017 Cost of Cyber Crime Study, Insights on the Security Investments that Make a Difference," 2017.

[176] E. Rosten, R. Porter, and T. Drummond, “Faster and better: A machine learning approach to corner detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 1, pp. 105–119, Jan. 2010.

[177] M. A. Kumar and M. Gopal, “A hybrid SVM based decision tree,” *Pattern Recognit.*, vol. 43, no. 12, pp. 3977–3987, 2010.

[178] C. Ma and X. Wang, “Inductive data mining based on genetic programming: Automatic generation of decision trees from data for process historical data analysis,” *Comput. Chem. Eng.*, vol. 33, no. 10, pp. 1602–1616, Oct. 2009.

[179] P. Refaeilzadeh, L. Tang, and H. Liu, “Cross-Validation,” Arizona State University, 2008.

[180] P. Stone and M. Veloso, “Multiagent Systems: A Survey from a Machine Learning Perspective,” *Auton. Robots*, vol. 8, no. 3, pp. 345–383, 2000.

[181] G. Stiglic, S. Kocbek, I. Pernek, and P. Kokol, “Comprehensive Decision Tree Models in Bioinformatics,” *PLoS One,* vol. 7, no. 3, 2012.

[182] L. Li, H. Zhang, H. Peng, and Y. Yang, “Nearest neighbors based density peaks approach to intrusion detection,” *Chaos, Solitons & Fractals*, vol. 110, pp. 33–40, May 2018.

[183] M. Fernandez-Delgado, E. Cernadas, S. Barro, D. Amorim, and D. Fernandez-Delgado, “Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?,” *J. Mach. Learn. Res.*, vol. 15, pp. 3133–3181, 2014.

[184] R. Samusevich, “Game Theoretic Optimization of Detecting Malicious Behavior,” Czech Technical University in Prague, 2016.

[185] S. Valenti, D. Rossi, A. Dainotti, A. Pescape, A. Finamore, and M. Mellia, “Reviewing Traffic Classification,” Springer, Berlin, Heidelberg, 2013, pp. 123–147.

[186] M. Khorshed, A. Ali, and S. Wasimi, “A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing,” *Futur. Gener. Comput. Syst.*, vol. 28, no. 6, pp. 833–851, Jun. 2012.

[187] R. Bryant, R. Katz, and E. Lazowska, “Big-Data Computing: Creating Revolutionary Breakthroughs in Commerce, Science and Society,” in *Computing Research Initiatives for the 21st Century*. 2008, pp. 1–15, 2008.

[188] J. Mena, *Machine Learning Forensics for Law Enforcement, Security, and Intelligence*. CRC Press, 2011.

[189] M. Masud *et al.*, “Cloud-based malware detection for evolving data streams,” *ACM Trans. Manag. Inf. Syst.*, vol. 2, no. 3, pp. 1–27, Oct. 2011.

[190] V. Nivargi, M. Bhaowal, and T. Lee, “Machine Learning Based Botnet Detection,” *CS 229 Final Proj. Report, Comput. Sci. Dep. Stanford Univ.*, Mar. 2006.

[191] S. Suthaharan, “Big Data classification: problems and challenges in network intrusion prediction with machine learning,” *SIGMETRICS Perform. Eval. Rev.*, vol. 41, pp. 70–73, 2014.

[192] R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *Proceedings of the 14th international joint conference on Artificial intelligence,* Montreal, Quebec, Canada, Aug. 1995, vol. 2, pp. 1137–1143.

[193] J. Kim, “Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap,” *Comput. Stat. Data Anal.*, vol. 53, no. 11, pp. 3735–3745, Sept. 2009.

[194] A. Molinaro, R. Simon, and R. Pfeiffer, “Prediction error estimation: a comparison of resampling methods,” *Bioinformatics*, vol. 21, no. 15, pp. 3301–3307, Aug. 2005.

[195] J. Guan, P. Xiang, and X. Keating, “Evaluating the replicability of sample results: A tutorial of double cross-validation methods,” *Measurement in Physical Education and Exercise Science*, vol. 8, no. 4. Lawrence Erlbaum Associates, Inc., pp. 227–241, Dec-2004.

[196] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning : data mining, inference, and prediction*, 2nd ed. Springer-Verlag New York, 2009.