Publication metadata

Title: Optimal Load Distribution for the Detection of VM-Based DDoS Attacks in the Cloud

Author(s): Omar Abdel Wahab; Jamal Bentahar; Hadi Otrok; Azzam Mourad

Journal: IEEE Transactions on Services Computing

DOI/Link: https://doi.org/ 10.1109/TSC.2017.2694426

How to cite this post-print from LAUR:

Wahab, O. A., Bentahar, J., Otrok, H., & Mourad, A. (2020). Optimal load distribution for the detection of VM-based DDoS attacks in the cloud. IEEE Transactions on Services Computing, DOI, 10.1109/TSC.2017.2694426, http://hdl.handle.net/10725/8317

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

Omar Abdel Wahab, Jamal Bentahar, Hadi Otrok, and Azzam Mourad

**Abstract**—Distributed Denial of Service (DDoS) constitutes a major threat against cloud systems owing to the large financial losses it incurs. This motivated the security research community to investigate numerous detection techniques to limit such attack's effects. Yet, the existing solutions are still not mature enough to satisfy a cloud-dedicated detection system's requirements since they overlook the attacker's wily strategies that exploit the cloud's elastic and multi-tenant properties, and ignore the cloud system's resources constraints. Motivated by this fact, we propose a two-fold solution that allows, firstly, the hypervisor to establish credible trust relationships toward guest Virtual Machines (VMs) by considering objective and subjective trust sources and employing Bayesian inference to aggregate them. On top of the trust model, we design a trust-based maximin game between DDoS attackers trying to minimize the cloud system's detection and hypervisor trying to maximize this minimization under limited budget of resources. The game solution guides the hypervisor to determine the optimal detection load distribution among VMs in real-time that maximizes DDoS attacks' detection. Experimental results reveal that our solution maximizes attacks' detection, decreases false positives and negatives, and minimizes CPU, memory and bandwidth consumption during DDoS attacks compared to the existing detection load distribution techniques.

**Index Terms**—Detection load distribution; Distributed Denial of Service (DDoS); cloud computing; security; trust; game theory; virtualization.

## 1 Introduction

Cloud systems are widely exposed to various types of security threats due to their multi-tenant nature that allows multiple Virtual Machines (VMs) owned by different (possibly malicious) clients to share a single physical infrastructure. Distributed Denial of Service (DDoS) constitutes one of the most widespread and painful attacks for both cloud providers and clients. Recently published reports reveal that a large number of well-known cloud providers, including Amazon EC2 and Rackspace, faced in the recent past years massive DDoS attacks resulting in financial losses amounting to tens of thousands of dollars [1].

**Problem Statement.** Several Intrusion Detection Systems (IDSs) [2], [3], [4], [5], [6], [7] have been advanced to identify intrusions in cloud environments, where most of these systems are developed and improved from traditional detection techniques used in non-cloud environments. Such systems can be categorized into three main branches: network-based, host-based, and hypervisor-based systems. In network-based IDSs, the incoming/outgoing network traffic along with the packets' content are monitored and analyzed to recognize intrusions. Although such an approach might be effective in capturing outsider attacks that occur at the network's level, it is ineffective however in detecting the insider attacks in which attackers infiltrate into the internal cloud system. To cope with this limitation, host-based IDSs propose to integrate a monitoring agent inside each VM to monitor its behavior and states (e.g., system calls) and be able hence to recognize any abnormal behavior. This aids host-based IDSs in capturing both insider and outsider attacks. Yet, the use of such systems entail supplementary responsibilities for VMs' owners in terms of managing the monitoring agents, which disadvantages the adoption of host-based systems in a cloud environment wherein a client might possess multiple VM instances.

To alleviate these responsibilities, hypervisor-based IDSs move the intrusion detection responsibilities from the VM's level to the cloud system's level by placing the monitoring agent at the hypervisor's layer. This enables the hypervisor to examine the VMs' system metrics (e.g., CPU usage) directly from the hosting infrastructure to recognize any malicious behavior. However, such a process demands storing and analyzing a huge number of events from each single VM in real-time, which entails significant storage and computational overheads. The situation becomes even worse when a single attack is being distributed across several VMs (as is the case in DDoS attacks) since this requires analyzing and correlating events from different VMs in order to identify one single attack. Moreover, all of the network-based, host-based, and hypervisor-based IDSs stop at the borders of monitoring and analyzing events to identify intrusions. Thus, these systems consider the detection problem from the perspective of the IDS only without accounting for the sophisticated strategies of the attackers who strive to complicate their attacks to minimize their detection chances. In summary, the main limitations of the existing cloud-based detection systems can be summarized by the following:

- The design of the existing detection systems accounts only for the IDS agent's perspective and overlooks the attacker's wily strategies. This gives the attacker higher chances of exploiting the cloud's elastic and multi-tenant properties in order to complicate the detection of the launched attacks.
- The cloud system's resources (e.g., CPU) constraints are ignored in the design of the existing detection systems, which limits their efficiency in today's large-scale applica-

- *O. Abdel Wahab and J. Bentahar are with the Concordia Institute for Information Systems Engineering, Concordia University, Montréal, Canada.*
- *H. Otrok is with the Department of ECE, Khalifa University of Science, Technology and Research, Abu Dhabi, UAE.*
- *A. Mourad is with the Department of Mathematics and Computer Science, Lebanese American University, Beirut, Lebanon.*

tions.

This raises the need for a resource-aware detection mechanism that can maximize the detection of DDoS attacks under a limited amount of resources. In a preliminary version of this work [8], we proposed an intelligent technique that guides the cloud system on the optimal distribution of detection load among VMs in such a way that maximizes the detection of generic distributed attacks. This paper builds on and extends our previous work by (1) considering a concrete DDoS attack scenario in the formulation of the problem and solution; (2) elaborating a trust model that allows the hypervisor to build trust relationships toward its VMs; and (3) incorporating trust as a building block factor in the maximin game to optimize the process of detection load distribution. Moreover, we compare the two versions experimentally to verify the improvements brought to the work by our new amendments.

**Contributions.** The objective of this work is to develop an offline detection load distribution strategy that enables the hypervisor, based on the trust relationships it builds toward VMs, to learn about the optimal detection load percentage that should be allocated to each of its guest VMs in real-time. The purpose is to maximize the detection of DDoS attacks under a limited amount of resources. To attain this objective, we propose first a trust framework that enables the hypervisor to construct trust relationships toward guest VMs. To ensure building credible relationships, we combine both the subjective and objective sources of trust. In the first place, the hypervisor monitors and analyzes offline the CPU, memory, and network bandwidth consumption of each VM and employs the interquartile statistical technique [9] to detect any abnormal behavior. This enables the hypervisor to establish a prior (objective) trust belief toward each of its VMs. However, since the results of such a monitoring process might be biased towards the properties of a certain cloud infrastructure or towards a certain period of time (e.g., promotion times), the hypervisor resorts then to the subjective source of trust by collecting recommendations from other hypervisors/VMs that have past interactions (e.g., compositions, hosting) with the VMs being judged. The objective (monitoring) and subjective (recommendations) sources are then aggregated using the Bayesian inference theory [10] to come up with posterior final trust scores. We discuss as well a trust bootstrapping mechanism that allows the hypervisor to allocate initial trust scores for the newly deployed VMs for which no historical data about their former behavior can be found.

On top of the proposed trust framework, we design a resource-aware trust-based maximin game between the hypervisor and DDoS attackers that incorporates the trust scores of the VMs into the game formulation to optimize the hypervisor's decisions. The strategy of the attackers is to select a probability distribution for each attack over a set of VMs in such a way to minimize the hypervisor's probability of detection (e.g., the VMs that the attacker thinks will be the least monitored). On the other hand, the strategy of the hypervisor is to select a probability distribution for the available detection load over the set of VMs in order to maximize the attacker's minimization. The game is converted then into a Linear Programming problem and solved using the simplex method [11]. The outcome of the game is a probability distribution over the VMs informing the hypervisor about the optimal percentage of detection load that should be placed on each of its guest VMs in real-time. In summary, the main contributions of this paper are highlighted in the following:

- Developing a trust model between the hypervisor and its guest VMs that uses objective and subjective sources of trust to optimize the credibility of the trust scores. To the best of our knowledge, our work is the first in the domain of cloud computing that investigates the trust relationships between the cloud system and VMs.
- Designing and solving a trust-based maximin game between the hypervisor and DDoS attackers. The solution provides the hypervisor with the optimal detection load distribution strategy over VMs that maximizes the detection of DDoS attacks under a limited budget of resources. We believe that such a (yet not available) detection load distribution strategy would substantially advance the state-of-the-art in distributed cloud-based IDSs.

We provide as well a complete numerical example that explains how our proposed solution can be practically applied in real-life applications. The performance of our solution is evaluated experimentally using the CloudSim simulator [12] that helped us create a cloud data center mimicking the Amazon public data-center in terms of VMs' configuration (inspired by Amazon EC2 X-large instances[1]) and pricing scheme (derived from the Amazon pricing model[2]). Moreover, the trust scores of the VMs have been populated from the Epinions trust dataset[3]. Experimental results reveal that our model maximizes the DDoS attacks detection and minimizes the false positive and negative rates. Moreover, our solution mitigates DDoS attacks' impact by minimizing the CPU, memory, and network bandwidth consumption in the presence of such attacks, while being efficient in terms of execution time.

**Paper Organization.** Section 2 presents a literature review on the IDSs proposed for cloud computing and highlights the unique features of our work. Section 3 formulates the problem and discusses the attack model. Section 4 explains the details of the trust model between the hypervisor and VMs. Section 5 describes our trust-based maximin game between the hypervisor and attackers and derives the solution of the game. Section 6 provides a numerical example that demonstrates how our solution can be effectively applied in practical scenarios. Section 7 explains the experimental setup and presents experimental results. Finally, Section 8 concludes the paper.

## 2 Related Work

In the section, we explain the main contributions in each of the aforementioned branches of IDSs and highlight the unique features of our solution. We shed light as well on the main trust models proposed for cloud-based environments and stress the original aspects of our trust framework.

### 2.1 Network-based Detection Systems

In [13], the authors propose a cooperative IDS for DoS attacks. They assume that an IDS is deployed in each cloud region to collect and analyze network packets. If the type of the packet matches any type defined in the block table (that maintains the bad packets to be blocked), then this packet is immediately dropped by the IDS. If no match exists but the packet is categorized as anomalous, then the degree of severity is checked. If the packet is classified as serious, then the IDS drops it and notifies the other IDSs accordingly. If the packet is classified as moderate, the

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

IDS performs data clustering and threshold check to find outliers and updates the alert level accordingly. Finally, if the packet is identified as slight, then the system simply ignores the alert. In [6], the authors address the DoS attack in cloud environment by proposing a scheme for tracing back the botmaster (i.e., the malicious user that administrates the botnets). To this end, the local network administrator of the victim machine collects information (i.e., memory images, network traffic between bots and Command-and-Control (C&C) servers, and hostname of the C&C server), files them to a traceback server, and asks the latter for a traceback service. The traceback server embeds then Pebbleware, a piece of code that reveals its host machine's information, on the communication packets from the victim node to the botmaster. Once the Prebbleware reaches the botmaster, the latter's machine is obliged to send its IP address to the traceback server.

To sum up, network-based IDSs consist of the idea of monitoring the incoming and outgoing network traffic to identify intrusions. This makes them effective in detecting outsider attacks but unsuccessful in identifying internal attacks in which attackers are part of the internal cloud system.

## 2.2 Host-based Detection Systems

The authors in [7] advance a distributed detection system for identifying DDoS attacks in the cloud. For this purpose, they propose to equip each VM with an IDS that monitors and collects alerts. These alerts are then converted into basic probabilities assignments and analyzed using Dempster-Shafer [14]. Ward and Barker [5] propose a multi-tier IDS called *Varanus* that operates at the Infrastructure as a Service (IaaS) layer. First, the *k*-nearest neighbor algorithm is used to split VMs into a set of clusters on the basis of the similarity between their software configuration features (e.g., database servers, web servers). Within each group, VMs launch a gossip-based monitoring process by exchanging relevant information (e.g., memory usage) and then the under-utilized VMs are selected to analyze the collected data. Lastly, each group's aggregate value is propagated to the other groups.

Summarizing, host-based IDSs deploy a monitoring agent inside each VM to watch its behavior and recognize any abnormal behavior. This makes them more successful than network-based systems in dealing with both internal and external attacks. Yet, host-based IDSs entail extra overhead and management responsibilities for VMs' owners who are required to use their own resources to manage the monitoring agents. Moreover, such systems can be manipulated by experienced attackers who compromise the VM instances and tamper the monitoring agents [8].

## 2.3 Hypervisor-based Detection Systems

Lombardi and Di Pietro propose in [2] a virtualization-supported security architecture whose main purpose is to ensure the integrity of the VMs while being invisible to end users. To this end, an *Interceptor* entity is deployed into the kernel space of the host system to constantly monitor the VMs' system-call invocations. Thereafter, a *Warning Recorder* entity registers the suspicious activities in a *Warning Pool* whose responsibility is to prioritize the evaluation order of these activities. The *Warning Recorder* derives checksums for code, data, and files and passes them to the *Evaluator* entity that inspects the activities and takes the appropriate decision on whether the system's security has been violated or not. In [3], the authors propose a Virtual machine Intrusion Detector (VICTOR) for protecting customers' VMs from different attacks at the IaaS cloud layer. The main purpose of VICTOR is to recognize the malicious entities that generate the attack flow and dynamically isolate them. The proposed model is able to distinguish between traffic produced from each VM even when several VMs share a single IP address. In [4], a hypervisor-based detection mechanism is proposed, where the VMs' performance metrics (e.g., block device read/write data, CPU usage) are retrieved every second by endpoint agents deployed at the hypervisor's layer. Collected data is conveyed then to a controller node that analyzes it against stored signatures to confirm whether there exists an attack or not.

To summarize, the existing IDSs stop at the borders of monitoring and analyzing events to identify intrusions. Thus, these systems consider the detection problem from the perspective of the IDS only without accounting for the sophisticated strategies of the attackers. In our previous work [8], we have developed a maximin game that is able to determine the optimal detection load distribution over VMs, while being aware of the attackers' strategies who distribute their attacks over several VMs to complicate the detection process. This work considers the prices of the VMs as the main factor when designing the utility functions. The idea is to let the hypervisor pay more attention to the worthy VMs whose violation incur painful losses for both providers and users. In this paper, we extend this work by (1) considering a practical DDoS attack scenario; (2) designing a trust model that allows the hypervisor to establish trust relationships toward its VMs; and (3) integrating trust as a building block factor in the maximin game with the aim of optimizing the detection load distribution process.

We argue that introducing trust in the maximin game would contribute in optimizing the detection load distribution process. In fact, such a trust model allows the hypervisor to learn about the behavior of the VMs over the time and enables it hence to adjust the detection load distribution strategy in such a way that assigns more load to the VMs that have a large number of misbehavior during their past history. Besides, the fact that trust is a private relationship between the hypervisor and VMs reduces the possibility of the attacker to predict the potential detection load distribution strategy that will be adopted by the hypervisor. Practically, trust in our case is simply a confidence degree believed by the hypervisor on the VMs' behavior in accomplishing their tasks smoothly without making malicious actions. Thus, the hypervisor is the sole party that will be aware of its (internal) trust beliefs. Moreover, the fact that trust scores in our model are not supposed to be communicated to any other party (other than the hypervisor) eliminates the risks of eavesdropping and/or altering them. The only threat that may arise against trust values would be compromising the hypervisor itself and sniffing the trust scores at the storage level. To remedy this vulnerability, cloud-dedicated cryptographic solutions [15] might be employed. In contrary, price is publicly available for all the users, which enables the attacker to anticipate, for example, that the hypervisor would dedicate more detection load to the more valuable VMs and adjust hence its attack distribution strategy accordingly. We compare, in Section 7, our solution against this price-based model [8] experimentally to validate the aforementioned claims.

## 2.4 Trust Models in Cloud Computing

Trust has been widely investigated in the context of cloud computing, where the current proposals have been focusing mainly on building trust relationships between cloud providers and customers. In [16], the authors advanced a trust model that helps

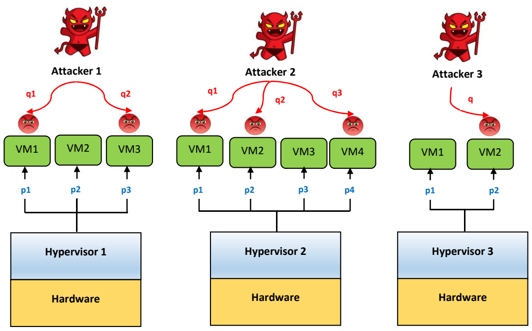


Fig. 1: Attack Scenario: Attackers distribute their attacks over a set of malicious VMs to minimize the detection probability, while hypervisors distribute the detection load over the set of guest VMs to maximize this minimization.

clients assess cloud services based on the Service-Level Agreement (SLA) criteria to aid these clients in the process of selecting the most reliable cloud resources. Similarly, the authors in [17] discussed a trust model that accounts for four metrics such as availability, reliability, turnaround efficiency, and data integrity to help users build trust values toward cloud resources. Finally, a multi-faceted trust management system is discussed in [18] to assist customers with identifying the trustworthiness of cloud providers on the basis of various parameters such as performance, security, and compliance.

Overall, the existing trust models seek mainly to regulate the relationships between cloud providers and customers, without accounting for the intra-cloud trust relationships that should be established among the cloud's system components. This makes our work the first in the domain of cloud computing that investigates the trust beliefs among the cloud system and its guest VMs. Moreover, our trust framework has been designed in such a way that overcomes the limitations of the existing trust models identified in a previous survey work [19]. Particularly, our trust framework combines objective and subjective trust sources to optimize the credibility of the trust results and overcome the collusion attacks. We discuss as well a trust bootstrapping mechanism that helps us solve the challenging problem of assigning initial trust scores for the newly deployed VMs.

# 3 System Model and Assumptions

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

## 3.1 System Model and Strategies

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

***Definition 1 (Virtualized Cloud System).*** A virtualized cloud system consists of a set of hardware resources $I = \{I_1, ..., I_n\}$ managed by a set hypervisors $H = \{h_1, \ldots, h_n\}$; where each hypervisor $h$ hosts a set of VMs $V = \{v_1, v_2, \ldots, v_l\}$ owned by a set of clients $C = \{c_1, \ldots, c_m\}$ to provide each $v \in V$ with a view that its OS and applications are operating directly on some physical hardware.

As a first stage, the hypervisor seeks to establish trust relationships toward its guest VMs. To do so, it first monitors and analyzes the CPU, memory, and network bandwidth utilization of each $v \in V$. This allows the hypervisor $h$ to build an initial belief in each $v$'s trustworthiness denoted as $InitialBelief_h^v$. The hypervisor collects then recommendations from other VMs/hypervisors on the behavior of the underlying VMs. In the rest of this section, we abstract on the identity of recommenders and refer to both VMs and hypervisors as *source*. Each recommendation $R_s^v \in [0,1]$ denotes a certain source $s$'s recommendation on the behavior of a VM $v$ based on their previous interactions. Note that each source $s$ enjoys a fixed number of inquiries it is allowed to make from every (other) hypervisor $h'$ and is denoted by $Inq(s \rightarrow h')$. Initially, all sources enjoy an equal amount of inquiries, where this amount is updated later during the trust establishment process (See Section 4). Now, the hypervisor $h$ aggregates the results of the monitoring phase with the results of the recommendations phase using the *Bayesian inference* technique to come up with a final belief $Belief_h^v$ in each $v$'s trustworthiness (See Section 4.3).

Having computed the final trust scores for the VMs, the hypervisor integrates these scores into its utility function (see Section 5). The objective is to benefit from the computed trust scores to find the optimal distribution of the detection load among its set of guest VMs that maximizes the attacks detection probability, knowing that DDoS attackers are distributing their attacks over a set of VMs to minimize this maximization.

## 3.2 Attacker Strategy

In order to complicate the detection process and rip off the hypervisor, attackers can use a mixed strategy by distributing a single DoS attack over multiple VMs running on top of the same hypervisor. To this end, the attacker splits its attack code into several malicious fragments and assigns a set of fragments to each VM. Each malicious VM aims at sending $k$ malicious fragments to the hypervisor at different time intervals. Let $Q_V = (q(v_1), ..., q(v_l))$ denote the probability distribution vector of the attacks over the set $V$ deployed on hypervisor $h$ such that $\sum_{v \in V} q(v) = 1$. The attack

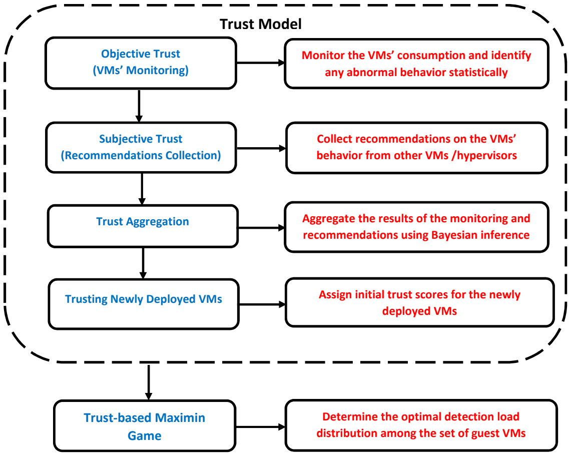

Trust Model
Objective Trust (VMs' Monitoring)
Monitor the VMs' consumption and identify any abnormal behavior statistically
Subjective Trust (Recommendations Collection)
Collect recommendations on the VMs' behavior from other VMs /hypervisors
Trust Aggregation
Aggregate the results of the monitoring and recommendations using Bayesian inference
Trusting Newly Deployed VMs
Assign initial trust scores for the newly deployed VMs
Trust-based Maximin Game
Determine the optimal detection load distribution among the set of guest VMs

Fig. 2: Solution Methodology

TABLE 1: Notations

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

succeeds if one or many malicious fragments attain the hypervisor without being detected.

***Definition 2 (Distributed Attack).*** A distributed attack is a set of $k$ malicious fragments $\{f_1,...,f_k\}$ distributed over $V$ with probability of $q(v)$ for each $v \in V$ such that $\sum_{v \in V} q(v) = 1$.

### 3.3 Hypervisor Strategy

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

For the readers' convenience, a graphical formulation of the above-mentioned problem is given in Fig. 1 and the methodology followed to perform our solution is schematized in Fig. 2. Moreover, Table 1 defines and summarizes the different notations that are used throughout the paper.

### 3.4 Attack Model

We consider in this work a DDoS attack scenario wherein attackers are a group of VMs targeting a particular cloud system. Although DDoS attacks in a cloud environment may take many forms and can be seen in different contexts (e.g., application, web services, network, etc.), we focus in this work on the DDoS attacks that occur at the virtualization layer between the hypervisor and its guest VMs (i.e., VM-based DDoS). Particularly, we study the case in which attacking VMs try to flood the victim cloud system in such a way that makes it unavailable to support further VMs. These attacking VMs may be either malicious or compromised by a malicious attacker to serve as bots in the DDoS attack process[4]. To perform their attacks, DDoS attackers benefit particularly from the *auto-scaling* (a.k.a *elasticity*) property that is provided by the virtualization technology as an appealing added-value feature to the cloud computing systems. Specifically, auto-scaling enables the cloud to keep assigning extra resources to the VMs that are in need for additional resources. Fortunately, such a property leads to enhance the cloud system's performance thanks to the fact that a VM will never undergo resource outages as long as the VM's owner agrees to keep paying bills versus receiving additional resources. Unfortunately, this property is being appealingly manipulated by DDoS attackers who compromise VMs and keep sending, through these compromised VMs, fake resources scaling requests. This allows the attacker to achieve the two following malicious objectives: (1) draining the resources of the cloud system to make it unable to support further VMs; and (2) increasing the bill of the VMs' real owners by obliging them to pay

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

for (supposedly) unrequested resources (a.k.a Economic Denial of Stainability (EDoS)) [1].

## 4 Building Trust on Virtual Machines

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

### 4.1 Objective Trust: Virtual Machines Monitoring

In this phase, the hypervisor monitors the VMs' CPU, memory, and network bandwidth consumption directly from the hosting infrastructure and applies the Interquartile Range (IQR) statistical measure to identify any abnormal usage. This constitutes the objective source of trust which is of prime importance in the field of trust and reputation to avoid biased and/or subjective judgements [19]. The IQR is a measure of variability whose basic idea is to split a given set of data into disjoint quartiles (i.e., $Q_1$, $Q_2$, $Q_3$).

The first quartile $Q_1$ corresponds to the value in the data set that 25% of the values are smaller than it. The second quartile $Q_2$ stands for the data set's median value. The third quartile $Q_3$ represents the value in the data set that 25% of the values are higher than it. The IQR is obtained then by subtracting the first quartile from the third quartile. The reasons for choosing the IQR measure for the considered problem lie in its (1) robustness to messy data and outliers, and (2) simple and lightweight nature that imposes no heavy computation efforts on the hypervisor [9].

The algorithm of this phase that is executed by the hypervisor is depicted in Algorithm 1. Having monitored and recorded the CPU, memory, and network bandwidth consumption of the VM in question at time $t$ (i.e., for the time window $[t-\mu,t]$), the hypervisor computes, for each of these metrics (e.g., CPU), the median usage of the VM (step 20). It finds then, based on the computed median, the first and third quartiles $Q_1$ and $Q_3$ for each metric respectively (steps 21-22). Using these quartiles, the IQR is computed by subtracting $Q_3$ from $Q_1$ and multiplying the obtained value by 1.5 (step 23). Intuitively, this means that any value lying more than one and a half times beyond the upper quartile is considered to be an outlier according to Tukey analysis [20]. By adding the IQR to the third quartile, the hypervisor computes the upper consumption limit for each underlying metric (step 24). Intuitively, this limit represents the pattern of maximal habitual utilization of the VM at a certain time period; where any future utilization above this limit would be considered unusual. The hypervisor checks then for any future consumption of the VM at time $t+\mu$ whether there exists any consumption that exceeds the computed upper limit (lines 25-26). If so, this event is appended to a table that stores the VM's unusual consumption (line 27) and the average of unusual consumption for each metric is computed (line 32). The hypervisor computes finally its initial belief in the VM's trustworthiness by dividing the sum of average unusual consumptions over all the metrics by the number of metrics that the VM has overconsumed, if any (line 40). If no metric has been overconsumed, the initial belief in the VM's trustworthiness would be set to 1 (line 38), which represents a full initial trust in the VM. Note finally that the whole process is repeated periodically after a certain period of time $\mu$ to continuously capture the dynamism in the VMs' performance and behavior.

**Algorithm 1:** Virtual Machines' Monitoring

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

17: procedure VMMonitoring
18:     repeat
19:         for each metric $x \in U$ do
20:             Compute the median $M_v^x(t)$ of $U_v^x(t)$
21:             Find $Q1_v^x(t)$ as the median of $U_v^x(t)$'s lower half
22:             Find $Q3_v^x(t)$ as the median of $U_v^x(t)$'s upper half
23:             Compute $IQR_v^x(t) = (Q3_v^x(t) - Q1_v^x(t)) \times 1.5$
24:             Compute $L_v^x(t) = IQR_v^x(t) + Q3_v^x(t)$
25:             for each data point $y \in U_v^x(t+\mu)$ do
26:                 if $y > L_v^x(t)$ then
27:                     $OverUse_v^x = OverUse_v^x + y$
28:                     $CountOverUse_v^x = CountOverUse_v^x + 1$
29:                 end if
30:             end for
31:             if $CountOverUse_v^x > 0$ then
32:                 $AvgOverUse_v^x = OverUse_v^x / CountOverUse_v^x$
33:                 $PropOverUse_v^x = L_v^x(t) / AvgOverUse_v^x$
34:                 $|OverusedMetrics| = |OverusedMetrics| + 1$
35:             end
36:         end for
37:         if $|OverusedMetrics| = 0$ then
38:             $InitialBelief_h^v = 1$
39:         else
40:             $InitialBelief_h^v = \frac{\sum_{x \in U} PropOverUse_v^x}{|OverusedMetrics|}$
41:         end
42:     until $\mu$ elapses
43: end procedure
```

For example, suppose that the CPU, memory, and bandwidth upper consumption limits for a VM $v$ were 60%, 70%, and 50% respectively. Suppose as well that $v$ has been overconsuming the CPU with an average of 73%, the memory with an average of 73%, but has not been overconsuming the bandwidth metric. Then, the proportional overuse of CPU and memory would be calculated respectively as follows (Algorithm 1 - line 33): $PropOverUse_v^{CPU} = 60/73 = 0.822$ and $PropOverUse_v^{Memory} = 70/73 = 0.959$. The initial hypervisor's belief in $v$'s trustworthiness would then amount to: $InitialBelief_h^v = \frac{0.822+0.959}{2} = 0.8905$. Note that we have divided by 2 since only two metrics, namely the CPU and memory have been overconsumed by $v$. Consider now another case wherein $v$ was overconsuming the CPU with an average of 95% and the memory with an average of 98%. Then, the proportional overuse of CPU and memory would be calculated respectively as follows:

$PropOverUse_v^{CPU} = 60/95 = 0.631$ and $PropOverUse_v^{Memory} = 70/98 = 0.714$. The initial hypervisor's belief in $v$'s trustworthiness would then amount to: $InitialBelief_h^v = \frac{0.631+0.714}{2} = 0.6725$. We notice from the two examples that as the overconsumption keeps going far from the upper consumption limit, the initial trust score keeps decreasing (i.e., $0.6725 < 0.8905$).

### 4.2 Subjective Trust: Recommendations Collection

In order to enhance the quality of the trust scores, the hypervisor collects recommendations $R_{s_1}^v, \ldots, R_{s_n}^v$ (where $R_{s_i}^v \in [0,1]$) on the former behavior of the VMs. This constitutes the subjective source of trust and is widely used in the context of trust and reputation thanks to the fact that it consults different parties' opinions to improve the quality of the judgements [19]. The source of the recommendations may be either VM(s) having dealt with the VM in question or other hypervisor(s) having previously hosted that VM. The recommendations obtained from the former source (i.e., VMs) are important to learn about the performance of the VMs in cases of services cooperation or composition in the cloud. The recommendations obtained from the latter source (i.e., hypervisors) allow us to capture the dynamism in the VMs' performance on different cloud infrastructures, which alleviates the risk of misjudging VMs because of bad performance of the host cloud system. This contributes in enhancing the detection accuracy under a changing cloud infrastructure environment. The recommendations are derived based on the overall behavior of the VMs, not only based on the metrics related to DDoS attacks. For example, if a certain VM is not launching a DDoS attack but launching a side-channel attack [21], then it should receive low recommendation scores. We argue that obtaining such recommendations is becoming easier with the emerging cooperation architectures that are being proposed and adopted for cloud-based services such as services' communities and cloud federations [14], [22]. Practically, such architectures allow services, coming either from one cloud or deployed even in different cloud centers, to cooperate with one another in order to improve the performance and security of the underlying system. In this way, services (in the form of VMs) are allowed to easily migrate from one cloud infrastructure to another as a result of the community/federation agreement contract. In the same context, the VMs that are grouped in the same community/federation are likely to cooperate with one another to better respond to customers' requests. Therefore, obtaining recommendations from both hypervisors and VMs is becoming simpler and more realistic.

### 4.3 Trust Aggregation

Having obtained both the objective and subjective sources of trust, the next step is to aggregate these sources and come up with final aggregate trust scores for the VMs. For this purpose, we employ the Bayesian inference from the *subjective probability theory* [10]. Bayesian inference is a theory that describes uncertainty using a probability distribution. In simple words, assume that a person has an uncertainty about an issue. This uncertainty may be depicted using a probability distribution known as that person's *prior distribution*. Assume now that this person has been able to gain some information pertinent to that issue. The information evolves his uncertainty, which may be then represented as a new probability distribution called *posterior distribution*. This posterior distribution reflects the knowledge obtained from both the prior distribution and new information. The main function of Bayesian inference lies in the process of moving from prior to posterior distribution. In our case, the prior distribution represents the hypervisor's initial beliefs about VMs obtained from the monitoring process described in Algorithm 1 prior to collecting recommendations. This allows us to overcome a substantial problem of Bayesian inference caused by the arbitrary choices of the initial prior beliefs, where such uninformative prior beliefs have a great negative impact on the precision of the posterior final beliefs [23]. Once the recommendations are gathered, the prior distribution is converted into a posterior distribution that reflects the updated hypervisor's beliefs after analyzing the received recommendations. This is done by employing the *conditional probability* laws often referred to as *Bayes theorem* [24].

By applying the *Bayes theorem* for aggregating the objective and subjective trust sources, we get that the Bayesian estimation of the trust belief in a VM $v$ with $n$ recommendations $R_{s_1}^v, \ldots, R_{s_n}^v$ (where $R_{s_i}^v \in [0,1]$) is given by:

$$Belief_h^v = \sum_{i=1}^{n} \frac{R_{s_i}^v + n\gamma}{2n}, \tag{1}$$

where $\gamma = InitialBelief_h^v$ represents the hypervisor $h$'s prior belief in $v$'s trustworthiness, $R_{s_i}^v$ denotes a recommendation given on $v$ by a source $s_i$, and $n$ is the total number of collected recommendations. Note that a similar aggregation function has been used in [25], [26] to compute trust values for Web services willing to participate in composition processes.

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

$$Inq(s \to h') = \begin{cases} \lceil \frac{Inq(s \to h')}{diff \times \alpha} \rceil, & \text{if } Inq(s \to h') > 0 \\ \lceil \frac{1}{diff \times \alpha} \rceil, & \text{otherwise} \end{cases} \tag{2}$$

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

### 4.4 Trusting Newly Deployed VMs

Building trust relationships toward the VMs that are newly deployed in cloud centers constitutes evidently a serious obstacle against our proposed trust mechanism. Indeed, the absence of any historical and actual information that corroborates the performance of such VMs makes it quite difficult to compute trust scores for them. This raises the need for a mechanism enabling the hypervisor to assign initial trust values for the newly deployed VMs in the absence of any historical and current data. Such a problem is referred to as a trust bootstrapping problem [14]. To handle this issue, we capitalize on the trust bootstrapping mechanism proposed in our previous work [14] to form trustworthy multi-cloud services communities and that combines the concept of endorsement in online social networks (e.g., LinkedIn) with the decision tree classification technique to solve the problem. We borrow the overall logic used in that mechanism, while performing some technical updates to adapt it to our studied problem. The idea is explained in the following.

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

The endorsements from the different bootstrappers are then aggregated using the Bayesian inference equation (Eq. (1)) in order to avoid biased endorsements. In this case, the prior beliefs in all the newly deployed VMs would be all set to $\frac{1}{2}$ (i.e., $\gamma = \frac{1}{2}$), where this expresses a neutral belief between trust and distrust. Finally, bootstrappers receive payments from the boostrapping requestor for having helped it construct trust beliefs. The payment for bootstrappers is given again in terms of additional inquiry requests they can make from the bootstrapping requestor as per Eq. 2. This payment mechanism is important to (1) stimulate the hypervisors/VMs that enjoy high classification accuracy rates to get involved in the bootsrapping process so as to receive payments; and (2) discourage the malicious hypervisors/VMs from offering bogus endorsements to illegally promote/demote some VMs.

## 5 Determining the Optimal Detection Load Distribution Strategy: Trust-based Maximin Game

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

$$U_{t_2+1}(h) = \sum_{v\in V} \frac{W(v)\times \beta_{[t_1,t_2]}}{Belief_h^v}, \tag{3}$$

where $W(v)$ represents the worth of each virtual machine $v$ (e.g., price, criticality of the applications running on it), $Belief_h^v$ denotes the belief of $h$ in $v$'s trustworthiness, and $\beta_{[t_1,t_2]}$ is the average detection rate of the IDS agent running on $h$ during the time window $[t_1,t_2]$ and is computed as per Eq. (4).

$$\beta_{[t_1,t_2]} = 1 - \sum_{x=t_1}^{t_2}\sum_{v\in V}\frac{(q_x(v)-p_x(v))}{t_2-t_1} \text{ for each } q_x(v) > p_x(v), \tag{4}$$

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

The idea behind dividing by the trustworthiness belief in the utility function is to make the utility of the hypervisor increase when the belief in a certain VM's trustworthiness decreases and vice versa. In this way, the hypervisor would pay more attention to those VMs that it believes are less trusted when deciding about the optimal detection load distribution strategy. This adds a learning component to the game and aids the hypervisor hence to optimize its detection load distribution strategy. It is worth mentioning that all the calculations in the rest of the paper are done at time $t_2+1$ (i.e., the current time for the hypervisor). Thus, we simplify the notation and use $U(h)$ instead of $U_{t_2+1}(h)$ when referring to hypervisor's $h$ utility at time $t_2+1$. The payoff of the attacker $a$ represents the loss incurred to the hypervisor as a result of a successful attack. Therefore, the payoff of the attacker is the negation of the hypervisor's payoff, i.e.,

$$U(a) = -U(h) \tag{5}$$

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

***Definition 3 (Hypervisor-attacker Zero-sum Game).*** A hypervisor-attacker zero-sum game is a tuple $G = \langle h, a, P_V, Q_V, U(h)\rangle$, where:

- $h$: denotes the hypervisor (i.e., the first player).
- $a$: denotes the attacker (i.e., the second player).

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

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

$$\arg\min_{Q_V} U(h) \tag{6}$$

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

$$\arg\max_{P_V} \min_{Q_V} U(h) \tag{7}$$

This forms a maximin game wherein the attacker tries to minimize the hypervisor's probability of detecting his attacks by distributing each attack over multiple VMs, whereas the hypervisor tries to maximize this minimization by choosing the optimal distribution of detection load over the VMs.

***Definition 4 (Hypervisor's Maximin Strategy).*** The maximin strategy for the hypervisor $h$ is $\arg\max_{P_V} \min_{Q_V} U(h)$ and the maximin value for $h$ is $\max_{P_V} \min_{Q_V} U(h)$.

The solution of the game can be devised using Linear Programming (LP), referred to as the problem of determining the values of some real variables for the purpose of minimizing or maximizing a linear function (the objective function) subject to linear constraints on these variables. To this end, let us consider the problem first from the point of view of the hypervisor trying to maximize the minimum of the attacker and let us rewrite Eq. (7) as follows:

$$\begin{aligned} \text{maximize} \quad & \min_{Q_V} \sum_{v \in V} p(v) \times U(h) \\ \text{subject to} \quad & \sum_{v \in V} p(v) = 1, \\ & p(v) \ \geq \ 0, \ for \ all \ v \in V. \end{aligned} \tag{8}$$

By inspecting Eq. (8), we can notice that the objective function is not linear in the $p$'s owing to the presence of the *min* operator. Therefore, the problem in its current form cannot be solved using LP. To linearize it, we define a variable $f$ such that $f \leq \min_{Q_V} \sum_{v \in V} p(v) \times U(h)$ and try to make $f$ as large as possible subject to this new constraint. Thus, the problem is turned into choosing $f$ and $\sum_{v \in V} p(v)$ to:

$$\begin{aligned} \text{maximize} \quad & f \\ \text{subject to} \quad & f \leq \sum_{v \in V} p(v) \times U(h), \\ & p(v_1) + \cdots + p(v_l) = 1, \\ & p(v) \ \geq \ 0, \ for \ all \ v \in V. \end{aligned} \tag{9}$$

Intuitively, this means that the hypervisor, by choosing its mixed strategy $p(v) \in P_V$ , is trying to make as large as possible the minimum that the attacker is attempting to inflict by playing his mixed strategy $q(v) \in Q_V$. To ease the computations, we transform the LP presented in Eq. (9) into a simpler form. Assume that $f > 0$ and let $x(v) = \frac{p(v)}{f}$. The constraint $p(v_1) + \cdots + p(v_l) = 1$ becomes then $x(v_1) + \cdots + x(v_l) = 1/f$. Since maximizing $f$ is equivalent to minimizing $f$'s reciprocal $1/f$, we can get rid of $f$ in our problem by rather minimizing $x(v_1) + ... + x(v_l)$. Thus, the problem becomes: choose $x(v_1) + \cdots + x(v_l)$ to:

$$\begin{aligned} \text{minimize} \quad & x(v_1) + \cdots + x(v_l) \\ \text{subject to} \quad & 1 \leq \sum_{v \in V} x(v) \times U(h), \\ & x(v) \ \geq \ 0, \ for \ all \ v \in V. \end{aligned} \tag{10}$$

The above problem may be solved in polynomial time using the simplex method for solving Linear Programming, which is known for its fast performance [11]. Having solved the problem, the hypervisor's optimal strategy would be $p(v) = f \times x(v)$ for each $v \in V$.

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

***Definition 5 (Attacker's Minimax Strategy).*** The minimax strategy for the attacker $a$ is $\arg\min_{Q_V} \max_{P_V} U(h)$ and the minimax value for $a$ is $\min_{Q_V} \max_{P_V} U(h)$.

The problem can be written as follows:

$$\begin{aligned} \text{minimize} \quad & \max_{P_V} \sum_{v \in V} q(v) \times U(h) \\ \text{subject to} \quad & \sum_{v \in V} q(v) = 1, \\ & q(v) \ \geq \ 0, \ for \ all \ v \in V. \end{aligned} \tag{11}$$

Using the same logic of transformation followed for the hypervisor's maximization problem, the problem in Eq. (11) can be rewritten as:

$$\begin{aligned} \text{minimize} \quad & g \\ \text{subject to} \quad & g \geq \sum_{v \in V} q(v) \times U(h), \\ & q(v_1) + ... + q(v_l) = 1, \\ & q(v) \ \geq \ 0, \ for \ all \ v \in V. \end{aligned} \tag{12}$$

By carefully examining Eq. (9) and Eq. (12), we notice that the two programs are dual. Following the duality theorem [27], the maximum that the hypervisor can realize in Eq. (12) is equivalent to the minimum that the attacker can achieve in Eq. (9).

## 6 Numerical Example

Consider a hypervisor $h$ hosting, at time $t$, three VMs $v_1$, $v_2$, and $v_3$. The prices of these VMs are \$5.33, \$5.24, and \$6.86 respectively. Suppose as well that the hypervisor's detection probability at time $t$ is 0.85. The first step would be to build trust relationships between the hypervisor and its three guest VMs. Assume that the hypervisor performs a monitoring process for the VMs' performance using Algorithm 1 and that the monitoring process results in the following initial trust beliefs: $InitialBelief_h^{v_1} = 0.66$, $InitialBelief_h^{v_2} = 0.32$, and $InitialBelief_h^{v_3} = 0.68$. Note that $v_2$ is (largely) suspected initially to be launching DDoS attacks. To alleviate the uncertainty and come up with final beliefs, the hypervisor asks three other sources (i.e., VMs and hypervisors), say $s_1$, $s_2$, and $s_3$, about each VM's past behavior.

For $v_1$, suppose that the trust recommendations from the three sources are: $R_{s_1}^{v_1} = 0.66$, $R_{s_2}^{v_1} = 0.43$, and $R_{s_3}^{v_1} = 0.78$. By applying

Eq. (1) to compute the final belief in $v_1$'s trustworthiness, we get: $Belief_h^{v_1} = \frac{1.87+3\times 0.66}{2\times 3} = 0.642$. For $v_2$, suppose that $v_2$ is actually malicious and that $s_1$ and $s_2$ colluded to give $v_2$ high (good) recommendation scores, while $s_3$ gives a honest recommendation. Let the trust recommendations from the three sources regarding $v_2$ be: $R_{s_1}^{v_2} = 0.77$, $R_{s_2}^{v_2} = 0.61$, and $R_{s_3}^{v_2} = 0.40$. By applying Eq. (1) to compute the final belief in $v_2$'s trustworthiness, we get: $Belief_h^{v_2} = \frac{1.78+3\times 0.32}{2\times 3} = 0.456$. Note that although both $s_1$ and $s_3$ colluded to give $v_2$ high recommendation scores, the final belief of $h$ in $v_2$'s trustworthiness is still low (i.e., 0.456), which reveals that our trust model is quite resilient to the collusion attacks even when attackers form the majority. This is the case because our model combines objective and subjective sources to maximize the accuracy of the final trust results. For $v_3$, suppose that the trust recommendations from the three sources are: $R_{s_1}^{v_3} = 0.66$, $R_{s_2}^{v_3} = 0.59$, and $R_{s_3}^{v_3} = 0.63$. By applying Eq. (1) to compute the final belief in $v_3$'s trustworthiness, we get: $Belief_h^{v_3} = \frac{1.88+3\times 0.68}{2\times 3} = 0.653$. Suppose now that $\alpha$ is set to 1.5 and that all of $s_1$, $s_2$ and $s_3$ were allowed initially to make 2 inquiries from $h$ (i.e., $Inq(s_1 \rightarrow h) = 2$, $Inq(s_2 \rightarrow h) = 2$, and $Inq(s_3 \rightarrow h) = 2$). As rewards for recommending VMs to $h$, these three sources receive payments in the form of additional inquiries that they can make from $h$. For recommending $v_1$ and as per Eq. (2), the number of additional inquiries $s_1$ can make from $h$ increases up to $\frac{2}{|0.66-0.642|\times 1.5} = 74$, the number of additional inquiries $s_2$ can make from $h$ increases up to $\frac{2}{|0.43-0.642|\times 1.5} = 6$, and the number of additional inquiries $s_3$ can make from $h$ increases up to $\frac{2}{|0.78-0.642|\times 1.5} = 10$. The same logic of payment calculation applies also as to recommending $v_2$ and $v_3$. It is worth noticing that $s_1$ whose recommendation score nearly agrees with $h$'s final belief receives a large amount of inquiries compared to $s_2$ and $s_3$ whose recommendation scores diverge somewhat from that belief.

By employing Eq. (3) and Eq. (5) for deriving the utility values of the hypervisor and attacker respectively, we obtain the following game matrix:

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

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

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

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

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

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

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

After applying step 2, we obtain:

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

**Step 3:** *Select the pivot belonging to row "a" and column "b" subject to the following properties:*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- $q(v_1) = 0.0572/0.1437 = 0.3979$,
- $q(v_2) = 0.0413/0.1437 = 0.2875$, and
- $q(v_3) = 0.0452/0.1437 = 0.3146$.

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

## 7 Experimental Results and Analysis

In this section, we describe the experimental setup and present experimental results by comparing our solution with a benchmark consisting of the *price-based maximin* [8] and the *fair allocation* [28], [29] detection load distribution strategies.

### 7.1 Experimental Setup

We provide experimental results to test the performance of our model and validate the theoretical and numerical results obtained in the previous sections. The objective of these experiments is three-fold. First, we aim to study how effective the proposed model is in terms of augmenting the attack detection and minimizing both the false positives and negatives. Second, we verify that applying our solution under DDoS attack environments contributes in minimizing the CPU, memory, and network bandwidth wastage. Third, we aim to test the efficiency of our solution in terms of execution time. To these ends, we conduct our experiments using CloudSim [12] in a 64-bit Windows 7 environment on a machine equipped with an Intel Core i7-4790 CPU 3.60 GHz Processor and 16 GB RAM. CloudSim is a cloud simulation tool that has witnessed in the past few years a growing recognition among both academic and industrial milieux. It provides several features that help mimic realistic cloud environments by enabling the simulation of (1) large-scale cloud environments including co-hosted virtualized services; (2) service provisioning and resources allocation policies; (3) network connections among the different cloud components; and (4) federated cloud environments that inter-network resources from both private and public domains [12]. We decided to simulate our own cloud instead of using rented resources from one of the existing cloud providers for the two following main reasons. First, most of the cloud providers such as Amazon EC2 have restriction rules regarding any security testing on their resources and systems, where all the large cloud providers list DoS testing as a non-permissible activity [30]. Second, no cloud provider offers its users direct access to the VMs' host, which makes inspecting performance information at the host level quite difficult to perform [30].

To build our cloud, we create a datacenter whose VMs' configuration is inspired by Amazon EC2 X-large instances[5]. Practically, the created datacenter hosts five physical machines each of which is assigned with a number of VMs varying from 10 to 50 of image size amounting to 10000 MB each. Every VM is equipped with 5-core CPU of 1000 Millions of Instructions Per Second (MIPS) each. Each VM has a memory RAM capacity of 16 GB, hard drive storage of 976.5625 GB, and network bandwidth share of 50000 Kbit/s. In the created datacenter, x86 has been used as a system architecture, Linux as an operating system, and Xen as a Virtual Machine Monitor (VMM). The properties of the datacenter and VMs are summarized in Table 2. The VMs are given a set of CPU-intensive tasks (i.e., cryptographic operations and scientific computations). The properties of the tasks have been populated from SPECjvm2008 [31], a standard benchmark suite for Java virtual machines.

TABLE 2: Datacenter Properties

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

To populate the trust recommendations regarding VMs, we resort to the use of the *Epinions* data set[6] that has been long used in cloud computing and many other domains for representing trust [32], [33]. The data set comprises $664,824$ ratings given by $49,290$ agents on $139,738$ items. The prices of the VMs that are used along with the trust scores to compute the utility functions of both the hypervisor and attackers have been populated from the Amazon EC2 pricing dataset [7]. We compare our model against a benchmark consisting of two other models, namely the *Price-based Maximin* [8] and *Fair Allocation* [28], [29]. Similar to our

[5] https://aws.amazon.com/ec2/details/

[6] https://snap.stanford.edu/data/soc-Epinions1.html

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

model, the *Price-based Maximin* employs a maximin game to derive the optimal detection load distribution. However, unlike our solution, this model considers the worth of the VMs (concretized as the VMs' prices) solely in the formulation of the problem. Our model considers, in addition to the worth, the trust scores of the VMs believed by the hypervisor. The fair allocation model, on the other hand, distributes the detection load in an equal fair manner among all VMs. Note that we have selected the fair allocation model to compare with since it is the commonly used allocation strategy for cloud resources in the domain of cloud computing [28], [29]. We have adapted the resources fair allocation model to the detection load distribution problem as we were not able to find, through our extensive literature review investigation, any detection load distribution strategy in the domain of cloud computing other than our previous *Price-based Maximin* strategy [8].

## 7.2 Experimental Results

Fig. 3 studies how effective the proposed model is in improving the detection performance metrics, namely attack detection, false positive and false negative percentages compared to the price-based maximin and fair allocation models. Fig. 3a reveals that our solution along with the price-based maximin outperforms the fair allocation model in terms of attack detection (this is the case as well for the rest of the parameters as will be shown later). The reason is that the two former models consider the attacker's strategy when deciding about the detection load distribution strategy among VMs contrary to the fair allocation model wherein the distribution is done by considering the IDS's perspective only. Moreover, Fig. 3a shows that the trust-based maximin performs better than the price-based maximin model in increasing the detection of the attacks. This is because our trust-based solution allows the hypervisor to learn about the behavior of the VMs over the time, which enables it to adjust the detection load distribution in such a way that assigns more load to the VMs that have a large number of misbehavior during their past history. Besides the incremental learning property that our solution offers to the hypervisor, the fact that trust is a private relationship between the hypervisor and VMs reduces the possibility of the attacker to predict the potential detection load distribution strategy that will be adopted by the hypervisor (i.e., the attacker is unable to know which VMs are the most trusted by the hypervisor to attack through). Per contra, in the price-based model, the attacker may anticipate, for example, that the hypervisor would dedicate more detection load to the more valuable VMs (knowing that the VMs' prices are publicly available for users) and adjust hence its attack distribution strategy accordingly.

Fig. 3b measures the false negative percentage that quantifies the percentage of attacks that the system was not able to capture during the detection process. This percentage is computed by subtracting the probability distributions of the attacker from those of the hypervisor when the values of the former are greater. The average false negative rate $\alpha_{[t_1,t_2]}$ during the time window $[t_1,t_2]$ is computed as follows:

$$\alpha_{[t_1,t_2]} = \sum_{x=t_1}^{t_2} \sum_{v \in V} \frac{(q_x(v) - p_x(v))}{t_2 - t_1} \text{ for each } q_x(v) > p_x(v). \tag{13}$$

The false negative rate in the example given in Section 6 would be: $\alpha = [(0.3979 - 0.3011)] = 0.0968$. Thus, the percentage of false negatives entailed by our model in the given example is: $0.0968 \times 100 = 9.68\%$. Fig. 3b shows that our model diminishes the false negative of the IDS for the same arguments explained in the context of attack detection. Intuitively, decreasing false negatives is an automatic result of increasing attacks detection.

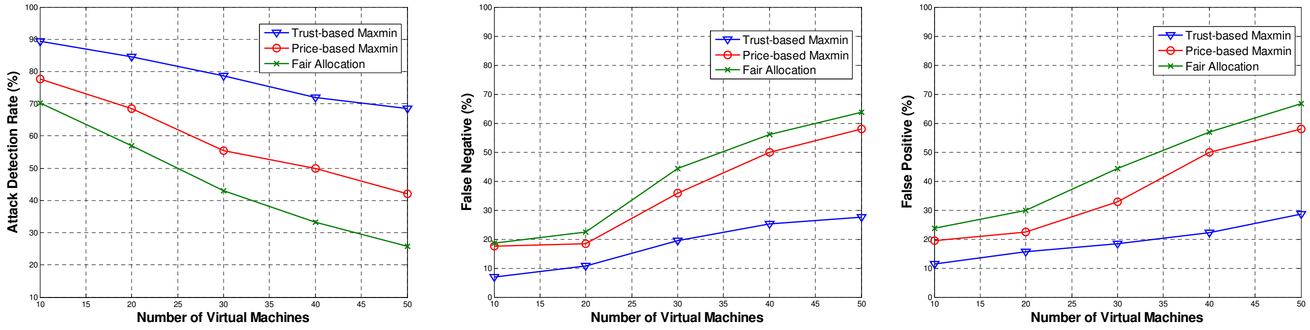
Fig. 3c measures the percentage of false positive incurred by the studied detection models. This metric is of prime importance in our case since it can tell us the percentage of resources wasted by the system during the detection process. Intuitively, false positive measures in our case the percentage of resources spent by the hypervisor in monitoring the VMs while these VMs are not sending any attack fragment. It is obtained by subtracting the probability distributions of the hypervisor from those of the attacker when the values of the former are greater. The average rate of resources wasted $\gamma_{[t_1,t_2]}$ during the time window $[t_1,t_2]$ is computed as follows:

$$\gamma_{[t_1,t_2]} = \sum_{x=t_1}^{t_2} \sum_{v \in V} \frac{(p_x(v) - q_x(v))}{t_2 - t_1} \text{ for each } p_x(v) > q_x(v), \tag{14}$$

The false positive rate in the example given in Section 6 would be: $\gamma = [(0.3562 - 0.2875) + (0.3427 - 0.3146)] = 0.0968$. Thus, the percentage of false positives entailed by our model in the given example is: $0.0968 \times 100 = 9.68\%$. Fig. 3c shows that our model is able to considerably reduce the percentage of false positives compared to the other two models. This is justified by the fact that our model guides the hypervisor on the optimal distribution of detection load that best synchronises with the attacker's probability distributions of the DDoS attacks.

Fig. 4 is introduced to study the effectiveness of the proposed model in minimizing the cloud system's resources consumption when this system faces DDoS attack scenarios. Fig. 4a measures the CPU consumption of the cloud system under DDoS attacks. Practically, we measure the percentage of CPU that is being consumed by the VMs in the presence of both legitimate requests coming from well-behaving VMs and fake (malicious) requests coming from malicious or compromised VMs. Fig. 4a reveals that using our proposed model minimizes the cloud system's CPU consumption under DDoS. Indeed, the fact that our model augments the attack detection and decreases the false positive and negative percentages enables the hypervisor to optimize its resources allocation strategy by limiting the CPU portion assigned to the VMs that are detected to be generating malicious requests, which helps thus save the system's resources and restrict the wastage. Similarly, Fig. 4b demonstrates that applying our solution minimizes the cloud system's memory consumption compared to the price-based maximin and the fair allocation models. In fact, by enhancing the hypervisor's ability to recognize the malicious requests, our model mitigates the load put on the system's memory by these requests. Fig. 4c shows that our solution reduces the network bandwidth's consumption compared to the other two models. This is due to the effectiveness of our model in identifying attacks, which reduces the flux of the malicious traffic on the datacenter's network.

Moreover, it is worth noticing from Fig. 3 and Fig. 4 that the performance of the different models either in terms of performance metrics (attack detection, false positive, and false negative) or in terms of resources consumption (CPU, memory, and bandwidth) decreases with the increase in the number of VMs. This fact is expected since the more the VMs deployed in a cloud infrastructure, the higher is the freedom given to the attacker to divide its attacks over a greater number of VMs and the less is the detection load

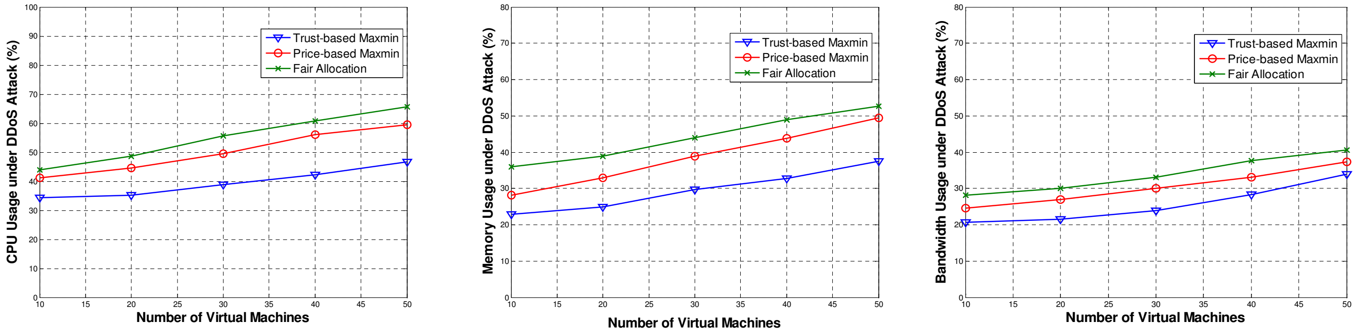


(a) Attack Detection

(b) False Negative

(c) False Positive

Fig. 3: Our model increases the percentage of detected attacks and decreases the percentages of false negatives and resources wastage compared to the price-based maximin and the fair allocation strategy.

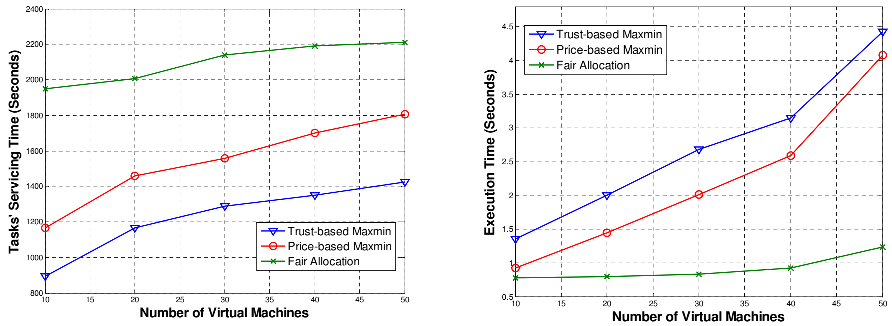


(a) CPU Usage

(b) Memory Usage

(c) Bandwidth Usage

Fig. 4: Our model minimizes the CPU, memory, and network bandwidth usage under DDoS attack compared to the price-based maximin and the fair allocation strategy.



(a) Tasks' Servicing Time

(b) Execution Time

Fig. 5: Our model reduces the tasks' servicing time compared to the price-based maximin and the fair allocation strategy and is efficient in terms of execution time.

that the hypervisor might be able to dedicate for each single VM. Though, our model is still far more resilient to a larger number of deployed VMs thanks to the advantages explained earlier that our solution brings, which supports the scalability of our model.

We study in Fig. 5a the average time taken by the VMs to respond to the assigned tasks. This metric is obtained by subtracting the tasks' end time from the tasks' start time for each VM and averaging these times over all the cloud system's VMs. The results demonstrate that our model aids in reducing the servicing time compared to the price-based maximin and fair allocation models. The reason is that our model is able to maximize the detection of the DDoS attacks, which helps reduce the congestion on the cloud system's resources and assists hence the legitimate tasks in being accomplished in a more efficient fashion.

Finally, Fig. 5b measures the efficiency of the three studied models in terms of their execution time with the variation in the number of VMs. The results reveal that the fair allocation model performs faster than both our model and the price-based model. This result is expected since the fair allocation model divides simply the detection load equally among VMs, which alleviates the time spent on computing the optimal detection load distribution. Moreover, the price-based model performs a bit faster than our model. This time difference may be thought of as the time needed by our model to collect and compute the trust scores for the VMs and adding these scores to the utility functions. In addition, we can notice from the results that our model takes around 4.4

seconds to compute the optimal detection load distribution for a cloud system consisting of 50 VMs, which boosts the efficiency of our solution. Moreover, by inspecting Fig. 5b, we can notice that the time complexity of the model grows polynomially with the increase in the number of VMs, which upholds the feasibility of our model in large-scale cloud datacenters.

## 8 Conclusion

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

## Acknowledgment

This work has been supported by the Fonds de Recherche du Québec - Nature et Technologie (FRQNT), Natural Sciences and Engineering Research Council of Canada (NSERC), Khalifa University of Science, Technology & Research (KUSTAR), Associated Research Unit of the National Council for Scientific Research (CNRS-Lebanon), and Lebanese American University.

## References

[1] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and R. Buyya, "DDoS Attacks in Cloud Computing: Issues, Taxonomy, and Future Directions," *ACM Computing Surveys*, 2015.

[2] F. Lombardi and R. Di Pietro, "Secure virtualization for cloud computing," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1113–1122, 2011.

[3] U. Tupakula, V. Varadharajan, and N. Akku, "Intrusion detection techniques for infrastructure as a service cloud," in *IEEE DASC*, 2011, pp. 744–751.

[4] J. Nikolai and Y. Wang, "Hypervisor-based cloud intrusion detection system," in *ICNC*, 2014, pp. 989–993.

[5] J. S. Ward and A. Barker, "Varanus: In situ monitoring for large scale cloud systems," in *IEEE CloudCom*, vol. 2, 2013, pp. 341–344.

[6] W. Lin and D. Lee, "Traceback Attacks in Cloud–Pebbletrace Botnet," in *ICDCSW*, 2012, pp. 417–426.

[7] A. M. Lonea, D. E. Popescu, and H. Tianfield, "Detecting DDoS attacks in cloud computing environment," *International Journal of Computers Communications & Control*, vol. 8, no. 1, pp. 70–78, 2013.

[8] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "How to distribute the detection load among virtual machines to maximize the detection of distributed attacks in the cloud?" in *IEEE SCC*, 2016.

[9] F. G. David, *Business Statistics: a decision-making approach*. Pearson, 2014.

[10] M. S. Hamada, A. Wilson, C. S. Reese, and H. Martz, *Bayesian reliability*. Springer Science & Business Media, 2008.

[11] T. S. Ferguson, "Game theory," *Mathematics Department, UCLA*, 2008.

[12] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[13] C.-C. Lo, C.-C. Huang, and J. Ku, "A cooperative intrusion detection system framework for cloud computing networks," in *ICPPW*. IEEE, 2010, pp. 280–284.

[14] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "Towards trustworthy multi-cloud services communities: A trust-based hedonic coalitional game," *IEEE Transactions on Services Computing*, 2016.

[15] W. Itani, A. Kayssi, and A. Chehab, "Privacy as a service: Privacy-aware data storage and processing in cloud computing architectures," in *IEEE DASC'09*, 2009, pp. 711–716.

[16] M. Alhamad, T. Dillon, and E. Chang, "SLA-based trust model for cloud computing," in *NBiS*. IEEE, 2010, pp. 321–324.

[17] P. Manuel, "A trust model of cloud computing based on quality of service," *Annals of Operations Research*, vol. 233, no. 1, pp. 281–292, 2015.

[18] S. M. Habib, S. Ries, and M. Muhlhauser, "Towards a trust management system for cloud computing," in *TrustCom*. IEEE, 2011, pp. 933–939.

[19] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "A survey on trust and reputation models for Web services: Single, composite, and communities," *Decision Support Systems*, vol. 74, pp. 121–134, 2015.

[20] J. W. Tukey, "Exploratory data analysis," 1977.

[21] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and Communications Security*, 2009, pp. 199–212.

[22] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, "A stackelberg game for distributed formation of business-driven services communities," *Expert Systems with Applications*, vol. 45, pp. 359–372, 2016.

[23] A. R. Syversveen, "Noninformative bayesian priors. interpretation and problems with construction and applications," *Preprint Statistics*, vol. 3, 1998.

[24] V. N. Vapnik and V. Vapnik, *Statistical learning theory*. Wiley New York, 1998, vol. 1.

[25] L. Li, Y. Wang, and E.-P. Lim, "Trust-oriented composite service selection and discovery," in *Service-Oriented Computing*. Springer, 2009, pp. 50–67.

[26] H. Wang, B. Zou, G. Guo, J. Zhang, and D. Yang, "Integrating trust with user preference for effective web service composition," *IEEE Transactions on Services Computing*.

[27] D. G. Luenberger, *Introduction to linear and nonlinear programming*. Addison-Wesley Reading, MA, 1973, vol. 28.

[28] W. Wang, B. Liang, and B. Li, "Multi-resource fair allocation in heterogeneous cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 10, pp. 2822–2835, 2015.

[29] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong, "A game-theoretic method of fair resource allocation for cloud computing services," *The journal of supercomputing*, vol. 54, no. 2, pp. 252–269, 2010.

[30] R. Shea and J. Liu, "Understanding the impact of denial of service attacks on virtual machines," in *IWQoS*. IEEE, 2012, pp. 1–27.

[31] "SPEC Java Virtual Machine Benchmark 2008," http://www.spec.org/jvm2008/, accessed: 2016-08-30.

[32] T. H. Noor and Q. Z. Sheng, "Trust as a service: a framework for trust management in cloud environments," in *WISE*. Springer, 2011, pp. 314–321.

[33] S. Deng, L. Huang, and G. Xu, "Social network-based service recommendation with trust enhancement," *Expert Systems with Applications*, vol. 41, no. 18, pp. 8075–8084, 2014.

**Omar Abdel Wahab** is a Ph.D. student at Concordia University, Canada. His research interests are cloud computing, security and trust, and game theory.

**Jamal Bentahar** is a full professor at Concordia University, Canada. His research interests are multi-agent systems, service computing, and game theory.

**Hadi Otrok** holds an associate professor position in the Department of ECE at Khalifa University. His research interests are on network and computer security.

**Azzam Mourad** is an associate professor at the Lebanese American University. His research interests are information security and security engineering.