entropy

Article

Entropy-Based Economic Denial of Sustainability Detection

Marco Antonio Sotelo Monge [†], Jorge Maestre Vidal [†] and Luis Javier García Villalba [*,†]

Group of Analysis, Security and Systems (GASS), Department of Software Engineering and Artificial Intelligence (DISIA), Faculty of Computer Science and Engineering, Office 431, Universidad Complutense de Madrid (UCM), Calle Profesor José García Santesmases , 9, Ciudad Universitaria, 28040 Madrid, Spain; masotelo@ucm.es (M.A.S.M.); jmaestre@ucm.es (J.M.V.)

* Correspondence: javiergv@fdi.ucm.es; Tel.: +34-91-394-7638

† These authors contributed equally to this work.

Received: 11 November 2017; Accepted: 27 November 2017; Published: 29 November 2017

Abstract: In recent years, an important increase in the amount and impact of Distributed Denial of Service (DDoS) threats has been reported by the different information security organizations. They typically target the depletion of the computational resources of the victims, hence drastically harming their operational capabilities. Inspired by these methods, Economic Denial of Sustainability (EDoS) attacks pose a similar motivation, but adapted to Cloud computing environments, where the denial is achieved by damaging the economy of both suppliers and customers. Therefore, the most common EDoS approach is making the offered services unsustainable by exploiting their auto-scaling algorithms. In order to contribute to their mitigation, this paper introduces a novel EDoS detection method based on the study of entropy variations related with metrics taken into account when deciding auto-scaling actuations. Through the prediction and definition of adaptive thresholds, unexpected behaviors capable of fraudulently demand new resource hiring are distinguished. With the purpose of demonstrate the effectiveness of the proposal, an experimental scenario adapted to the singularities of the EDoS threats and the assumptions driven by their original definition is described in depth. The preliminary results proved high accuracy.

Keywords: Cloud Computing; Denial of Service; Economic Denial of Sustainability; Entropy; Intrusion Detection; Information Security

1. Introduction

The main goal of Denial of Service attacks (DoS) is to deplete the resources of the victim systems with the purpose of their disabling. The abbreviation DoS typically refers to threats with a single source; when they are originated in multiple sources, the expression Distributed Denial of Service attacks (DDoS) is applied. In the last decades these threats have grown, become more sophisticated and have acquired a greater intrusive capacity, which magnifies their impact and hinder their mitigation. The different information security agencies have warned about this problem. For example, the European Union Agency for Network and Information Security (ENISA) registered a 30% increase of DDoS threads in the last year [1]. Given the magnitude of impact of the threads registered at Autumn 2016 [2], both European Commission (EC) [3] and US Government [4] announced an important reinforcement in their measures against these attacks. According to the European Police (Europol), their harmful capabilities are propitiated by different circumstances: The rapid proliferation of botnets, the emergence of novel vulnerabilities and amplifying elements, a greater offer of malicious products as Crimeware-as-a-Service (CaaS) in the black market, the massive popularization of certain technologies (e.g., mobile devices, Internet of Things (IoT), etc.), and the ignorance of users concerning good practices related with data protection and information security [5].

The most common DDoS approach is based on flooding, which modus operandi is to inject several requests in order to saturate the victim processing capabilities. As highlighted in [6], this is achieved by the constant and continuous generation of large volumes of requests (i.e., high rate flooding), or by seasonal injection of less noisy numbers of requests (i.e., low rate flooding). Consequently, the research community assumed these behaviors and over the last decade has proposed solutions that facilitate their prevention, detection, mitigation and identification of sources, some of them discussed in-depth at [7]. However, the emergence of new monitoring environments, in particular those that adapt the technologies that take part of the backbone of fifth generation networks, has led to a variation of these threats that instead of compromising computing resources, has focused on damaging the economic sustainability of the services they support [8]. They are well-known as Economic Denial of Sustainability attacks (EDoS), and they are the object of study of the research described throughout the rest of the paper. With the purpose of cooperate with the research community towards their mitigation, the following main contributions are accomplished:

- An in-depth review of the EDoS threats and the efforts made by the research community for their detection, mitigation and identification of sources.
- A multi-layered architecture for EDoS attack detection, which describes the management of the acquired information from its monitoring to the notification of possible threats.
- A novel entropy-based EDoS detection approach, which assuming its original definition, allows to discover unexpected behavior on local-level metrics related with the auto-scaling capabilities of the victim system.
- An evaluation methodology adapted to the singularities of the EDoS threats and the assumptions driven by their original definition.
- Comprehensive experimental studies that validate the proposed detection strategy, in this way motivating its adaptation to future use cases.

The paper is organized into five sections, being the first of them the present introduction. Section 2 studies the main features of the EDoS attacks and their countermeasures. Section 3 introduces a novel EDoS detection system based on entropy variations analysis. Section 4 describes the performed experimentation. Section 5 describes and discusses the obtained results. Finally, Section 6 presents the conclusions and future work.

2. Background

This section describes the main features of the Economic Denial of Sustainability threats and some of the most relevant countermeasures in the bibliography.

2.1. Economic Denial of Sustainability Attacks

Hoff coined the term Economic Denial of Sustainability attacks (EDoS) in 2008 [9,10] and Cohen extended its definition [11], which is currently adopted by the research community. EDoS attacks are usually directed against Cloud Computing infrastructures, which play an essential role in the emergent communication technologies. Because of this, Singh et al. formally defined EDoS attacks as "threats which target is to make the costing model unsustainable and therefore making it no longer viable for a company to affordability use or pay for their Cloud-based infrastructure" [12]. EDoS are also tagged in the bibliography as Reduction of Quality (RoQ) threats [13], or Fraudulent Resource Consumption attacks (FRC) [14]. These intrusions take advantage of the "pay-as-you-go" accounting model offered by most of the Cloud Computing providers and their auto-scaling services [15]. Their modus operandi slightly varies depending on the providers and the Cloud solutions they offer (e.g., OpenStack, Microsoft Azure, Amazon EC2, etc.) [13], as well as the scaling policies they implement (discrete, adaptive, etc.). However, EDoS tends to display a common pattern: The attacker injects requests that must be processed at server-side. They pose an important workload effect, which may be caused by different actions, among them requesting large files or queries [16], HTTP-requests on XML files [17], or exploiting alternative Application layer vulnerabilities [18–20]. When the flooding of requests exceeds the computational capabilities of the hired services, the auto-scaling processes trigger the need of contract additional resources, which increases the bill that the client must pay. Sonami et al. [14] studied the consequences of this increasing of costs, which have distinct impacts depending on the side. For example, in addition to the impact on the offered Quality of Service, the economic losses may become unsustainable for the clients, and consequently they probably will try to find a more profitable provider. This obviously also affects the supplier, which loses reputation, and hence money at long-term. The attack also impairs other services and network layers, mostly because the impact of deploying additional resources. This involves, among others, physical infrastructure, Network Function Virtualization (NFV) or multi-tenancy, which may compromise additional network resources [21]. For example, in [13], a low-rate flooding variant of EDoS is introduced with the purpose of maximize the collateral damage (i.e., the consequences of auto-scaling) and make its detection more difficult. Such publication reviews its consequences at different Cloud Computing architecture levels.

2.2. Countermeasures

In general terms, the extensive literature related to the defense against conventional DDoS threats lacks publications effective against EDoS attacks. This is because EDoS focus on make Cloud resources economically unsustainable instead of their depletion. This often occurs by far less noisy attacks, and with a greater resemblance to the behavior of the legitimate user [16]. Because of this, EDoS detection is driven by metrics related with resource consumption at server-side, while conventional DDoS detections usually analyzes network traffic metrics at packet and flow level [22]. Several specific approaches against EDoS attacks are collected and discussed in [8,16,23]. Some of them aim on their detection, which typically distinguishes two methods. The first of them analyzes network traffic metrics, as is the case of those that describe the web browsing behaviors [24], time spent at web pages [25] or packet header attributes, for example their TTL [26]. They are easy to implement and efficient, but lie on Application layer or networking protocol features more related to DDoS than EDoS; hence their accuracy is greatly restricted to each use case [27]. On the other hand, the second approach is based on modeling the economical sustainability of the services looking for suspicious discordances [28]. This method was significantly less considered by the research community, mainly because of its specificity; in particular it entails a greater difference with conventional DDoS detection strategies and demand more complex processes at server-side. However, it is independent of the exploited network layer and provides a more comprehensive understanding of the impact of the requests to the protected services, the latter usually leading to greater accuracy.

Publications based on prevention and mitigation of EDoS threats focus on hampering their execution and minimizing their impact. The most complex prevention solutions mathematically model the resources required by the protected services, and anticipate the consumption of future requests, which usually adopts game theory or queuing methods [29]. They allow anticipating harmful situations facilitating proactive responses, but must be complemented by reactive solutions. Major efforts towards mitigation EDoS threats focus on deploying access control mechanisms, as is the case of Crypto-puzzles [30–32], Graphical Turing tests [26,33] or reputation systems [34,35]. They are effective, but as highlighted in [23], resolving hard tests or deploying complex reputation schemes consume additional resources at both client and server sides, and significantly affect the Quality of Experience (QoE) on the protected environment.

Once the threats are detected and mitigated, the final step is to identify their source. The bibliography lacks publications that specifically address this problem, excluding certain exceptions as [24]. They usually model server usage behaviors based on Application layer metrics, among them web session duration, number of HTTP requests, or their impact on the protected environment. More generalist solutions are inherited from the advances on the conventional DDoS attack source identification. They mainly include packet traceback techniques, some of them being collected in [36]. For example, in [37] a novel approach that bypasses the deployment difficulties of the conventional IP traceback techniques by studying ICMP error messages is proposed. As reviewed in [38], the features of the network topology have an important impact in the effectiveness of the source identification approaches, which tend to be problematic in highly non-seasonal environments. Alternatively, traps as honeypots [39], or decoy virtual machines that co-exist with those real in the same physical hosts [40] are deployed. They implement the aforementioned methods, thus providing an additional level of security.

3. EDoS Attack Detection

With the purpose of establish the basis for defining an appropriate design methodology, the peculiarities of the conventional Denial of Service attacks, the legitimate mass access to the protected services (i.e., flash crowds), and their differences with the Denial of Sustainability threats have been taken into account. They allowed to define the following assumptions and limitations concerning the proposal described in the rest of this section:

- As remarked by Hoff in the original definition of EDoS attacks [9], they pose threats that do not aim on deny the service of the victim systems, but increase the economic cost of the services they offer to make them unsustainable.
- Hereinafter, Chris clarified that at network-level, EDoS threats resemble activities performed by legitimate users [10]. This implies that the distribution of the different network metrics (number of request, number of sessions, frequency, bandwidth computation, etc.) does not vary significantly when these attacks are launched. This is because in order to ensure their effectiveness, they must go unnoticed.
- It is possible to identify EDoS attacks by analyzing performance metrics at local-level. Given that at network-level there are no differences between EDoS and normal traffic, the requests performed by these threats must involve a greater operational cost.
- Requests performed by EDoS attacks have a similar quality to those from legitimate users (for example, a similar success rate). However, attackers may exploit vulnerabilities (usually at Application layer) to extend their impact [14].
- DDoS attacks usually originate from a large number of clients, where each of them performs a huge number of low-quality requests. On the other hand, EDoS attacks also come from many sources, but each client performs an amount of request similar to that of legitimate users. Unlike in flash crowds, EDoS attacks affect the predictability of the performance metrics related to the costs resulting from attending the requests served by the victim [18].

Based on these premises, it is possible to assume that, by studying the predictability of performance metrics at local-level (e.g., processing time, memory consumption, input and output operations, CPU consumption, etc.), it is possible successfully identify EDoS attacks. This is taken into account in the following subsections, where the introduced detection strategy is described. The proposal has the architecture illustrated in Figure 1. Therefore, it must perform three main tasks: (1) monitoring and aggregation; (2) novelty detection and (3) decision-making. They are described below.

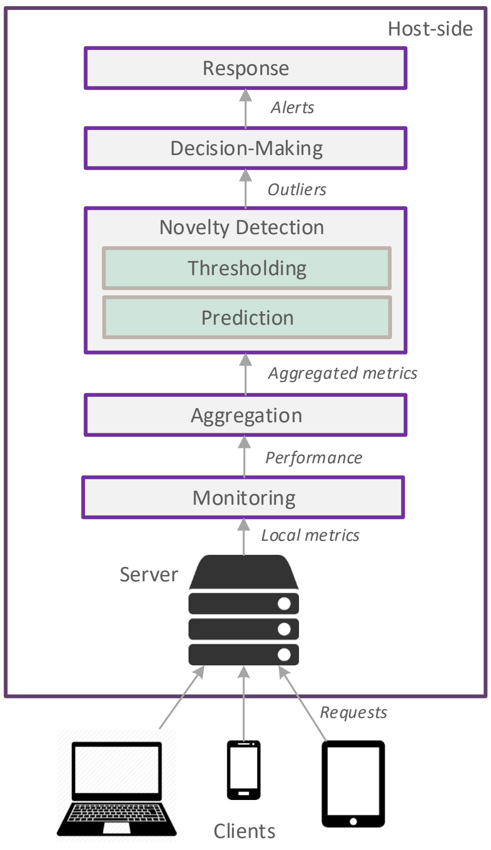


Figure 1. Architecture for EDoS attack detection.

3.1. Monitoring and Aggregation

At the monitoring stage, the factual knowledge necessary to deduce the nature of the requests to be analyzed is collected. Therefore, the detection system monitors local metrics related to the operational cost of responding the received request. Assuming that in order to success, EDoS attacks attempt to trigger the auto-scaling mechanisms of the victim-side, the metrics that determine these actions acquire special relevance. Note that they are widely studied in the bibliography, which vary according to the management services. Examples of well-known local-level metrics are: CPU utilization, warming time, response time, number of I/O requests, bandwidth or memory consumption [13,14]. Because of its relevance in the recent Cloud computing commercial solutions (e.g., Google Cloud, Amazon EC2, etc.) the performed experimentation considered the percentage CPU usage of the victim system. On the other hand, it is important to borne in mind that the analysis of the predictability degree of events has played an essential role in the defense against conventional DDoS threats. Among the most used aggregated metrics, it is worth mentioning the classical entropy adaptation to the information theory proposed by Shannon [41]. Note that in approaches like [42] it is demonstrated its effectiveness when applied to DDoS detection, being a strong element in the discovery of flooding threats. Recent publications such as [16,27,28] tried to adapt this paradigm to the EDoS problem. However, most of them made the mistake of only considering information monitored at network-level, hence ignoring part of the information that truly defines the auto-scaling policies. Because of this, the Aggregation stage of the proposed method calculates the information entropy $H(X)$ of the $\{x_1, x_2, \ldots, x_n\}$ instances of the qualitative variable X monitored per observation, as well as their $\{p_1, p_2, \ldots, p_n\}$ probabilities. The proposed detection scheme defines X as "the response time (rate) to the different requests performed by each client". Given that X describes discrete events, its entropy is expressed as follows:

$$H(X) = \sum_{i=1}^{n} p_i \log_a p_i \tag{1}$$

where $\log_a b. \log_b x = \log_a x$. $H(X)$ is normalized, hence being calculated when dividing the obtained value by the maximum observable entropy $\log_b n$. When the maximum entropy is reached, all the monitored clients made requests with the same CPU overload; on the contrary, if the registered entropy is 0 then (1) a single customer carried out all the requests, or (2) there was no CPU consumption during the observation period. The sequence of monitored entropies is studied as a time series $H(X)_{t=0}^{N}$.

3.2. Novelty Detection

The next analytic step is to recognize the observations that significantly vary from normal behaviors. This is a one-class classification problem where it is assumed that the normal data compiles the previous $H(X)_{t=1}$,..., $H(X)_{t=N-1}$ observations and it is intended to deduce if $H(X)_{t=N}$ belongs to the same activities. The bibliography provides a large variety of solutions to this problem [43]. However, because it was assumed that EDoS attacks could be identified by discovering discordances at the predictability of local-level aggregated metrics [18], the proposed system implements a forecasting approach.

3.2.1. Detection Criteria

In particular, the entropy for certain horizon h, $\hat{H}(X)_{t=N+h}$, is predicted. Hence, letting the following Euclidean distance:

$$dist(o, \hat{o}) = \sqrt{(\hat{H}(X)_{t=N+h} - (X)_{t=N+h})^2} \tag{2}$$

If $(X)_{t=N+h}$ differs from $\hat{H}(X)_{t=N+h}$, so $dist(o, \hat{o}) > 0$ an unexpected behavior is detected. The significance of this anomaly is established by two adaptive thresholds: Upper Threshold (Th_{sup}) and Lower Threshold (Th_{inf}). A novelty was discovered if any of the following conditions is met:

$$\begin{aligned} dist(o, \hat{o}) > 0 \quad and \quad H(X)_{t=N+h} > Th_{sup} \\ dist(o, \hat{o}) > 0 \quad and \quad H(X)_{t=N+h} < Th_{inf} \end{aligned} \tag{3}$$

3.2.2. Prediction

The implemented prediction methodology adopted the Autoregressive Integrated Moving Average $ARIMA(p, d, q)$ paradigm [44], which defined by the following general-purpose forecast model:

$$Y_{\tau-1} - a_1 Y_{\tau-1} - \cdots - a_{p'} Y_{\tau-p'} = \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} \tag{4}$$

where a_i are the parameters of the autoregressive part, θ_i are the parameters of the moving average part and ϵ_t is white noise. The adjustment of p, d, q may be the ARIMA model equal to other classical forecasting models. For example simple random walk $(ARIMA(1, 1, 0))$, $AR(ARIMA(1, 0, 0))$, $MA(ARIMA(0, 0, 1))$, simple exponential smoothing $(ARIMA(0, 1, 1))$, double exponential smoothing $(ARIMA(0, 2, 2))$, etc. Predictions $(\hat{y}_t)$ on ARIMA models are inferred by a generalization of the autoregressive forecasting method expressed as follows:

$$\hat{y}_t = \mu + \phi_1 Y_{\tau-1} + \phi_p Y_{\tau-p} - \phi_1 \epsilon_{t-1} - \cdots - \phi_q \epsilon_{t-q} \tag{5}$$

and the calibration of the adjustment parameters p, d, q considered the Akaike Information Criterion (AIC) as described in [45].

3.2.3. Adaptive Thresholding

On the other hand, the adaptive thresholds define the Prediction Interval (PI) of the sensor, which is deduced in the same way as it is usually described in the bibliography [4], hence assuming the following expressions:

$$\begin{aligned} Th_{sup} &= H(X)_{t=N+h} + K\sqrt{\sigma^2 var(dist(o,\hat{o}))} \\ Th_{inf} &= H(X)_{t=N+h} - K\sqrt{\sigma^2 var(dist(o,\hat{o}))} \end{aligned} \tag{6}$$

and being K the confidence interval of the estimation (by default $Z_{\frac{\alpha}{2}}$). Note that despite linking its value to the normal distribution, it was demonstrated that when time series does not approach such distribution, the obtained error is unrepresentative [46]. Figure 2 illustrates an example of novelty detection. In the first 60 observations non $H(X)$ exceeds the adaptive thresholds; but at observation 61 an EDoS attack was launch, and the inferred changes meet the conditions to be considered novel.

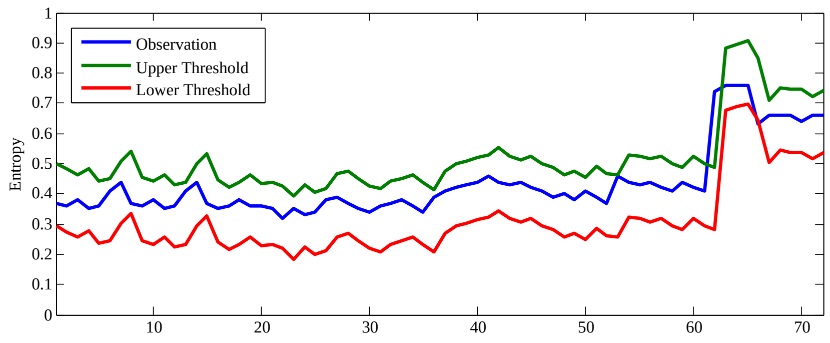


Figure 2. Example of novelty detection.

3.3. Decision-Making and Response

According to the principles of anomaly-based intrusion detection compiled and discussed by Chandola et al. [47], once assumed the appropriate premises, the identification of discordant behaviors may be indicative of malicious activities. As stated at the beginning of this section, the introduced EDoS detection system lies on the original definitions of C. Hoff and R. Cohen. Therefore, when a local metric directly related with triggering auto-scaling capabilities on Cloud computing became unpredictable, it is possible deduce that the protected environment is misused, hence jeopardized. This occurs when $dist(o,\hat{o}) > 0$ and (1) $H(X)_{t=N+h} > Th_{sup}$ or (2) $H(X)_{t=N+h} < Th_{inf}$. Because the performed research focused only on detect the threats, its response is to notify the detected incident. The report may trigger mitigation measures such as initiate more restrictive control access [30,31,33] or deploy source identification capabilities [24] (which decision and development is out of scope). Therefore, it entails a good complement to many of the proposals in the bibliography.

4. Experiments

The following sections describe the Cloud-based testbed and related architectural components considered throughout the performed experimentation. They are depicted in Figure 3.

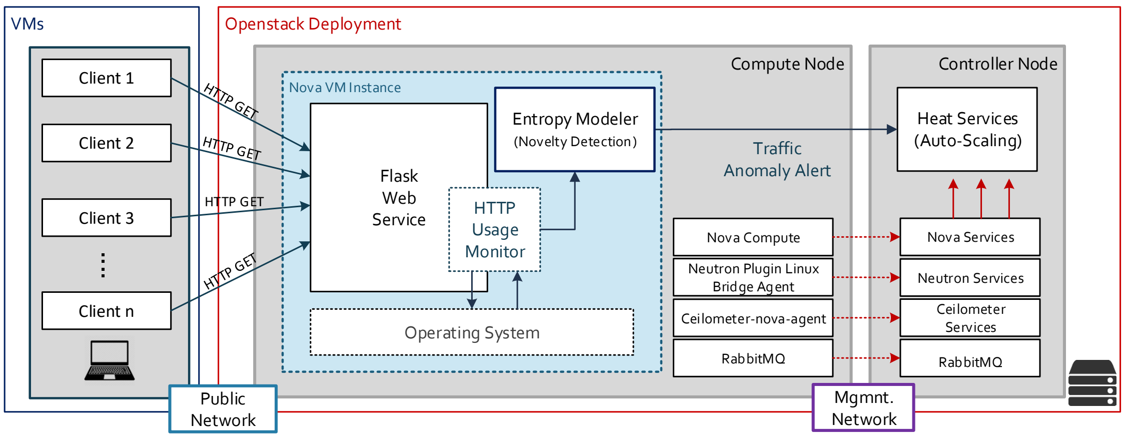


Figure 3. Cloud execution environment for experiments.

4.1. Execution Environment

The experimental cloud computing environment was built with Openstack [48], a well-known open source cloud platform suitable to deploy public and private cloud environments of any size. The auto-scaling features of this cloud platform have also been tested effectively on recent publications [49,50]. The Openstack deployment for the experimental testbed was composed by one controller node and one compute node. The controller runs core Openstack services and it also holds the Networking (Neutron), Compute (Nova) essentials, Telemetry (Ceilometer) and Message Queue (RabbitMQ) services. In addition, it runs the Orchestration (Heat) services to allow the configuration of auto-scaling policies. The compute node runs in a separate server, hosting the Nova core services. A new Compute instance has been launched to deploy the web service used for experimentation. This virtual instance runs an Ubuntu 16.04-x64 server with 8 CPU cores and 8 GB of RAM memory.

On top of the operating system, a REST (Representational State Transfer) web service written in Flask [51] has been implemented. A REST web service has been chosen due to its simplicity and rapid development. REST is the predominant web API design model built upon HTTP methods [52], which accommodates the system to interact with several entities (i.e., humans, IoT devices). In REST every client request (1) only generates a single server response (one-shot) and (2) every response must be generated immediately (one-way) [53]. This request-response model is suitable to focus the analysis on the measurement of CPU processing times, by tracking the connected user and the impact of its client requests on the CPU consumption.

In addition to the web service, two modules were developed to be run in the background: The HTTP Usage Monitor module and the Entropy Modeler. The former logs information regarding the monitoring of client requests processing times, whereas the latter performs novelty detection methods to trigger anomaly-based alerts to the Openstack orchestration services.

On the client-side, a set of REST-clients have been deployed to generate traffic according to several execution scenarios. The implementation details and characteristics of the components tested in the experimentation stage are explained in the forthcoming sections.

4.2. Server-Side Components

The following describes the deployed server-side components: RESTful Web Service, HTTP Usage Monitor and the Entropy Modeler.

4.2.1. RESTful Web Service

To facilitate a seamless interaction with HTTP clients, a REST web service has been implemented on Flask, a Python-based framework for rapid development of web applications. The REST service exposes four HTTP endpoints that produce the execution of different list-sorting operations on the server, each of them consumes a different amount of CPU time which is measured in the background. The endpoints and their average execution times are summarized in Table 1.

Table 1. HTTP GET endpoints and CPU average cost.

URI	Parameters	Average CPU Time in Second (1000 exec.)
/1	?id={clientID}	0.02158
/2	?id={clientID}	0.02781
/3	?id={clientID}	0.03673
/4	?id={clientID}	0.33604

4.2.2. HTTP Usage Monitor

Once the server receives a client HTTP request, the Usage Monitor module permanently measures the amount of CPU time consumed to process the request before sending the response back to the client. The module makes use of Python libraries and standard Linux utilities to track the CPU consumption per each client request. The collected data is then aggregated per client in configurable time intervals before being logged to the system. If more than one client connection is being observed in the given time interval, only the sum (aggregated metric) of all the processing times is logged. This allows the creation of a time series, required for the next processing level.

4.2.3. Entropy Modeler

This module gathers the time series logged by the HTTP Usage Monitor and computes the entropy of the CPU time usage of the different requests performed by each client. With the resultant normalized entropy, the module forecasts the next h observations for the given time series, in conformance with the ARIMA model. The predicted values are taken to estimate the forecasting upper and lower thresholds. Whenever the resultant entropy falls outside the prediction intervals, a Traffic Anomaly alert is reported to the auto-scaling engine of the corresponding Cloud platform (i.e., Openstack Heat).

4.3. Client-Side Component

On a separate server, several clients have been implemented as Python multi-threading scripts for HTTP traffic generation, which is sent to the web service hosted in the Openstack virtual machine instance. The generated number of traffic requests is a discrete variable that follows a random Poisson distribution, since their similarity with this distribution is widely assumed by the research community [54]. It is modeled according to the traffic load requirements for each evaluation scenario. Every client is represented by a process thread, which models multiple parallel clients handling their own sets of requests independently from others. When normal network conditions are modeled, all the clients send an HTTP GET request to the lower CPU-consuming requests (endpoints 1–3) described in Table 1. When an attacker is modeled, it only calls the most complex endpoint (4), which has higher CPU demands at server-side. Note that GET requests can also accept the client ID as a parameter. It facilitates the implementation of different client connections originated in the same computer since all the thread-based clients share the same source IP address, but are differentiated by client ID.

4.4. Test Scenarios

Five main scenarios have been showcased to validate the proposal. All of them compare the entropy levels of CPU processing times under normal traffic conditions against the entropy measured when an EDoS attack is launched. Those attacks target to produce CPU overhead. Therefore, the attack decrements the server capacities to handle more connections, and it forces the decision to scale up the current virtual machine instance when the CPU usage is above a pre-defined CPU limit in the Cloud-platform auto-scaling engine. The set of network traffic conditions described in Table 2 are assumed throughout the experiments. There, clients (C) generate the total number of web requests

(TR) at the expected rate (ERS). It is worth remarking that ERS corresponds to the expected number of occurrences (λ) of the Poisson distribution. Therefore, the generated web requests represent the sample of connections to be analyzed. The MTR observation number (5000) is the frontier that divides the TR into two groups of 5000 client requests each. The first one operates under the normal traffic conditions described in Table 2; whereas a percentage of the second group contains the malicious requests, letting the remaining connections to operate under the normal conditions. For instance, in the second group a 5% malicious requests rate indicates that 250 malicious requests and 4750 normal requests were observed. Table 3 defines the evaluation scenarios (E1 to E5) considered to deploy the EDoS attacks.

Table 2. Normal traffic conditions for experiments.

Characteristic	Value
Web clients (C)	500
Expected requests per second (ERS)	60
Total web requests (TR)	10,000
Malicious Triggering Request (MTR)	5000

Table 3. Network attack conditions and scenarios.

Parameter	E1	E2	E3	E4	E5
Malicious Request Rate (MRR)	1%	5%	10%	15%	20%
Attacker Clients (AC)	5	25	50	75	100
Total number of malicious requests $(TR - MTR) \times MRR$	150	250	500	750	1000

The experiments performed for each scenario started their execution with the normal web traffic conditions (first group of connections), with all the participant clients requesting the endpoints 1–3, as explained before. However, at the time specified by the MTR connection, the attack was launched. It compromised several normal clients (C), which sent malicious requests to the endpoint 4, thus increasing the CPU overhead. It is important to remark that the attackers connect to the server under the same ratio (ERS) configured for normal clients, making them unnoticeable since their connection rate resembled legitimate traffic, but they targeted to exploit the highest time-consuming endpoint which was exposed as a service vulnerability. To validate the proposal, it has been considered a Cloud auto-scaling policy, configured to launch a new virtual machine instance when the CPU consumption ran above 40% in a one minute interval.

5. Results

The experiments were performed with the parametrization presented in Table 3, adapted to each evaluation scenario. The first monitored metric was the CPU time consumption caused to process web requests launched from clients. A summary of the CPU consumption of the server, measured on one-second intervals, is depicted in Figure 4. There, in all scenarios, half of the client connections exposed the same behavior until the attack was triggered (MTR). From that moment on, the CPU overhead was influenced by the traffic attack volume described in Table 3. Bearing in mind the defined auto-scaling policy, it is noted that the scenarios E3, E4 and E5 would have automatically launched a new virtual machine instance if the presence of the attack had been unnoticed. Hence demonstrating the consequences of the EDoS threats and bringing the attack detection strategy to play an essential role. On the other hand, besides the CPU estimation, the entropy of the per-client processing time was constantly measured by the Entropy Modeler on one-second intervals, as plotted in Figure 5. The graph shows that the overall behavior of the entropy was contrary to the behavior noticed in the CPU overhead with the higher entropy values before the MTR observation. The slumped entropy level was slightly noticeable on scenario E1 (Figure 5a), but became quite more perceptible on scenarios E2 to E5 (Figure 5b to Figure 5e). Thereby, this pattern was directly influenced by the presence of the compromised devices, decreasing the entropy as long as more malicious requests were generated. Only when the entropy was measured for the observed time, the Entropy Modeler estimated the prediction thresholds to infer if the observed entropy was running outside the predicted intervals, thus leading to the decision of triggering an alert if the EDoS attack was detected.

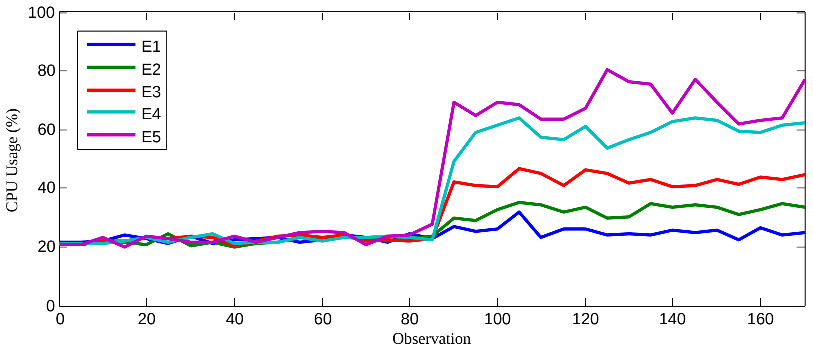
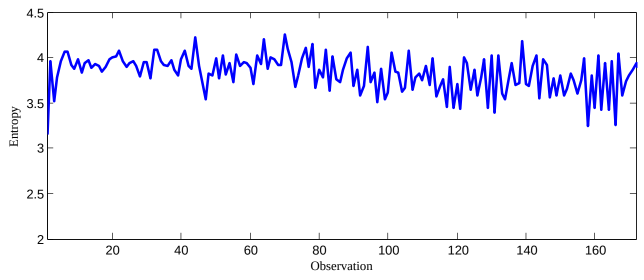


Figure 4. Average CPU consumption per scenario.

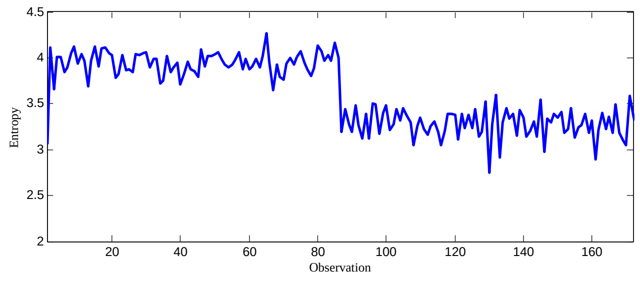
The precision observed at the Receiver Operating Characteristic (ROC) space is summarized in Figure 6. There five curves are illustrated, each one associated with one of the aforementioned evaluation scenarios (E1, E2, E3, E4, E5). Table 4 compiles several evaluation metrics (True Positive Rate (TPR), False Positive Rate (FPR) and Area Under Curve (AUC)) and the best calibrations (K) to reach the highest accuracy. Bearing in mind these results, it is possible to deduce that the proposed method has proven to be more effective when the attack is originated from a larger number of compromised nodes (e.g., E5 with 20% of the total number of connected clients). This is because a greater number of instances of the random variable X represent similar probabilities, which leads to a more significant decrease in the $H(X)$ entropy, and therefore to display less concordance with the normal observations. On the other hand, labeling errors have occurred mainly due to issuing false positives, in situations where fluctuations of $H(X)$ derived from changes in the behavior of legitimate clients acquire a similar relevance to those inferred by malicious activities. Note that the larger is the number of compromised nodes that take part of the attacks, the greater possibility of forcing auto-escalating reactions. Based on this fact it is possible to state that the proposed method improves its detection capabilities when facing more harmful threats. In addition, the existence of a K calibration parameter allows operators to easily configure the level of restriction in which the system operates: When greater discretion is required, K must adopt higher values. This considerably reduces the likelihood of issuing false alerts, hence facilitating to minimize the cost of the countermeasures to be applied. On the opposite case, when the monitoring environments require greater protection it is advisable to decrease K, hence improving the possibility of detecting threats, but potentially leading to deploy more unnecessary countermeasures.

Table 4. Summary of results in ROC space.

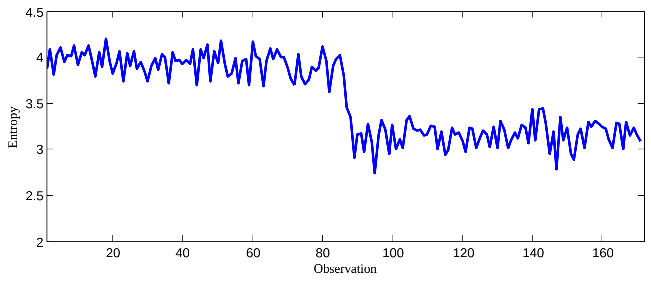
Scenario	AUC (Trapezoidal)	TPR	FPR	K
E1	0.8858	0.7480	0.17	0.160
E2	0.9637	0.9630	0.09	0.163
E3	0.9766	0.9680	0.08	0.160
E4	0.9794	0.9644	0.06	0.160
E5	0.9830	0.9431	0.03	0.167



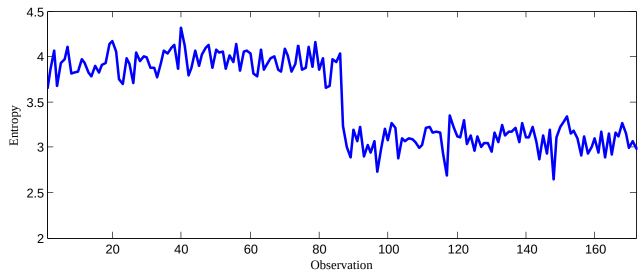
(**a**) Entropy evolution in E1



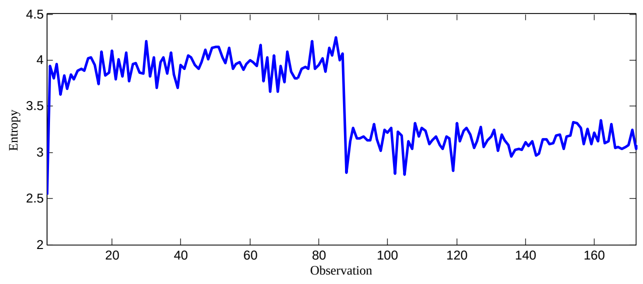
(**b**) Entropy evolution in E2



(**c**) Entropy evolution in E3



(**d**) Entropy evolution in E4



(**e**) Entropy evolution in E5

Figure 5. Entropy measurements per scenario.

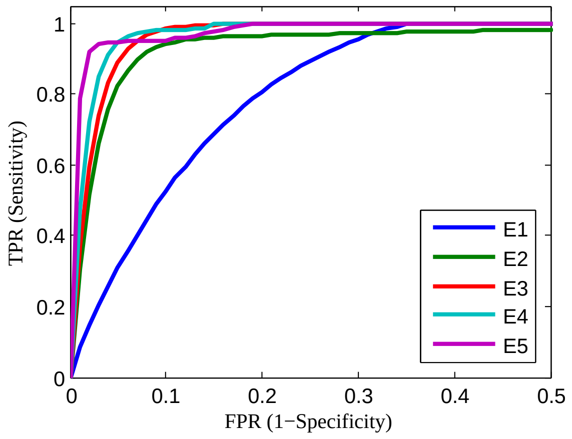


Figure 6. Results in ROC space.

6. Conclusions

In this paper, an entropy-based model for the detection of EDoS attacks in Cloud environments has been introduced. For this purpose, a comprehensive revision of the EDoS related research has been covered to elaborate a multi-layered architecture tackling the detection of EDoS attacks. The proposed work suggested good detection accuracy, thus preventing the unnecessary consumption of additional Cloud-resources if they were issued by auto-scaling policies based on unreal demands.

The experiments conducted to validate the proposed architecture have encompassed all the stages defined in the architecture, starting from the monitoring and aggregation of metrics that directly affect the Cloud computing cost model, the novelty detection procedures to recognize an EDoS attack, and the decision-making and response actions to be applied in the system. The experimental testbed implemented a client-server REST architecture executed on different network scenarios. On the web server, the monitored per-client CPU times have been evaluated by analyzing the entropy levels, which have exposed a decrement when malicious requests originated by the compromised nodes have been processed at server-side. In such scenarios, entropy has behave indirectly proportional to the consumed CPU. In addition, the detection method has also demonstrated its effectiveness when predicting the entropy thresholds to be compared against the real measured entropy. Thereby, this approach has proven high accuracy by quantifying the area under the ROC curve. It is also worth mentioning the enhancement of the proposed model compared to other resource-consuming approaches presented in the literature; such as the requesting of large files, database queries, or other web vulnerabilities; since this architecture relies on server-side consumption rather than anomalous network-level metric patterns.

The presented approach, evaluation methodology and the experiments conducted throughout this work poses also new potential research lines. The experimental scenarios should be extended to couple diverse network conditions to either enhance the validation or to disclose some evasion techniques. The defined model of measuring the resource consumption and diagnosing its entropy can be accommodated to include more metrics, thus extending its scope to wider analysis scenarios. Furthermore, it might be fitted to enhance adaptive auto-scaling policies on Cloud platforms by incorporating more complex evaluation criteria. Finally, the existing decision-making and countermeasures to EDoS attacks remain far from being evolved, and might effectively complement the conducted research.

Acknowledgments: This work is supported by the European Commission Horizon 2020 Programme under grant agreement number H2020-ICT-2014-2/671672 - SELFNET (Framework for Self-Organized Network Management in Virtualized and Software Defined Networks).

Author Contributions: The authors contributed equally to this research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. European Union Agency for Network and Information Security (ENISA) Threat Landscape Report 2016. Available online: https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2016 (accessed on 28 November 2017).
2. Kolias, C.; Kambourakis, G.; Stavrou, A.; Voas, J. DDoS in the ioT: Mirai and other botnets. *Computer* **2017**, *50*, 80–84.
3. European Comission Cybersecurity Stratregy. 2017. Available online: https://ec.europa.eu/digital-single-market/en/policies/cybersecurity (accessed on 28 November 2017).
4. US National Cyber Incident Response Plan (NCIRP). 2017. Available online: https://www.us-cert.gov/ncirp (accessed on 28 November 2017).
5. European Police (Europol). The Internet Organised Crime Threat Assessment (IOCTA). 2017. Available online: https://www.europol.europa.eu/activities-services/main-reports/internet-organised-crime-threat-assessment-iocta-2017 (accessed on 28 November 2017).
6. Wei, W.; Chen, F.; Xia, Y.; Jin, G. A rank correlation based detection against distributed reflection DoS attacks. *IEEE Commun. Lett.* **2013**, *17*, 173–175.
7. Zargar, S.T.; Joshi, J.; Tipper, D. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Commun. Surv. Tutor.* **2013**, *15*, 2046–2069.
8. Baig, Z.A.; Sait, S.M.; Binbeshr, F. Controlled access to cloud resources for mitigating Economic Denial of Sustainability (EDoS) attacks. *Comput. Netw.* **2016**, *97*, 31–47.
9. Chris, H. Cloud Computing Security: From DDoS (Distributed Denial Of Service) to EDoS (Economic Denial of Sustainability). 2008. Available online: http://rationalsecurity.typepad.com/blog/2008/11/cloud-computing-security-from-ddos-distributed-denial-of-service-to-edos-economic-denial-of-sustaina.html (accessed on 28 November 2017).
10. Chris, H.A Couple of Follow-Ups on the EDoS (Economic Denial of Sustainability) Concept ... 2009. Available online: http://rationalsecurity.typepad.com/blog/edos/ (accessed on 28 November 2017).
11. Reuven, C. Cloud Attack: Economic Denial of Sustainability (EDoS). Available online: http://www.elasticvapor.com/2009/01/cloud-attack-economic-denial-of.html (accessed on 28 November 2017).
12. Singh, P.; Manickam, S.; Rehman, S.U. A survey of mitigation techniques against Economic Denial of Sustainability (EDoS) attack on cloud computing architecture. In Proceedings of the IEEE 3rd International Conference on Reliability, Infocom Technologies and Optimization (ICRITO), Noida, India, 8–10 October 2014; pp. 1–4.
13. Bremler-Barr, A.; Brosh, E.; Sides, M. DDoS attack on cloud auto-scaling mechanisms. In Proceedings of the IEEE Conference on Computer Communications (INFOCOM 2017), Atlanta, GA, USA, 1–4 May 2017; pp. 1–9.
14. Somani, G.; Gaur, M.S.; Sanghi, D.; Conti, M. DDoS attacks in cloud computing: Collateral damage to non-targets. *Comput. Netw.* **2016**, *109*, 157–171.
15. Somani, G.; Gaur, M.S.; Sanghi, D.; Conti, M.; Buyya, R. DDoS attacks in cloud computing: Issues, taxonomy, and future directions. *Comput. Commun.* **2017**, *107*, 30–48.
16. Bhingarkar, A.S.; Shah, B.D. A survey: Securing cloud infrastructure against edos attack. In Proceedings of the International Conference on Grid Computing and Applications (GCA), Athens, Greece, 27–30 July 2015; pp. 16-22.
17. Vivinsandar, S.; Shenai, S. Economic Denial of Sustainability (EDoS) in Cloud Services Using HTTP and XML Based DDoS Attacks. *Int. J. Comput. Appl.* **2012**, *41*, 11–16.
18. Zhou, W.; Jia, W.; Wen, S.; Xiang, Y.; Zhou, W. Detection and defense of application-layer DDoS attacks in backbone web traffic. *Future Gener. Comput. Syst.* **2014**, *38*, 36–46.
19. Singh, K.; Dee, T. MLP-GA based algorithm to detect application layer DDoS attack. *J. Inf. Secur. Appl.* **2017**, *36*, 145–153.
20. Singh, K.; Singh, P.; Kumar, K. Application layer HTTP-GET flood DDoS attacks: Research landscape and challenges. *Comput. Secur.* **2017**, *65*, 344–372.

21. Singh, A.; Chatterjee, K. Cloud security issues and challenges: A survey. *J. Netw. Comput. Appl.* **2017**, *79*, 88–115.
22. Berezinski, P.; Jasiul, B.; Szpyrka, M. An entropy-based network anomaly detection method. *Entropy.* **2015**, *17*, 2367–2408.
23. Bawa, P.S.; Manickam, S. Critical Review of Economical Denial of Sustainability (EDoS) Mitigation Techniques. *J. Comput. Sci.* **2015**, *11*, 855–862.
24. Idziorek, J.; Tannian, M.; Jacobson, D. Attribution of fraudulent resource consumption in the cloud. In Proceedings of the IEEE 5th International Conference on Cloud Computing, Honolulu, HI, USA, 24–29 June 2012; pp. 99–106.
25. Koduru, A.; Neelakantam, T.; Bhanu, S.M.S. Detection of Economic Denial of Sustainability Using Time Spent on a Web Page in Cloud. In Proceedings of the IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), Bangalore, India, 16–18 October 2013; pp. 1–4.
26. Al-Haidari, F.; Sqalli, M.H.; Salah, K. Enhanced EDoS-Shield for Mitigating EDoS Attacks Originating from Spoofed IP Addresses. In Proceedings of the IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Liverpool, UK, 25–27 June 2012; pp 1167–1174.
27. Singh, K.J.; Thongam, K.; De, T. Entropy-Based Application Layer DDoS Attack Detection Using Artificial Neural Networks. *Entropy* **2016**, *18*, 350.
28. Idziorek, J.; Tannian, M. Exploiting Cloud Utility Models for Profit and Ruin. In Proceedings of the IEEE International Conference on Cloud Computing (CLOUD), Washington, DC, USA, 4–9 July 2011; pp. 33–40.
29. Yu, S.; Tian, Y.; Guo, S.; Wu, D.O. Can We Beat DDoS Attacks in Clouds? *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 2245–2254.
30. Masood, M.; Anwar, Z.; Raza, S.A.; Hur, M.A. EDoS armor: A cost effective economic denial of sustainability attack mitigation framework for e-commerce applications in cloud environments. In Proceedings of the IEEE 16th International Multi Topic Conference (INMIC), Lahore, Pakistan, 19–20 December 2013; pp. 37–42.
31. Khor, H.; Nakao, A. Spow: On-demand cloud-based eDDoS mitigation mechanism. In Proceedings of the IEEE/IFIP International Conference on Dependable Systems & Networks (DSN), Lisbon, Portugal, 29 June–2 July 2009.
32. Kumar, M.N.; Sujatha, P.; Kalva, V.; Nagori, R.; Katukojwala, A.K.; Kumar, M. Mitigating Economic Denial of Sustainability (EDoS) in Cloud Computing Using In-Cloud Scrubber Service. In Proceedings of the IEEE 4th International Conference on Computational Intelligence and Communication Networks (CICN), Mathura, India, 3–5 November 2012; pp. 535–539.
33. Alosaimi, W.; Al-Begain, K. A new method to mitigate the impacts of the economical denial of sustainability attacks against the cloud. In Proceedings of the 14th Annual Post Graduates Symposium on the convergence of Telecommunication, Networking and Broadcasting (PGNet), Liverpool, UK, 24–25 June 2013; pp. 116–121.
34. Liu, J.K.; Au, M.H.; Huang, X.; Lu, R.; Li, J. Fine-Grained Two-Factor Access Control for Web-Based Cloud Computing Services. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 484–497.
35. Yan, Q.; Yu, F.R.; Gong, Q.; Li, J. Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 602–622.
36. Alenezi, N.M.; Reed, M.J. Uniform DoS traceback. *Comput. Secur.* **2014**, *45*, 17–26.
37. Yao, G.; Bi, J.; Vasilakos, A.V. Passive IP traceback: Disclosing the locations of IP spoofers from path backscatter. *IEEE Trans. Inf. Forensics Secur.* **2015**, *10*, 471–484.
38. Jeong, E.; Lee, B. An IP traceback protocol using a compressed hash table, a sinkhole router and data mining based on network forensics against network attacks. *Futur. Gener. Comput. Syst.* **2014**, *33*, 42–52.
39. Wang, K.; Du, M.; Maharjan, S.; Sun, Y. Strategic Honeypot Game Model for Distributed Denial of Service Attacks in the Smart Grid. *IEEE Trans. Smart Grid*, **2017**, *8*, 2474–2482.
40. Al-Salah, T.; Hong, L.; Shetty, S. Attack Surface Expansion Using Decoys to Protect Virtualized Infrastructure. In Proceedings of the 2017 IEEE International Conference on Edge Computing (EDGE), Honolulu, HI, USA, 25–30 June 2017; pp. 216–219.
41. Shannon, C.E. A mathematical theory of communication. *Bell Syst. Tech. J.* **1948**, *27*, 379–656.
42. Bhuyan, M.H.; Bhattacharyya, D.K.; Kalita, J.K. An empirical evaluation of information metrics for low-rate and high-rate DDoS attack detection. *Pattern Recognit. Lett.* **2015**, *51*, 1–7.

43. Pimentel, M.A.F.; Clifton, D.A.; Clifton, L.; Tarassenko, L. A review on novelty detection. *Signal Process.* **2014**, *99*, 215–249.
44. Hillmer, S.C.; Tiao, G.C. An ARIMA-Model-Based Approach to Seasonal Adjustment. *J. Am. Stat. Assoc.* **1980**, *77*, 63–70.
45. Ong, C.S.; Huang, J.J.; Tzeng, G.H. Model identification of ARIMA family using genetic algorithms. *Appl. Math. Comput.* **2005**, *164*, 885–912.
46. Hyndman, R.J.; Koehler, A.B.; Ord, J.K.; Snyder, R.D. Prediction intervals for exponential smoothing state space models. *J. Forecast.* **2005**, *24*, 17–37.
47. Chandola, V.; Banerjee, A.; Kumar, V. Anomaly Detection : A Survey. *ACM Comput. Surv.* **2009**, *41*, doi:10.1145/1541880.1541882.
48. Open Source Sotware for Creating Private and Public Clouds. Available online: https://www.openstack.org (accessed on 28 November 2017).
49. Kang, S.; Lee, K. Auto-scaling of Geo-based image processing in an OpenStack cloud computing environment. *Remote Sens.* **2016**, *8*, 662.
50. Krieger, M.T.; Torreno, O.; Trelles, O.; Kranzlmuller, D. Building an open source cloud environment with auto-scaling resources for executing bioinformatics and biomedical workflows. *Futur. Gener. Comput. Syst.* **2017**, *67*, 329–340.
51. Flask-A Python Microframework. Available online: http://flask.pocoo.org (accessed on 28 November 2017).
52. Schnase, J.L.; Duffy, D.Q.; Tamkin, G.S.; Nadeau, D.; Thompson, J.H.; Grieg, C.M.; Mclnerney, M.A.; Webster, W.P. MERRA analytic services: Meeting the big data challenges of climate science through cloud-enabled climate analytics-as-a-service. *Comput. Environ. Urban Syst.* **2017**, *61*, 198–211.
53. Fielding, R.T.; Taylor, R.N.; Erenkrantz, J.R.; Gorlick, M.M.; Whitehead, J.; Khare, R.; Oreizy, P. Reflections on the REST architectural style and principled design of the modern web architecture (impact paper award). In Proceedings of the 11th Joint Meeting on Foundations of Software Engineering, Paderborn, Germany, 4–8 September 2017; pp. 4–14.
54. Barakat, C.; Thiran, P.; Iannaccone, G.; Diot, C.; Owezarski, P. Modeling Internet backbone traffic at the flow level. *IEEE Trans. Signal Process.* **2003**, *51*, 2111–2124.