A Collaborative Load Balancer for Network Intrusion Detection in Cloud Environments

Taha Arian[1*], Amir Kusedghi[1], Bijan Raahemi[2], Ahmad Akbari[1]
1 School of Computer Engineering Iran University of Science and Technology, Tehran, Iran.
2 University of Ottawa, Ottawa, Canada.

* Corresponding author. Tel.: (0098) 912-6403212; email: t_arian@ce.sharif.edu
Manuscript submitted September 15, 2015; accepted December 28, 2015.
doi: 10.17706/jcp.12.1.28-47

Abstract: Information security and permeability in the system is a major concern for cloud computing. Cloud service providers should ensure that user information remains private from other people (external or internal). Deployment of an intrusion detection system (IDS) is a technique to protect the cloud from existing security and intrusion threats. Due to the high traffic volume and the need for flexibility and the diversity of applications in a cloud environment, a distributed IDS is used to apply traffic management, and provide scalability concurrently. The challenge is to propose a method of load balancing that distributes input traffic among IDS sensors, and optimally balance the workload on the sensors, to improve the overall performance of the IDS, with minimum degradation in quality. In this paper, a load balancing method is presented which operates based on the sensors' hardware specification and their compatibility with incoming requests, and the sensors' available resources. Dynamic request allocation to sensors is done in real time, using application layer load distribution, with no need to migrate requests or sensors, which could lead to extra overhead. The accuracy of IDS sensors in detection of attacks can be affected by the distribution of input traffic, but this shortcoming is resolved by our method. We compare the proposed method with existing algorithms in terms of load balancing and IDS functionality. The results confirm superior performance of IDS functionality in the proposed architecture.

Key words: Cloud computing, distributed intrusion detection system, adaptive weighted load balancing.

1. Introduction

Today, one of the most important applications of cloud computing is for trade and economic affairs. The reasons for cloud technology's popularity include flexibility, scalability based on request volume, payment based on usage and availability. The technology's challenges include the need to safeguard the security of users, and network bottlenecks. Use of an intrusion detection system (IDS) is a common method of protecting cloud networks from security and intrusion threats [1]. The high traffic volume in a cloud environment and the lack of a suitable central management make IDS deployment desirable. Methods have been presented in recent years for IDS improvement. The most important of these are the methods using software-defined networking, redundancy security tools and distributed methods [2]. The distributed methods are superior in terms of lower cost of using resources, the ability to use resources with diversified abilities, and with no need for strong and powerful standalone devices or infrastructure. The likelihood of the existence of a single point of failure is also reduced with a distributed architecture. Most proposed IDS applications have been associated with an input traffic distribution approach, employing several intrusion detection sensors [3].

The high traffic volume of a cloud environment can lead to a drop in efficiency when sensors are overloaded. To maximize compatibility within the context of the cloud, while safeguarding the efficiency of intrusion detection sensors, the main approach has been to increase distribution of input traffic, with net load reduction for each system sensor [2]. Load balancing is classified as being either static or dynamic. In dynamic methods, load-distribution threshold changes can be made base on sensor status reports [4]. Static methods are typically lower-cost, with simpler load balancing done based on sensor specifications and performance history. Dynamic methods have suitable scalability and flexibility, and can usually improve efficiency of the sensors and the quality of load balancing. Their drawbacks are complexity and execution time [4]. In contrast, static methods focus on load balancing speed, based on a time optimization and cost reduction approach. Usually, limitations in the efficiency and flexibility of sensors lead to difficulties in productivity of resources. In practice, there is a heterogeneous mix of dynamic and static methods, with different approaches by different vendors, and an increasing variety of network services [5]. The flexibility and scalability of load balancing methods are of paramount importance, as well as compatibility with the network environment. The dynamic method is used more often, due to its more consistent performance amid a changing environment, while affording proper flexibility in a cloud environment [3]. In this paper, important existing methods are examined, and a new cloud-context method is presented with improved efficiency for IDS applications.

2. Load Balancing in Cloud Computing

Load balancing is a key and effective method for increasing the efficiency and speed of data traffic in the context of the cloud. For a system in which some nodes have been loaded severely and others partially, load balancing reallocates loads to take maximum advantage of resources and improve response time. The nature of the load under consideration can be processing, memory use or network load. Therefore, there should be a mechanism for the selection of nodes [6]. The load balancing process reduces a system's response time, increases utilization of resources, and reduces the number of aborted tasks. In addition, load balancing increases end-user satisfaction and enhances system efficiency. The objectives and advantages of load balancing can be categorized among three groups [7]:

- Scalability
- Security
- Consumption of energy and resources

2.1. Classification of Load Balancing Systems

In addition to general architectural structure, improvement in the efficiency of load balancing methods usually hinges on improvement of their algorithm [8]. There are two common algorithm types:

- **Round Robin**: In this method, each node request is assessed and allocated sequentially, with no weighting.
- **Weighted Balancing**: In this type, each node is weighted based on its processing power and capabilities. Requests are then allocated according to the weighting. (The server with a higher weighting is able to respond to more requests.)

The most important published load balancing methods and algorithms include fuzzy logic [6], Central Load Balancing Decision Module (CLBDM) [7] and the Min-Min Method [9]. In terms of the network layers under consideration, load balancing systems can be divided into two groups. (These two groups are briefly discussed further in Section 6).

- **Load Balancer in Network High Layers [10]**: These types of distributor, which are active in layers of 4 to 7 of the network, usually have a specific approach for distributing traffic and requests between

servers. Load distribution is conducted using data of applied programs which have been executed by users. The load balancing process is one of the newest approaches of efficient distribution of requests. This method has the possibility of dispatching requests based on type of request [9]. For example, data in the header of an HTTP request can be used for server selection, such as directing multimedia requests to a specific server.

- **Load Balancer in Network Low Layers [11]**: This type of distributor usually makes decisions based on the specifications of bottom layers of network. They tend to have lower quality in functionality and flexibility than the distributors of high layers. For example, distribution could be carried out based on the originating or destination IP address, or a combination of them [4].

Load balancing methods can also be classified based on the condition of system, and divided into two groups [5]:

- **Static**: This type of load distribution does not depend on the current state of the system, but requires previous knowledge of the system. In this method, efficiency is not a paramount consideration.
- **Dynamic [4]**: In this method, the system's previous condition or behavior has no significance for load distribution. Instead, the current state and behavior of the system are used as criteria for load distribution. The comparative efficiency of this method is higher by far.

3. Intrusion Detection Systems

Setting up a computer system without the presence of weak security points is impossible in practice. Therefore, conducting research in the field of intrusion detection is of paramount importance for users of computer systems. An IDS comprises the hardware and/or software which is tasked with discovering any unauthorized application of system resources, misuse and/or incurred damage, arising from either domestic or foreign users. IDS are classified into two groups in terms of the supervision environment:

- **NIDS (Network IDS)**: This system is installed on an entrance point of a network (hub, switch and similar cases) and monitors traffic passing from the pertinent point.
- **HIDS (Host IDS)**: This system is installed on each host, and monitors the network's input/output traffic for this particular host.

3.1. IDS Use in the Context of the Cloud

Taking advantage of security components, such as an IDS, is necessary in a cloud environment. Service providers are obliged to focus on their system's security aspects for suitably safeguarding customer interests. Further users of a cloud environment cannot always trust providers of cloud infrastructure. Breaches of these conditions have spurred research innovations. One example is the introduction of IDS as a service (IDSaaS), in the context of cloud, allowing organizations and customers to take advantage of its security measures [12]. The Open Stack project is one of the most prominent platforms for this configuration in a cloud environment. This service has been implemented within the framework of Open Stack Networking (Neutron), allowing users to add desired redundancy to their project's main nucleus [13]. An IDS has the following requirements in a cloud environment [14]:

- An IDS should monitor several virtual machines (VMs), based on software requirements.
- Customers of cloud services should sometimes be able to control IDS management themselves.

Use of a traditional IDS in a cloud context typically leads to significant challenges. They usually cannot be applied in this context without some changes being needed [15]. In the following, the challenges that network designers face at the time of deployment of an IDS in a cloud environment are explained as follows:

- In a conventional IDS, security policies have a significant tendency to be static due to the nature of IDS sensors in monitored systems. Although in cloud environments, monitored VMs are typically added or

removed dynamically. In addition, security requirements of each VM will differ from those of its peers [13].

- The issue of the volume of input data to the system is one of the most important challenges faced in a vast cloud environment. The volume of traffic passing an IDS will see a considerable hike with increased numbers of users and nodes in the cloud. As these systems are subjected to higher rates of incoming requests, the ability of an IDS to monitor traffic is reduced with the increased rate of passing traffic [13].

4. IDS Load Balancing in a Cloud Context

Typically, it is not possible to use a single-sensor IDS in a cloud environment, with a distributed multisensory system preferred [16]. With this scheme, single points of failure are prevented, and any expected necessity for using concentrated, powerful sensors is mitigated. The process of removing and adding sensors could be carried out in a better way, due to the dynamic nature of requests in a cloud environment. Along with the potential for disrupted distribution of sensors, the process of distribution of requests among these sensors is of paramount importance. With high traffic in a cloud context, and diversity of the traffic, there is a possibility of overflowed sensors and the likelihood of a resulting loss of efficiency. Consequently, load balancing methods are usually used for a distributed IDS system [16], [17].

Load balancing for IDS sensors in the context of the cloud can follow two general approaches [18]:

1) **Proper allocation of requests at entrance time**: A suitable sensor is selected at the time of entrance of a request, and the request is allocated to that sensor.
2) **Secondary allocation of requests based on the status of sensors**: In the beginning, requests are allocated using presumption of sensor states. Based on studying the status of sensors during light loading, requests are allocated to an appropriate sensor.

The selected approach will depend on the objectives and criteria for the desired efficiency. The subject of load balancing with IDS use is one of the latest concerns in cloud contexts, and in recent years various methods have been presented for improving it. The methods can be divided into these groups:

1) **Request integration methods based on current states**: Request integration methods usually try to improve load balancing with the help of current management processes and topologies of the network. These methods will usually be compatible with the architecture of "software-based networks" [2].
2) **Weighting and Classification Methods of Input Load**: Using a process of prioritizing and weighting the traffic is one of the typically used methods. In general, a useful method has not yet been presented for IDS use in the field of load balancing.
3) **Adding Redundant Security Hardware or Software**: In this method, IDS efficiency improvements are sought, using other security tools, such as a honeypot [19].

Each of these classifications can be used for an existing scenario. Currency methods are usually used in clouds that enjoy a network protocol and virtual switches with proper capabilities. And enjoy the possibility of management of flow. Hardware methods will require a lateral security tool [20].

Here, weighting and prioritizing methods of sensors' input load will be more appropriate in terms of existing complexity calculations and costs, since they can be managed independently. Also, weighting methods will have better performance due to the dynamicity of a cloud environment, a situational change of active sensors, and fluctuating input traffic rates. Unlike methods based on currency, there is no need to periodically adjust specifications. Also, similar to hardware methods, there is no redundant security tools overhead [2], [19], [20]. Two methods can be used to characterize the efficiency of a distributed IDS in a cloud context. One is related to the evaluation of the efficiency of the load balancing process, while the other is related to the evaluation of IDS efficiency. Therefore, usability criteria can be based on these two evaluation methods.

4.1. IDS Efficiency Evaluation Criteria

IDS evaluation is usually conducted in terms of the identification of potential attacks, and the rate of warnings produced. For this purpose, four types of warnings can be defined for all IDS applications [21]:

1) **True Positive**: an alarm is produced, and relevant attacks have in fact been carried out.
2) **False Positive**: an IDS alarm occurs, but no attack has occurred.
3) **False Negative**: An attack has been conducted, but no IDS alarm is generated.
4) **True Negative**: An event when no attack has taken place and no detection is made.

The true-positive rate can also be called the "detection rate." The false-positive alarm rate is also known as the "False Alarm Rate". With increased identification rate of attacks, and a reduced rate of false alarms, the relevant IDS quality will be improved [21].

4.2. Load Balancing Efficiency Evaluation Criteria

A load distributor's evaluation process is usually specified by studying the status and specifications of VMs. Criteria used in this field include:

1) **Degree of Imbalance (DI)**: This criterion is defined to evaluate the balance of time and resources (memory, bandwidth and processor use) [22].

$$DI = \frac{T_{\max} - T_{\min}}{T_{avg}} \tag{1}$$

Refer to (1), values of $T_{\max}$ and $T_{\min}$ show the maximum and minimum amount of execution time of a request, among all requests (T_i) which have been executed on all VMs. Also, T_{avg} gives the average time of execution of all requests on all VMs. With a reduced value of DI, a better load balancing can be obtained.

2) **Load Balancing Metric (LBM) [23]**: This criterion shows the degree of balancing the loads of VMs, and is usually defined within the framework of the productivity rate of the processor of each VM. In this equation, the value of the load of each "i" sensor is defined by a "j" test (among "m" carried out tests) using the symbol $\text{load}_{i,j}$. Also, the maximum value of the load among all VMs in test "j" is shown by peak_load_j. LBM is defined as follows:

$$\text{LBM} = \frac{\sum_{1 \le j \le m} \text{peak\_load}_j}{\left(\sum_{1 \le j \le m} \sum_{1 \le i \le n} \text{load}_{i,j}\right) / n} \tag{2}$$

The value of LBM varies between 1 and the number of VMs (n). Smaller values of LBM indicate more appropriate load balancing.

3) **Makespan [24]**: This criterion studies the completion time of each request directly, and its value is derived from the requested resources and extant processing volume. The completion time of request "i" on VM "j" is denoted by the CT_{ij} parameter, with Makespan defined as:

$$\text{Makespan} = \max \left\{ CT_{ij} \middle| i \in T, i = 1,2, \dots n, j \in Vm, j = 1,2, \dots m \right\} \tag{3}$$

The smallest values of Makespan indicate more appropriate balancing of the relevant load.

5. Related Work

In earlier methods, sample sets of sensors were configured on hosts separately, and a central balancer undertook distribution of input traffic. Static hashing methods were used on the network and the header of the transfer layer, and the proper IDS sensor for undertaking load balancing was selected [25]. These methods have been improved by using algorithms which are adapted dynamically based on the current load dynamically [26], [27].

Previously, an IDS operated in isolation, and did not share its status. Instead, a comprehensive protocol

was presented for allowing the sharing of this information [28]. This technique provides identification of abnormality and adaptation of patterns in multi-connection mode. With any significantly increased rate of information transfer, homogeneity and updating of this information becomes difficult. For mitigation of this, a multi-nucleus environment was presented [29], which was configured based on Bro IDS [30]. This approach improved multi-nucleus architecture, with the division of processing data into stages. Both dynamic and sensitive distribution methods have been presented. These methods use all available resources of sensors optimally, based on assessment of the current status of sensors. With the proper distribution of input requests, both the availability and expandability properties of parallel sensors are improved [31]. Consistency of managing the current load under different conditions and changes of available hardware resources are considered to be one of the most important challenges facing these methods [12]. For this purpose, compatible methods have been presented.

In general, Dynamic Adaptive Load Balancing includes two parts: 1-Controlling Status, and 2-Load Balancing. "Controlling status" is used to monitor and collect current information, with load information including the productivity of the processor and bandwidth, with adding or removing sensors and other sensor information. The "load balancing" sector activates based on collected load information and weighting of sensors homogeneously [31].

A group of these distributors are based on activation and deactivation functions, operating due to single or double threshold states. These functions are summoned at the time of overflowing or underflowing of sensors. Based on this, an available redundancy load is transferred to the proper new sensor. Reduction in the number of these summons, and aggregation of produced alarms, lead to improvement in the efficiency of the system [32], [33]. Methods based on activation/deactivation with threshold extent are used in the load balancing process, in the cloud context. Based on the distribution of states of VMs, they are usually classified into the following three groups: 1-Overflowed, 2-Normal and 3-Underflowed. Reduction of the rate of activation/deactivation of VMs at the time of overflowing and underflowing is one of the most important challenges of this group [18]. Another group of these distributors carry out the process of proper allocation of a request at the time of its entrance. Secondary overhead will not exist, due to the lack of activation/deactivation. In this group, selection of the proper sensor is usually carried out with the help of algorithms assigning weighting to sensors, with input load and/or similar dynamic hashing algorithms [31], [34], [35]. Weighting-based load balancing methods are used by service providers comprehensively and, are generally designed based on their nature in the form of Real Time [36]. For this purpose, this group of methods is called "Real Time Server-Statistics Based Load Balancing" (RTSLB). A scoring process and selection of priority in the selection of service providers is one of the statistical methods used in RTSLB. This scoring has been conducted based on the following four parameters: 1-VM processing load, 2-VM response rate, 3-number of requests allocated to the VM, 4-the record of similar requests or multi-requests that should be cached [34]. VMs are scored at the time of entrance of each request using these four parameters. The VM which obtained the maximum score is selected to undertake the current load.

6. Proposed Architecture

Intrusion detection sensors are selected for use in specific networks based on their hardware specifications. A cloud environment requires more specialized intrusion detection sensors due to the very high traffic. For example, sensors have been chosen to identify attacks employing multimedia requests, with another group planned for a use because of database requirements [37], [38].

For use in a cloud context, intrusion detection sensors can be classified into the following three groups based on hardware specifications [39]:

1) Sensors with proper data bandwidth

2) Sensors with proper processing power
3) General sensors

Rules used for detection of attacks can be specified for each type of sensor. Alternatively, sensors may not differ in terms of rules used, but only in their hardware specifications.

As mentioned above, selection of the desired layer for employment of a load balancing process is one of the most important considerations in load balancing. Since this proposed architecture is considered for a cloud environment, the cloud (i.e., layer of VMs) is one of the main levels and the load balancing process is executed in the context of these machines. In Fig. 1, shows balancer locations in layered cloud architecture.

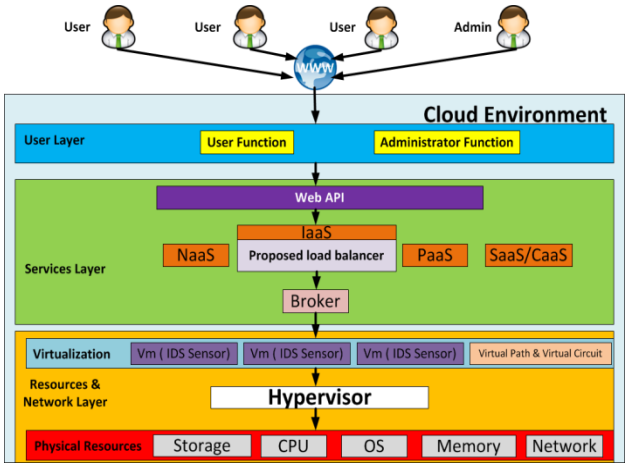

Fig. 1. Location of the proposed balancer in a layered cloud architecture.

Input requests to the cloud environment can be allocated using two methods:

1) Web service level
2) Super computer level

The super computer method was the first specialized method of request in a cloud environment, but was set aside due to practical restrictions and lack of proper totality [40]. Today, presenting cloud services in the context of web protocols (such as HTTP) is possible, because of the expansion of web-based systems and the availability of various types of web services.

Web API (application programming interface) agents have been presented for a web-based cloud environment, the most important of which include: Feedly Cloud API [41], Simple Cloud API [42] and Microsoft Azure user agent [43]. Each of the applied links has usually implemented various types of request analysis methods. Identification and isolation of various types of request, and their guidance to the relevant service providers, is one of the main duties of these methods. There are two main request analysis methods:

1) **Based on Header**: Isolation of web-based requests based on their HTTP header.
2) **Based on RESTful protocol**: Isolation of web-based requests based on the URL address [43].

```
GET /ws/IMFS/Multimedia.ashx?sessionToken
        =8da051b0-a60f HTTP/1.1
    Host: services.nirvanix.com
Date: Wed, 20 Oct  2013 12:00:00 GMT
```

Fig. 2. A sample of multimedia request in RESTful protocol.

Any isolation process and header analysis are more costly with the introduction of a new version of HTTP/2, and increased diversity of the headers within. With the increased rate of cloud requests, the time cost of analysis of these headers will create a considerable overhead. As a result, most cloud-applied links

use the method based on the RESTful protocol. In this method, any type of service has been provided presumably based on the personal address, with requests from users classified within the framework of applied links and sub-addresses. For instance, Fig. 2 shows multimedia requests may be sent with a certain date [44].

Consequently, all requests are classified based on address. From Fig. 2, this process is conducted in a web API[1], and requests have been tagged due to their types and are directed toward the proposed distributor. Apart from the type of API used, all of them are specifically tagged requests [45].

6.1. Scoring Distributed Sensors for Load Balancing

The proposed method uses a dynamic load balancing algorithm. With due observance of various types of requests, which are directed from a web API, and considering status and the available types of intrusion detection sensors in the cloud, a score is attributed to each sensor. The sensor obtaining the maximum score will be selected for the input request. Requests are classified based on their specifications and the optimal type of sensor corresponding to them. Classification details are shown in Table 1.

Table 1. Prioritizing Sensors Based on Request Type [46]

Request Type / Sensor Type	*Sensor with Bandwidth / Proper Band*	*Sensor with Proper Processing Power*	*General Sensor*
Application	✓	✓✓	
Audio	✓✓	✓	
Image	✓✓	✓	
Message	✓	✓✓	
Multipart	✓✓	✓	
Video		✓	✓✓

In Table 1, available sensors are prioritized for each request based its type. For example, for audio requests, a sensor with proper bandwidth is the most appropriate sensor type (indicated by two ticks). A general sensor (without a tick) can be selected if a more appropriate sensor type is not available. With such classification, consistency and compatibility of various types of cloud input requests have been correlated with intrusion detection sensors. In other papers, a combination of productivity value and sub-network physical specifications of each sensor is used as a prioritizing parameter for sensors. For example, in [47], the degree of productivity and response time of each sensor is used in a combined form. The main problem with these parameters is that the combinations merely rely on sensor specifications. Therefore, they will not necessarily attain the appropriate efficiency. Therefore, in the proposed method two parameters are used for scoring sensors

1) Available resources
2) Degree of compatibility of the type of input request and intrusion detection sensors ($C_{\text{request-sensor}}$)

Available resources indicate the usable capacity of a sensor's hardware components, which is defined based on (4).

$$\text{Available Resources} = 1 - U_{\text{sensor}} \tag{4}$$

In this formula, U_{Sensor} is the utilization of a sensor, which indicates the value of applied resources. This parameter is defined using a weighted combination of utilization of bandwidth, main memory and processor:

$$U_{\text{Sensor}} = \frac{\sum_{i=\text{Bandwidth,Memory,CPU}} (W_i * U_i)}{\sum_{i=\text{Bandwidth,Memory,CPU}} W_i} \qquad 0 \le U_{\text{Sensor}} \le 1 \tag{5}$$

[1]Application Program Interface

Weighting utilization in terms of bandwidth, main memory and processor is conducted according to the following three factors, for the production of a total utilization unit parameter, according to (5):

1) **Based on Input Request**: In this method, weighting is conducted based on the type of utilization and the input request type (Table 1). For example, if the input request is video, more weight can be associated to $U_{\text{Bandwidth}}$ parameter. Consequently, the effect of this parameter will be increased in the calculation of U_{sensor} .
2) **Based on Load Balancing Manager Decision**: In this method, weighting coefficients are allocated by the balancer manager empirically, based on the adopted policy.
3) **Based on Current Status of Each Sensor Dynamically**: With increased utilization and productivity in terms of bandwidth, main memory and processor, changing coefficients for the calculation of U_{sensor} is possible. If, for example, $U_{\text{Bandwidth}}$ is overflowed severely, its coefficient could be increased to have a larger share in the calculation of U_{sensor}.

The degree of compatibility and consistency of requests and sensors is calculated according to Table. The scoring formula for sensors is obtained according to (6), using the two parameters related to the available resources and consistency of request and sensor.

$$\text{sensor score} = \left(w_{\text{compatibility}} * \ C_{\text{request}-\text{sensor}}\right) + \left(w_{\text{availability}} * \text{Available Resources}\right) \quad (5)$$

At the time of allocation of each request, a scoring formula is calculated for each sensor. The sensor obtaining the maximum score is selected as the appropriate sensor, and the relevant request is allocated to this sensor. Each of these parameters has coefficients $w_{\text{compatibility}}$ and $w_{\text{availability}}$ that indicate its effect in calculation of a general score for this sensor.

The value of these coefficients is selected by the balancer manager, based on the adopted decisions. For example, $w_{\text{availability}}$ is taken into consideration more for a sub-cloud which is focused on reduced cost of hardware resources. If the compatibility of cloudlets (requests) and a particular intrusion detection sensor are assigned higher significance, the coefficient of $w_{\text{compatibility}}$ will have a higher value. The adoption of weighting coefficients will differ based on the system that the algorithm is being applied in, details of which are out of this paper's scope. The quality achieved of studying data packets and successful identification of attacks with intrusion detection sensors has a direct relationship with the current load of sensor. Intrusion detection sensors with normal loading are more efficient than those in an overflow state. Generally, an IDS will have two option for dealing with input traffic based on the sampling rate [48]:

1) **Sampling with a Static Rate**: A certain number of input packets are studied during specific time intervals.
2) **Adaptive Sampling**: The sampling rate is changed based on the current load and sensor status.

In the static sampling method, utilization of sensors is increased with increased input traffic. Intrusion detector sensor efficiency will be decreased as a result of overflowing.

In the adaptive sampling method, this efficiency drop will be found to be relatively "less" compared with the increase of input traffic, because the packets' sampling rate is reduced when the sensor is overflowed, which prevents overflowing of the relevant buffer. For example, when utilization is between 70 and 80 percent, only one of 100 packets will be analyzed. For increased productivity, the degree of sampling is adaptive, to avoid overflowing of buffers and resulting severe loss of efficiency [48]. Snort and Suricata as standard IDS types use these two methods [49], [50].

6.2. Implementation of the Proposed Architecture

In Fig. 3 shows the proposed architectural. The cloud environment is considered to be a complex of sub-clouds, with each sub-cloud consisting of a distributor, a number of intrusion detection sensors and a

number of end users. Input traffic is guided as HTTP requests to an input gate. Following Fig. 1, requests are tagged based on their various types, and are guided toward a distributor. If an input request belongs to a particular session (to which its requests have been allocated in the past), this request is allocated to the previously selected sensor; otherwise, the relative sensors of intrusion detection are scored and finally, the desired traffic is guided to the sensor with the highest score. This method provides the possibility of detecting attacks that require processing of a session to identify them. This feature had not been considered in previous works. The calculation of the scoring formula is carried out according to the following hypotheses and points, with minimum overhead:

- Calculation of a scoring formula for sensors is done at the time of allocation of each request (that is, in real time).
- The $C_{\text{request-sensor}}$ parameter can be calculated at the time of entrance of a request, in a suitably short time, because it is obtained from a comparison of the type of input request and the type of sensors.
- Weighted parameters are always are at hand, having been selected by the designer of the network based on the desired approach.
- The $C_{\text{request-sensor}}$ parameter is sent by sensors to the balancer every five seconds, with the value of the Available Resources parameter calculated by the balancer. In some methods, the U_{sensor} value is collected by the balancer at the time of allocation of the request. This method is not suited for balancer determination, due to the varying delay involved in the sensors sending a U_{sensor} value, and in different receiving times, which is considered as time overhead for the calculation of the score [5], [32], [34].

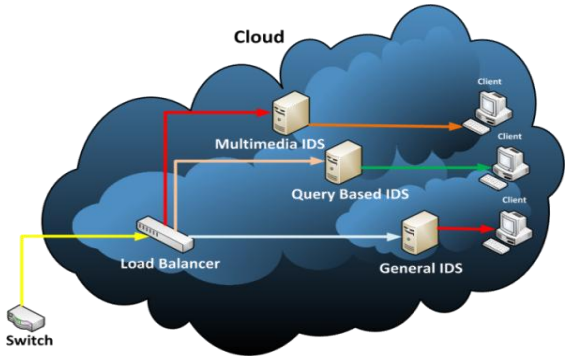

Fig. 3. General scheme of the proposed architecture.

The proposed architecture can be divided into subgroups, from the moment of traffic entrance to the time of reaching end user; details are shown in Fig. 3.

6.3. Simulation of the Proposed Architecture

CloudSim has been used for the simulation of the proposed method due to different reasons such as possibility of personification and use of workload. Simulations have been conducted on a system with the specifications of Table 2.

The Lawrence Livermore National Lab (LLNL) workload is used for simulation of the proposed method. Since sensors are defined within the framework of VMs, and its types using particulars of that VM, simulation of requests are extracted from the LLNL standard workload. Considering the parameter of Submit Time, and given the relevant sub-cloud, the load is guided towards the relevant load balancer of that sub-cloud. It should be noted that the load distributor undertakes the responsibility of allocation of each request to the proper intrusion detection sensor. In this, the main architectural part of load balancer is used. In the beginning, the App ID parameter of each request is studied. The relevant request type is identified using this parameter (video, audio, etc). A score is allocated to all VMs using the scoring formula. Finally, the

sensor which has obtained the maximum score is selected as the most suitable sensor.

Table 2. Specifications of the System Used for Simulation

Processor	
Processor type	Intel Core i5
Processor speed	2400 MHz
Storage	
Drive size	500 GB
Hard Drive Speed	7200.0 RPM
Memory	
Memory size	6000 MB
Memory type	DDR3
Memory speed	1333.0 MHz

```
Step1:  Redirect HTTP Requests to Web API
Step2:  Extract Type of Requests by Web API
Step3:  Add Type of Request to HTTP Header by Web API ( Use "X-" convention or new convention)
Step4   Navigate The Customize Request to Proposed Load Balancer
Step5:  Define  → HistoryOfSessionID&Sensor = Map<SessionID,Sensor>;
Step6:  IF( Request.SessionID is Repetitive){
// Allocate The request to Selected Sensor that accepted requests with this session ID
                SelectedSensor = HistoryOfSessionID&Sensor.findSensor (Request.SessionID);
   }else {
Define → HistoryOfSensor&SensorScore = Map<Sensor,SensorScore>;
For( All Sensors ){
Calculate  Sensor_Score = ( w_compatibility * C_request−sensor ) + ( w_availability * Available Resources );
HistoryOfSensor&SensorScore.Add(Sensor,Sensor_Score );
}
SelectedSensor = FindSensorWithMaxSensorScore(HistoryOfSensor&SensorScore);
HistoryOfSessionID&Sensor.Add(Request.SessionID, SelectedSensor);
}
Step7 : Allocate Request to SelectedSensor
Step8 : The SelectedSensor Analyzes the Request and Generates Different kind Of Alarms
Step9: IF ( Request is Safe){
Navigate Request to End User
}else {
Drop Request }
```

Fig. 4. Pseudo-code of the proposed architecture.

To simulate an IDS in CloudSim, common IDS anomaly definitions are used. Two samples of the most famous anomalies, studied in Snort and Suricata, have been simulated in the cloud context [51]:

1) **Alpha Flows [52]**: Transfer of data and voluminous requests in the form of a Point-to-Point Attack, with addresses of specific origin and destination.
2) **Flash Crowds Anomaly [53]**: Various requests from the originators are distributed to a specific destination, explosively over a time interval.

Any other anomaly can be inserted and simulated thanks to the available requirements; indicating an improvement on results instead of a negative influence.

Table 3. Specifications of Simulated Intrusion Detection Sensors

Type of Sensor	QTY	Specifications
Based on bandwidth	4	10 GB Bandwidth , 2 GB Ram , 10000 Mips , 6 Pes
General	4	10 MB Bandwidth , 1 GB Ram , 10000 Mips , 2 Pes
Based on CPU Load	4	100 MB Bandwidth , 4 GB Ram ,1000000 Mips , 16 Pes

Test Scenario: Two data centers have been modeled, so that two hosts will exist in each scenario. The first

host of each center has higher processing power, while the second host owns more bandwidth. Simulation of an actual cloud environment is the main objective of using two data centers with different hosts, allowing testing of the allocation of input loads to various data centers. Multimedia requests will usually require considerable bandwidth, and database requests have a high processing load. 12 sensors (VM) are placed in each host, per Table 3:

- **Static Sampling Scenario**: sampling every five-packets.
- **Adaptive Sampling Scenario**: Given the general utilization distance, sensors have been considered as follows:
  a) 1/5 packets should be analyzed in less than 60% of utilization.
  b) 1/20 packets should be analyzed between 60 to 70% of utilization.
  c) 1/200 packets should be analyzed between 70 to 80% of utilization.
  d) 1/2000 packets should be analyzed between 80 to 90% of utilization.
  e) 1/20000 packets should be analyzed above 90% of utilization.

6.4. Comparison and Evaluation of the Proposed Method and Other Previous Methods

As discussed in Sections 2 and 3, using single-sensor IDSs in a cloud context is impossible, due to high traffic. To demonstrate this important effect, a single-sensor IDS will be simulated, and compared with the explained specifications. Here, along with a single-sensor IDS, a rotary load balancing method and the RTSLB weighting method (Weighted Algorithm) [34] are simulated and compared using the proposed method.

Comparison of the methods is summarized within the framework of IDS efficiency evaluation criteria and the balancer, details of which have been explained in sections 1 to 4 and 2 to 4. Also, each of two sampling modes is taken into consideration:

1) **Comparison of True Positive Alarm Criterion**: With increasing value of this parameter, efficiency of an IDS sensor is improved.

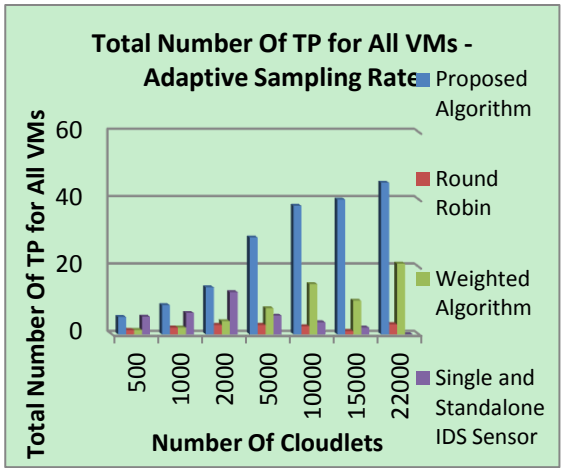

Fig. 5. Rate of true positive alarms vs. the number of cloudlets (requests) using adaptive sampling method.

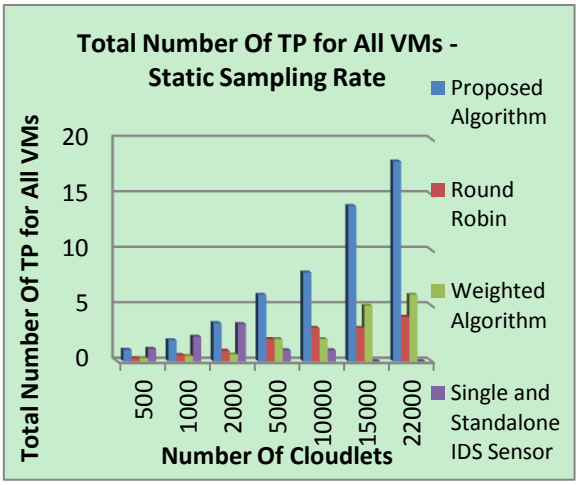

Fig. 6. Rate of true positive alarms vs. the number of cloudlets (requests) using static sampling method.

As it can be observed from Fig. 5 and Fig. 6, efficiency of the single-sensor model drops severely with the increase of input traffic, so that its true alarm rate will decline as well. Such an efficiency drop is more vivid in the static sampling model, because the pertinent buffer is saturated with the increased number of requests. Inevitably, it starts abandoning packets. Consequently, attack detection capability will be reduced. In adaptive sampling, a drop of the alarm rate is observed with an increased number of requests. This drop

is found to be less at the time of overflowing sensors, due to the reduced rate of sampling.

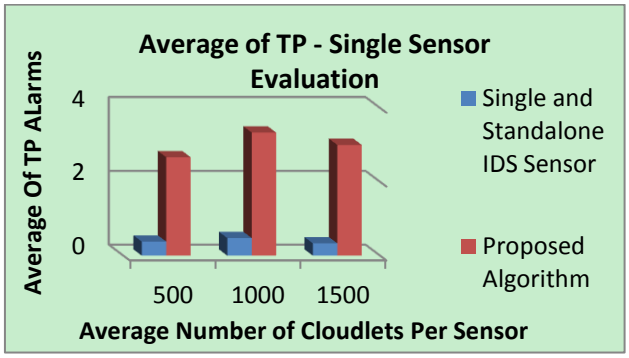

Fig. 7. Average rate of true positive alarms vs. the average cloudlets (requests) of each sensor.

In the Round Robin and Weighted Algorithm methods, RTSLB offers less efficiency than the proposed method, because of the inability to manage attacks based on session, as well as dealing with load imbalance during increased request rates. These two methods use "numbers of requests" for distribution directly so their efficiency can be severely compromised by request diversity, leading to unnecessary overflowing of sensors. Fig. 7 illustrates a comparison of the use of each sensor separately in the proposed method, and single-sensor use. In this comparison, the single-sensor mode has been configured through consideration of various hardware specifications used in the proposed method. According to the figure, the proposed method will present better efficiency, due to the distribution of requests at the time of sensor overflow, and its accounting for sensor type. Consequently, the average rate of identified attacks for the same true positive alarms turned out to be better using the proposed method by considering sensor type.

2) **Comparison of False Positive Alarm Criterion**: This type of alarm indicates a sensor's false detection, identified as false alarms. Reduction of this parameter indicates better sensor efficiency.

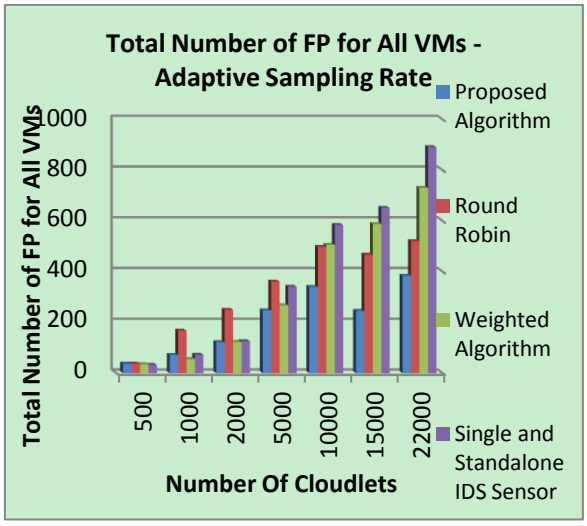

Fig. 8. Rate of false positive alarms vs. number of cloudlets (requests) using adaptive sampling method.

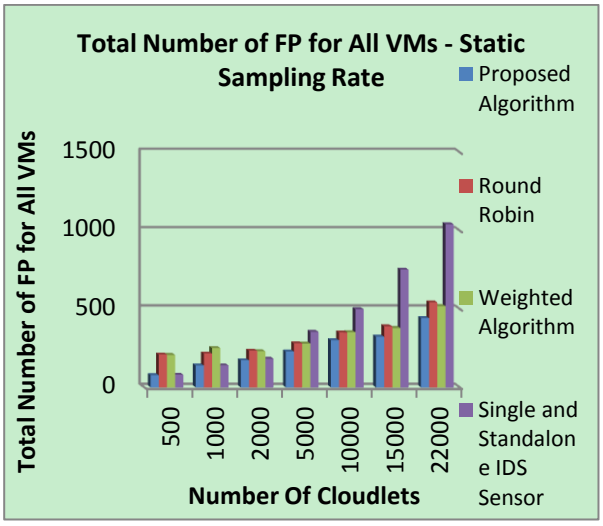

Fig. 9. rate of false positive alarms vs. number of cloudlets (requests) using static sampling method.

Analysis of this parameter is similar to the procedure for true alarms. The single-sensor model has suitable efficiency (similar to the proposed method), but with increasing number of requests, the static sampling model starts abandoning packets due to the saturation of the sensor buffer. Intrusion detection sensors usually produce alarms at the time of abandoning packets. (Recall that a denial of service occurs

with increased traffic) [51].

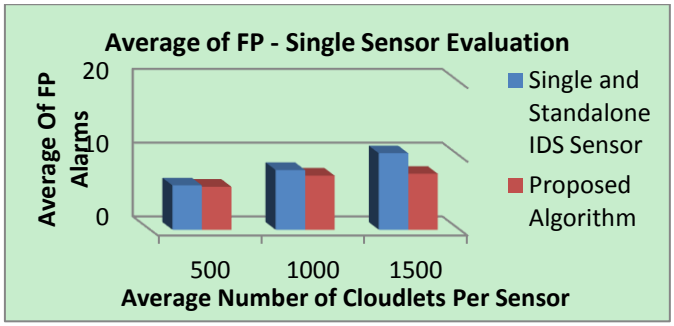

Fig. 10. Average rate of false positive alarms vs. the average cloudlets (requests) of each sensor.

Since by definition a false positive means no attack has occurred, it should be considered as a false alarm. Analysis is the same as for the adaptive sampling model, with the difference that trending of sensor overflow will decrease due to the reduced rate of sampling.

In Fig. 10, average false positive alarm rates for each sensor are shown individually, increasing with a larger average number of cloudlets. An increasing trend of this parameter will occur with sensor overflow. The proposed method transfers load to another sensor, based on the recommendations of the scoring algorithm. Consequently, a negative average rate of positive alarms will have slower rate than the single-sensor mode. Similar to the analysis for true alarms, the proposed method has better efficiency and performance than other methods. Also, this improvement is more vivid in adaptive sampling model.

3) **Degree of Imbalance (DI)**: This parameter is based on balancer evaluation parameters, which are defined within the framework of the three categories of processor, main memory and bandwidth. From the definitions of 2-4 part, it can indicate imbalance of the processing load, memory use and consumable bandwidth.

4)

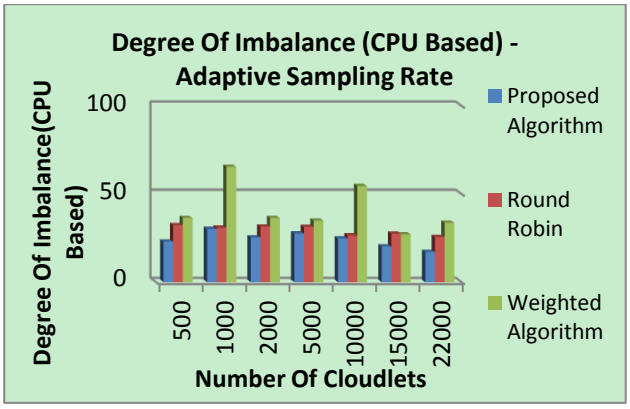

Fig. 11. Degree of imbalance (DI) of processing load vs. the number of cloudlets (requests) using adaptive sampling method.

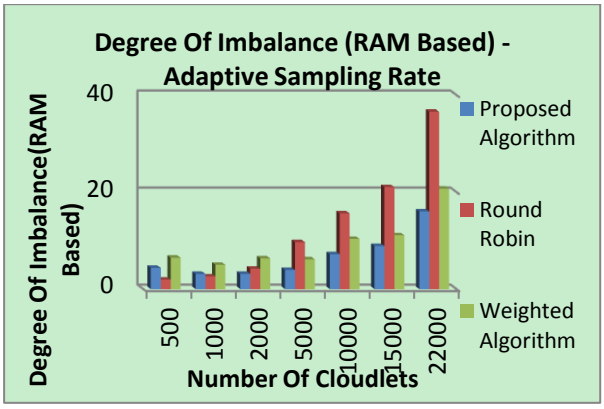

Fig. 12. degree of imbalance of memory load vs. the number of cloudlets (requests) using adaptive sampling method.

The ***Weighted Algorithm, RTSLB*** shows a higher imbalance relative to other methods, due to the rate of complicated processing, which is carried out for determining the selected sensor. In the same way, the proposed and round robin methods will be more appropriate due to the greater simplicity of the selection process. Also, the ***RTSLB*** method will be relatively inefficient, due to the calculation of a parameter (number of cloudlets) during an interval of diversified requests, with different processing volume. The round robin and weighted algorithm (RTSLB) methods are insensitive to the volume of memory consumed in the

execution of requests. For this purpose, the value of memory consumed will have an irregular distribution. The degree of imbalance parameter will be interpreted based on bandwidth. As was mentioned in the introduction, a cloud environment exhibits different types of cloudlets (requests), with varying hardware specifications and requirements. Consequently, identification of these conditions can bring about considerable improvement. Considering that our proposed method uses parameters indicating the utilization of the processor, memory and bandwidth in calculation of the sensor scores, the load allocated to sensors will stand at an appropriate level in terms of these three resources. The management of coefficients of these parameters will safeguard keeping the degree of imbalance within reasonable limits.

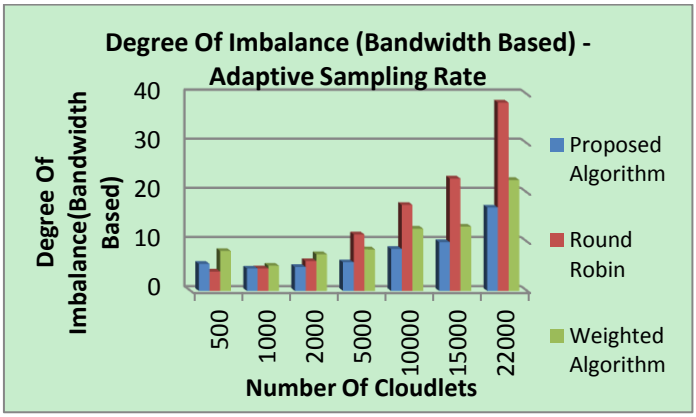

Fig. 13. Degree of imbalance of bandwidth load vs. the number of cloudlets (requests) using adaptive sampling method.

5) **Load Balancing Metric (LBM)**: This parameter can be used to evaluate the system balance.

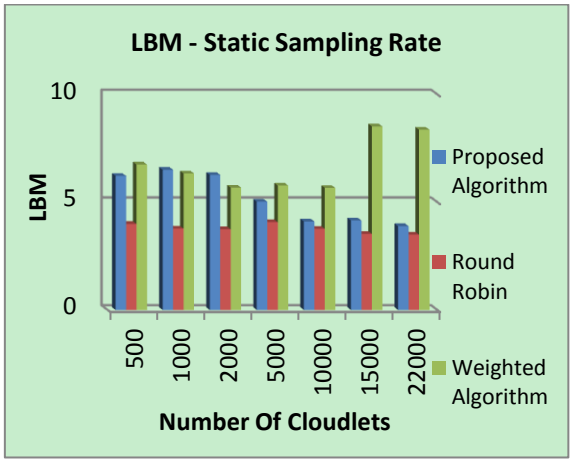

Fig. 14. Value of LBM vs. the number of cloudlets (requests) using static sampling method.

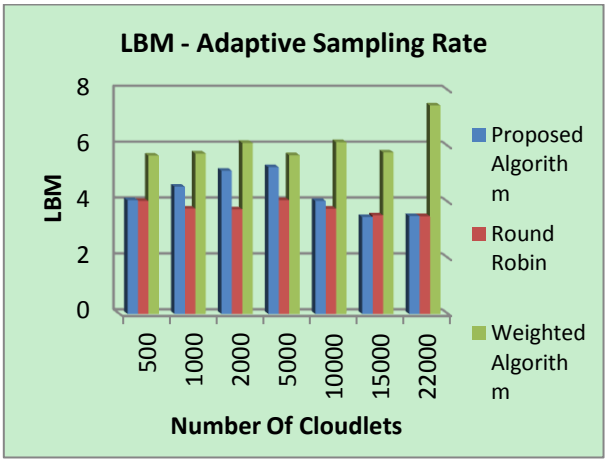

Fig. 15. Value of LBM vs. the number of cloudlets (requests) using adaptive sampling method.

Since this parameter considers only maximum utilization and productivity of a processor, it does not show an appropriate indication of the general balance of the load. The peak–load parameter is instantaneous, and does not give results from an integrated and distributed perspective. Instead, it can merely be inferred that a RTSLB method has an improper balance in terms of the instant peak–load, while the proposed and round robin methods impose less burden on sensors during the specific time interval of the requests (cloudlets).

6) **Makespan**: This parameter is defined based on the completion of requests (cloudlets).

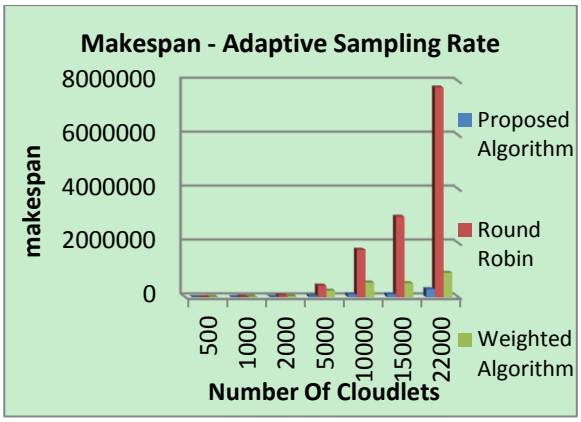

Fig. 16. Value of Makespan vs. the number of cloudlets (requests) using adaptive sampling method.

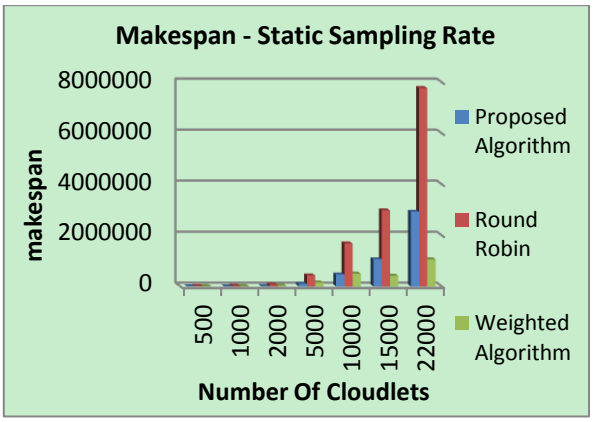

Fig. 17. Value of Makespan vs. the number of cloudlets (requests) using fixed or static sampling method.

As mentioned above, the completion time for each cloudlet depends on factors of the entire system, such as processor capability, memory, bandwidth, etc. To complete each request, its execution requirements (re-processing, value of required memory, requested bandwidth) should be provided. If none of these requirements are prepared, the pertinent cloudlet (request) will be stored in a sensor buffer, allowing all prerequisites to be met.

For this purpose, the completion time for cloudlets (request) is defined. However, in the static sampling model, the average waiting time will be enhanced, due to the prolongation of sensors' buffer data. Consequently, the Makespan criterion will drop a bit more. In contrast, when an adaptive sampling model is used, the completion time for cloudlets (requests) will be characterized by their execution time. The length of sensors' queue is reduced during overflow states, due to the adaptive sampling strategy. Eventually, the proposed method will provide a more appropriate Makespan.

7. Conclusion

The popularity of cloud computing and its associated services, has led to increased security concerns as a particularly challenging issue. The nature of cloud systems requires new approaches to overcome security and attack threats. Even traditional network security solutions and tools could benefit from modifications and changes. For this reason, IDS, as one of the most common and inseparable components of networks, must be adapted to these environments. Distributed IDS use has been introduced, because of the dynamic nature and high-traffic volume of cloud environments. However, this distribution imposes the need for a mechanism to balance the load of IDS sensors, to prevent overflow situations in some sensors while others are underflowed. This will lead to better performance and more efficient resource utilization. On the other hand, an overflow of IDS sensors can severely degrade their functionality. Also, different sensors, which do not necessarily have suitable processing ability, bandwidth and memory, can be used in a cloud environment.

In this article we presented a weighted load balancing method, suitable for a cloud environment, and offering proper distribution of traffic among intrusion detection sensors. This allows functionality according to the characteristics and challenges of cloud computing architecture.

In this method, a specific weighting is given to each sensor by defining two parameters: 1) the compatibility of the type of request and the intrusion detection sensor; 2) the current state of the sensor in terms of available resources. The most suitable sensor will be chosen according to the weighting and prioritizing process, and the relevant request will be assign to this sensor. The advantage of this method is that request-sensor compatibility is assessed and matched, rather than focusing only on the hardware

diversity of intrusion detection sensors. Also, the current state of available sensor resources is appropriately considered, in terms of processor, bandwidth and memory. With this scheme, and in considering the history of packets of the same connection session, load balancing will not adversely affect alarm detection of the IDS. The simulation results show that with every parameter considered for evaluating a load balancer and IDS functionality, our proposed method has superior results.

References

[1] Somasundaram, G., & Shrivastava, A. (2009). *Information Storage and Management*. Wiley Publishing, Inc.

[2] Heorhiadi, V., Reiter, M. K., & Sekar, V. (2012). New opportunities for load balancing in network-wide intrusion detection systems. *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies* (pp. 361-372). ACM.

[3] Katyal, M., & Mishra, A. (2014). *A Comparative Study of Load Balancing Algorithms in Cloud Computing Environment*. arXiv preprint arXiv:1403.6918.

[4] Rajguru, A. A., & Apte, S. S. (2012). A comparative performance analysis of load balancing algorithms in distributed system using qualitative parameters. *International Journal of Recent Technology and Engineering, 1(3).*

[5] Hamo, A. Y., & Saeed, A. A. (2013). Towards a reference model for surveying a load balancing. *IJCSNS, 13(2)*, 42.

[6] Sethi, S., Sahu, A., & Jena, S. K. (2012). Efficient load balancing in cloud computing using fuzzy logic. *IOSR Journal of Engineering*, 2250-3021.

[7] Radojević, B., & Žagar, M. (2011, May). Analysis of issues with load balancing algorithms in hosted (cloud) environments. *Proceedings of the 34th International Convention* (pp. 416-420). IEEE.

[8] Sharma, S., Singh, S., & Sharma, M. (2008). Performance analysis of load balancing algorithms. *World Academy of Science, Engineering and Technology*, *38*, 269-272.

[9] Kokilavani, T., & Amalarethinam, D. D. G. (2011). Load balanced min-min algorithm for static meta-task scheduling in grid computing. *International Journal of Computer Applications, 20(2)*, 43-49.

[10] Patel, P., Bansal, D., Yuan, L., Murthy, A., Greenberg, A., Maltz, D. A., & Kim, C. (2013). Ananta: Cloud scale load balancing. *ACM SIGCOMM Computer Communication Review, 43(4),* 207-218.

[11] Wilcox, T. C. (2008). *Dynamic Load Balancing of Virtual Machines Hosted on Xen*.

[12] Alharkan, T., & Martin, P. (2012). Idsaas: Intrusion detection system as a service in public clouds. *Proceedings of the12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (pp. 686-687). IEEE Computer Society.

[13] Bell, T. (2015, October). Neutron. OpenStack Wiki. Retrieved October, 2015, from https://wiki.openstack.org/wiki/Neutron

[14] Khorshed, M. T., Ali, A. S., & Wasimi, S. A. (2012). A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing. *Future Generation Computer Systems, 28(6),* 833-851.

[15] Kizza, J. M. (2015). System intrusion detection and prevention. *Guide to Computer Network Security*, 273-298.

[16] Gul, I., & Hussain, M. (2011). Distributed cloud intrusion detection model. *International Journal of Advanced Science and Technology, 34*, 71-82.

[17] Kameda, H., Li, J., Kim, C., & Zhang, Y. (2012). Optimal load balancing in distributed computer systems. *Springer Science & Business Media.*

[18] Mesbahi, M., Rahmani, A. M., & Chronopoulos, A. T. (2014). Cloud light weight: A new solution for load

balancing in cloud computing. *Proceedings of 2014 International Conference on Data Science & Engineering* (pp. 44-50). IEEE.

[19] Sharifi, A. M., Amirgholipour, S. K., Alirezanejad, M., Aski, B. S., & Ghiami, M. (2012). Availability challenge of cloud system under DDOS attack. *Indian Journal of Science and Technology, 5(6),* 2933-2937.

[20] Koushika, A. M., & Selvi, S. T. (2014). Load balancing Using Software Defined Networking in cloud environment. *Proceedings of International Conference on Recent Trends in Information Technology* (pp. 1-8). IEEE.

[21] Gu, G., Fogla, P., Dagon, D., Lee, W., & Skoric, B. (2005). *An Information-theoretic Measure of Intrusion Detection Capability.*

[22] Malarvizhi, N., & Uthariaraj, V. R. (2009). Hierarchical load balancing scheme for computational intensive jobs in Grid computing environment. *Proceedings of First International Conference on Advanced Computing* (pp. 97-104). IEEE.

[23] Bunt, R. B., Eager, D. L., Oster, G. M., & Williamson, C. L. (1999). *Achieving Load Balance and Effective Caching in Clustered Web Servers*.

[24] Brucker, P., & Brucker, P. (2007). *Scheduling Algorithms, 3*, Berlin: Springer.

[25] Xinidis, K., Charitakis, I., Antonatos, S., Anagnostakis, K. G., & Markatos, E. P. (2006). An active splitter architecture for intrusion detection and prevention. *IEEE Transactions on Dependable and Secure Computing, 3(1),* 31-44.

[26] Colajanni, M., & Marchetti, M. (2006). A parallel architecture for stateful intrusion detection in high traffic networks. *Proceedings of the IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation.* Tuebingen, Germany.

[27] Le, A., Boutaba, R., & Al-Shaer, E. (2008). Correlation-based load balancing for network intrusion detection and prevention systems. *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks* (p. 2). ACM.

[28] Vallentin, M., Sommer, R., Lee, J., Leres, C., Paxson, V., & Tierney, B. (2007). The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. *Recent Advances in Intrusion Detection*, 107-126.

[29] Paxson, V., Sommer, R., & Weaver, N. (2007). An architecture for exploiting multi-core processors to parallelize network intrusion prevention. *Proceedings of the IEEE Sarnoff Symposium* (pp. 1-7). IEEE.

[30] BroCon Group. (2015, August). Bro IDS. *Bro Documentation, 2(6).*

[31] Liu, J. F., & Dong, F. M. (2008). A dynamic adaptive load balance algorithm in parallel intrusion detection system. *Proceedings of International Symposium on Computer Science and Computational Technology: Vol. 1* (pp. 180-183). IEEE.

[32] Andreolini, M., Casolari, S., Colajanni, M., & Marchetti, M. (2007). Dynamic load balancing for network intrusion detection systems based on distributed architectures. *Proceedings of Sixth IEEE International Symposium on Network Computing and Applications* (pp. 153-160). IEEE.

[33] Mashaly, M., & Kühn, P. J. (2012). Load balancing in cloud-based content delivery networks using adaptive server activation/deactivation. *Proceedings of International Conference on Engineering and Technology* (pp. 1-6). IEEE.

[34] Shadrach, D. C., Balagani, K. S., & Phoha, V. V. (2009). A weighted metric based adaptive algorithm for web server load balancing. *Proceedings of Third International Symposium on Intelligent Information Technology Application*: *Vol. 1* (pp. 449-452). IEEE.

[35] Limmer, T., & Dressler, F. (2011). Adaptive load balancing for parallel IDS on multi-core systems using prioritized flows. *Proceedings of 20th International Conference on Computer Communications and*

Networks (pp. 1-8). IEEE.

[36] Wolf, J. L., & Yu, P. S. (2001). On balancing the load in a clustered web farm. *ACM Transactions on Internet Technology*, *1(2),* 231-261.

[37] Baillargeon, P. (2005). *A Method for Adding Multimedia Knowledge for Improving Intrusion Detection Systems*. Thesis (M.S.), Florida Atlantic University.

[38] Valeur, F., Mutz, D., & Vigna, G. (2005). A learning-based approach to the detection of SQL attacks. *Detection of Intrusions and Malware, and Vulnerability Assessment*, 123-140.

[39] Rights, R. F. (2001). SANS Institute InfoSec Reading Room. *Risk, 1*, 27.

[40] Lew, T. (2009). *Sun Cloud API: A RESTful Open API for Cloud Computin*g. CTO, Cloud Computing Sun Microsystems, Inc.

[41] Feedly Developers Group. (2015, August). Feedly Cloud Center. *Feedly API Documentation, 5(2).* Retrieved October, 2015, from https://developer.feedly.com/v3/

[42] IBM Developers Group. (2014, November). Simple Cloud Center. *Cloud API Documentation, 7(3).* Retrieved October, 2015, from https://ibm.com/developerworks/community/blogs/home/tags/cloud?lang=en

[43] Azure Developers Group. (2015, April). Azure Cloud Center. *Cloud Services Documentation, 2(6).* Retrieved October, 2015, from https://azure.microsoft.com/en-us/documentation/services/cloud-services/

[44] Steven, C. M. (2013). REST in the cloud. *IBM Developer Works*.

[45] Dino Esposito. (2014) Content negotiation and Web API. *Microsoft Magazine*.

[46] Oracle White Paper. (2009). Securing web services and service-oriented architectures with oracle web services manager.

[47] Brocade Group. (2015, April). Brocade Load Balancing App. *ServerIron Layer Documentation, 4(1).* Retrieved October, 2015, from http://www.brocade.com/content/brocade/en/backend-content/pdf-page.html?/content/dam/common/documents/content-types/datasheet/brocade-application-delivery-ds.pdf

[48] Hulboj, M. M., & Jurga, R. E. (2007). *Packet Sampling and Network Monitoring.*

[49] Snort IDS Group. (2011, August). Capacity Planning for Snort IDS. *Snort IDS Documentation, 3(1).* Retrieved October, 2015, from http://mikelococo.com/2011/08/snort-capacity-planning/

[50] Suricata IDS Group. (2013, July). Suricata Developers Guide. *Suricata Wiki, 5(4).* Retrieved October, 2015, from https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricata_Developers_Guide

[51] Ning, Z., & Gong, J. (2008). A sampling method for intrusion detection system. *Challenges for Next Generation Network Operations and Service Management*, 419-428.

[52] Sarvotham, S., Riedi, R., & Baraniuk, R. (2001). Network traffic analysis and modeling at the connection level. *Proceedings of Internet Measurement Workshop*.

[53] Jung, J., Krishnamurthy, B., & Rabinovich, M. (2002). Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. *Proceedings of the 11th International Conference on World Wide Web* (pp. 293-304). ACM.

Taha Arian received his B.Sc degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2012 and he is an M.Sc candidate of computer engineering in Iran University of Science and Technology since 2013. He has done several researches on multimedia, cloud computing, network security and load balancing. His current research interests include secure cloud computing, intrusion detection and security management. He

was teacher assistant for advanced programming (Fall 2010) and multimedia (Summer 2011). Presently he is a member of NRG lab where he is active in different industrial projects.

Amir Kusedghi received his B.Sc degree in computer engineering from Shahid Beheshti University and his M.Sc degree from Sharif University of Technology in Iran in 2012. He is a PhD candidate at Iran University of Science and Technology since 2013. He has done several researches on computer architecture, embedded real time systems, multimedia, cloud computing, NFV and computer network architectures. Presently he is a member of NRG Lab.

Bijan Raahemi received his B.Sc. degree in electrical engineering from Isfahan University of Technology, and M.Sc. degree in electrical engineering from Sharif University of Technology, Iran, in 1988 and 1991, respectively. He received his Ph.D. in electrical and computer engineering from the University of Waterloo, Canada, in 1997. Dr. Raahemi then held several research positions in telecommunications industry focusing on computer networks architectures and services, simulation and performance analysis of data networks, and dynamics of the Internet traffic.

His current research interests include data analytics, data mining and machine learning with applications in engineering, healthcare and business, big data analytics, information systems and technologies, data communications networks and services, business intelligence, and systems modeling and performance analysis. Dr. Raahemi's work has appeared in peer-reviewed journals and conference papers. He also holds several patents in data communications. Dr. Raahemi participated in the IEEE 802.3ae Standard Committee (1999-2001), and contributed to the definition of the WAN PHY in 10Gig Ethernet Standard.

Dr. Raahemi is a registered member of the Professional Engineers of Ontario (PEO), a senior member of the Institute of Electrical and Electronics Engineering (IEEE), and a member of the Association for Computing Machinery (ACM).

Ahmad Akbari received his B.Sc. degree in electrical engineering from Isfahan University of Technology, and M.Sc. degree in communications engineering from Isfahan University of Technology, Iran, in 1987 and 1989, respectively. He received his Ph.D. in electrical engineering from University of Rennes 1, Rennes, France, in 1995. He joined the Computer Department of Iran University of Science and Technology in 1995 where he is the head of RCIT(research center for information technology) and head of the department.

Dr. Akbari has established ASPL (Audio and Speech Processing Lab) and NRG(Network Research Group) including graduate students and researchers from different areas of computer and electrical engineering. The research programs in the NRG lab focus on the five main streams of (a) cloud computing (b) big data analytics (c) software-defined network(SDN) and network functions virtualization(NFV) (d) multimedia streaming and signal processing (e) network security and intrusion detection systems(IDS).