Scheduling framework for distributed intrusion detection systems over heterogeneous network architectures

José Francisco Colom, David Gil, Higinio Mora, Bruno Volckaert

PII: S1084-8045(18)30041-9

DOI: 10.1016/j.jnca.2018.02.004

Reference: YJNCA 2063

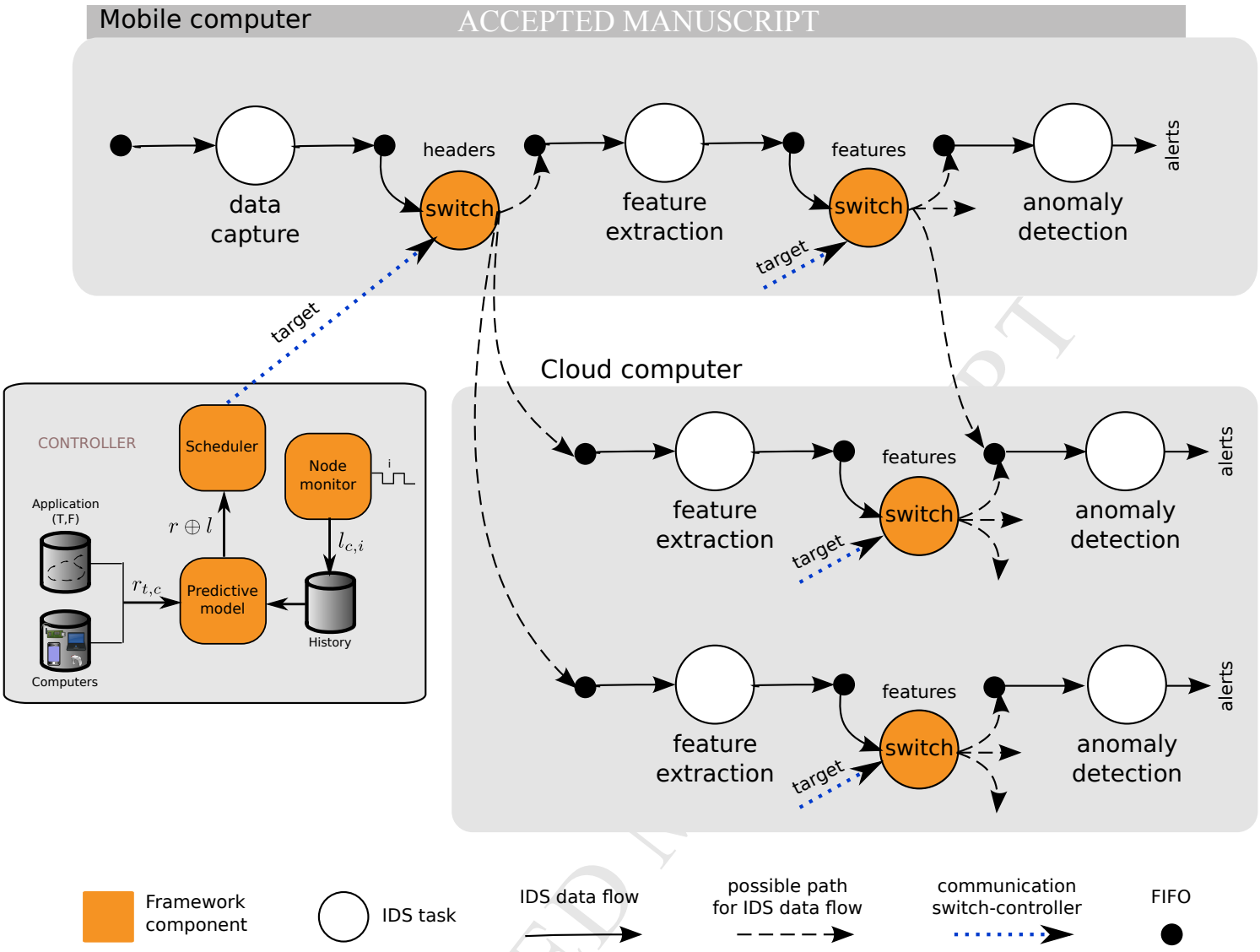
To appear in: *Journal of Network and Computer Applications*

Received Date: 19 July 2017

Revised Date: 22 December 2017

Accepted Date: 2 February 2018

Please cite this article as: Colom, José.Francisco., Gil, D., Mora, H., Volckaert, B., Scheduling framework for distributed intrusion detection systems over heterogeneous network architectures, *Journal of Network and Computer Applications* (2018), doi: 10.1016/j.jnca.2018.02.004.



Scheduling framework for distributed intrusion detection systems over heterogeneous network architectures

José Francisco Colom[a,*], David Gil[a], Higinio Mora[a], Bruno Volckaert[b]

[a] *Department of Computer Science Technology and Computation, University of Alicante, Alicante, Spain*
[b] *Department of Information Technology, Ghent University, Technologiepark-Zwijnaarde 15, B-9052 Ghent, Belgium*

Abstract

The evolving trends of mobility, cloud computing and collaboration have blurred the perimeter separating corporate networks from the wider world. These new tools and business models enhance productivity and present new opportunities for competitive advantage although they also introduce new risks. Currently, security is one of the most limiting issues for technological development in fields such as Internet of Things or Cyber-physical systems.

This work contributes to the cyber security research field with a design that can incorporate advanced scheduling algorithms and predictive models in a parallel and distributed way, in order to improve intrusion detection in the current scenario, where increased demand for global and wireless interconnection has weakened approaches based on protection tasks running only on specific perimeter security devices.

The aim of this paper is to provide a framework to properly distribute intrusion detection system (IDS) tasks, considering security requirements and variable availability of computing resources. To accomplish this, we propose a novel approach, which promotes the integration of personal and enterprise computing resources with externally supplied cloud services, in order to handle the

*Corresponding author
Email addresses: jfcolom@dtic.ua.es (José Francisco Colom), dgil@dtic.ua.es (David Gil), hmora@dtic.ua.es (Higinio Mora), bruno.volckaert@UGent.be (Bruno Volckaert)

security requirements.

For example, in a business environment, there is a set information resources that need to be specially protected, including data handled and transmitted by small mobile devices. These devices can execute part of the IDS tasks necessary for self-protection, but other tasks could be derived to other more powerful systems. This integration must be achieved in a dynamic way: cloud resources are used only when necessary, minimizing utility computing costs and security problems posed by cloud, but preserving local resources when those are required for business processes or user experience.

In addition to satisfying the main objective, the strengths and benefits of the proposed framework can be explored in future research. This framework provides the integration of different security approaches, including well-known and recent advances in intrusion detection as well as supporting techniques that increase the resilience of the system.

The proposed framework consists of: (1) a *controller* component, which among other functions, decides the source and target hosts for each data flow; and (2) a *switching* mechanism, allowing tasks to redirect data flows as established by the controller scheduler.

The proposed approach has been validated through a number of experiments. First, an experimental DIDS is designed by selecting and combining a number of existing IDS solutions. Then, a prototype implementation of the proposed framework, working as a proof of concept, is built. Finally, singular tests showing the feasibility of our approach and providing a good insight into future work are performed.

Keywords: Cyber security, Distributed Intrusion Detection System, Cloud computing, Internet of Things

1. Introduction

Over the past decade, IT environments have become increasingly vulnerable. The evolving trends of mobility, cloud computing and collaboration, have

blurred the perimeter separating corporate networks from the wider world. While increased mobility may make an organisation and its employees more productive, it also creates layers of complexity for securing the enterprise [1].

In the coming years, cyber attacks will almost certainly intensify. Networking technology vendor Cisco Systems forecasts that by 2020, 50 billion devices will be connected to the Internet, including a large portion of industrial, military and aerospace related devices and systems. Each new thing that connects to cyberspace is a potential target for a cyber attack [2].

One of the main approaches to information security and cyber security (see [3] for a discussion about the difference between these two terms) has been the development and deployment of intrusion detection systems (IDS) [4]. An IDS dynamically controls the operations that need to be considered in an environment by monitoring log files, network traffic or other sources. Then, it infers whether these actions indicate an attack or they are usual practices in the environment [5]. Many intrusion detection techniques, frameworks, projects, and products have been developed since the proposal of this approach. Currently, the interest of diverse IDS approaches is growing as shown by the recent works in anomaly detection [6, 7], wireless sensor networks [8], mobile agents [9], new statistical and machine learning techniques [10, 11, 12], smart grids [13], among many others.

Taking into account the current scenario, where the network perimeter is increasingly complex, having a number of instances of IDS processes deployed in select interconnection devices and security solutions, has become ineffective. Furthermore, the classical "insiders" vs. "outsiders" distinction when referring to network attackers could be irrelevant, since an outsider computer can become internal without breaking any physical barrier by means of wireless network attacks (other reasons for fighting insider threats can be found in [14]). Ideally, an effective network IDS should be able to examine all the data flows between all computers regardless of its position in relation to corporate firewalls. Figure 1 gives a simplified view of this evolution of the connectivity and the implications on the required security processes.

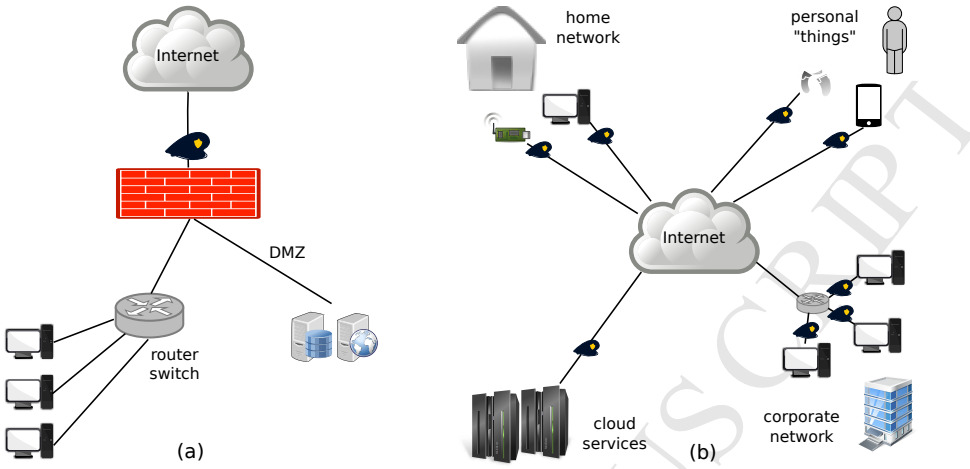


Figure 1: Conventional (a) and trendy (b) connectivity patterns. As connectivity moves forward, all nodes should incorporate security processes for intrusion detection.

The proliferation of radio frequency identification (RFID), wireless sensor network (WSN), and smart mobile technologies, among others, in a communicating-actuating network creates the Internet of Things (IoT). In this situation, sensors and actuators are conjunct with the environment that surrounds us in order to share information across platforms [15]. In a typical IoT scenario, sensor devices with limited computing power and a specific purpose, interact with general purpose computers with relatively high computing power and capable of addressing a diverse task load (laptops and desktop computers, among others); moreover, they can utilize the almost limitless (albeit at a price) supply of computing power brought by cloud vendors [16].

In order to maximize the protection offered by an IDS, IoT devices must participate in the detection process by capturing, filtering and analysing log records and network traffic. However, such a distributed IDS configuration comes with a major drawback: the computing resources embedded in user devices are intended to serve mainly user applications, but some IDS processes are usually resource-hungry which could compromise device performance or energy consumption, and therefore, deteriorate user experience. Fortunately, this load

can be alleviated by offloading part of the processing to the cloud, although potentially incurring other drawbacks, such as increased cost or additional security problems.

This work aims to provide a framework to properly schedule IDS tasks, considering security requirements and variable availability of computing resources in the devices. The presented approach favours the integration of personal and enterprise computing resources with externally supplied cloud services, in order to fit IDS requirements. Moreover, this integration is achieved in a flexible way: cloud resources are used only when necessary, minimizing utility computing costs while at the same time preserving local resources when they are required for business processes or user experience. In other words, different IDS activities are distributed throughout the network, taking the best of both worlds: local execution with own computing resources and remote execution through cloud-supplied services.

The rest of the paper is organized as follows. First, a review of current IDS related research areas and their relevant architectural solutions is presented. Then, the framework design is proposed by formally defining the problem and providing a conceptual view of the solution. After that, the experimental design is addressed and tests are conducted, and the feasibility of the solution is verified. Finally, relevant conclusions and future directions of the research are drawn.

2. Related work

With the arrival of anomaly-based intrusion detection systems, there are many approaches and techniques which have been developed to track novel attacks on the systems. The scope of the review presented in [17] encompass core methods of computational intelligence, including artificial neural networks, fuzzy systems, evolutionary computation, artificial immune systems, swarm intelligence, and soft computing. Machine learning methods are having a relevant presence, especially in the anomaly detection area. For example, in [18], mutual information is successfully explored to perform feature selection; in [19], the

current state of ensemble-based methods in modern IDS is presented; in [20], decision trees and support vector machines are combined to model intrusion detection systems.

Alert correlation in collaborative IDS is another important approach; a number of algorithms and methods have been described in [21] and [22]. An IDS may be adopted and implemented to detect attacks and alert the operators triggering a human intervention. However, the resource-constrained nature of certain environments (Cyber-physical Systems, Internet of Things) presents a challenge, since a reliable and accurate IDS can be computationally expensive [23]. Consequently, computational nodes may not be able to perform intrusion detection uninterruptedly. This leads researchers to devise a distributed approach to intrusion detection [24].

In this article, the problem of deciding which devices process which parts of the IDS system is tackled, and this in computer networks where intelligent sensors and mobile devices can exchange information with servers through cloud services. In such an environment there is no a permanent corporate firewall which hosts the IDS processes, but each device is responsible for its own protection against intrusions. On the other hand, machine learning IDS methods as mentioned above require computing resources; these resources can be provided by the device itself or by other computers, with the corresponding advantages and drawbacks. The proposed model allows to dynamically switch among different resource providers considering different parameters, e.g. performance, operational cost or energy efficiency.

Intrusion detection has been extensively studied for servers, corporate networks and cloud resources [25], where computational power is not an important issue. However, applying traditional IDS techniques to IoT is difficult due to its particular characteristics such as constrained-resource devices, specific protocol stacks, and standards; a recent taxonomy of the proposed solutions and future trends can be found in [26]. In [27], a survey of IDS using recent ideas and methods proposed for the IoT is also presented; this work illustrates IDS platform differences and the current research trend towards a universal,

cross-platform distributed approach. Our proposed framework aims to allow execution of proven techniques, distributed on a diversity of devices, globally interconnected in an IoT setting, where resource scarcity remains a big challenge.

Another important issue is trust management, which plays an important role in IoT for reliable data fusion and mining, qualified services with context-awareness, and enhanced user privacy and information security. However, current literature still lacks a comprehensive study on trust management in IoT. The paper presented in [28] investigates the properties of trust, propose objectives of IoT trust management, and provide a survey on the current literature advances towards trustworthy IoT.

The paper presented by [29], survey different intrusions affecting availability, confidentiality and integrity of Cloud resources and services. Consequently, the authors examine structures of Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) in the Cloud. They recommend IDS/IPS in Cloud environment to accomplish desired security in the next generation networks. Currently one of the remarkable solution for industrial systems is the integration of cyber physical system (CPS) with the Internet of Things (IoT) utilizing cloud computing services.

The focus of the study presented in [30] is to highlight the security challenges that the industrial systems of supervisory control and data acquisition face in an IoT-cloud environment. The classical systems are already lacking in proper security measures; however, with the integration of complex new architectures for the future Internet based on the concepts of IoT, cloud computing, mobile wireless sensor networks, and so on, there are large issues at stakes in the security and deployment of these classical systems. Integration of Cloud and IoT, which is called the CloudIoT paradigm is also reviewed in [31]. Although, many works in literature have surveyed Cloud and IoT separately these works lack a detailed analysis of the new CloudIoT paradigm, which involves completely new applications, challenges, and research issues. Accordingly this paper provides a literature survey on the integration of Cloud and IoT.

The work presented in [32] deals with another interesting problem in the IDS area: the lack of datasets to carry out effective comparisons. Sometimes these datasets are internal and therefore, due to privacy concerns, are not opened to the public; others are anonymized without clear trend indications. The authors therefore propose a systematic approach to generate the datasets to address this requirement.

A lot of research has been performed on scheduling strategies and mapping heuristics. New frameworks are still being proposed for specific types of applications (recent examples can be found in [33, 34]), where proper use of heterogeneous distributed resources remains a challenge. According to these works, heterogeneous computing environments are capable of executing distributed IDS (DIDS).

In summary, current global computer systems combining IoT with Cloud services require a novel mapping of state-of-the-art techniques, collaborative methods and algorithms for IDS. The main contribution of this work is the proposal of a novel framework capable of dynamically distributing many IDS related tasks over a heterogeneous network. This includes mobile devices and cloud services, in such a way that user experience is not compromised by the resource consumption of the IDS, while at the same time guaranteeing that all relevant data flows are passed through the IDS.

3. Proposed distributed IDS framework

First, a distributed IDS is characterized as a set of processes which exchange data flows. Then, a model is defined to quantify the resources available in each computer and the requirements of each process; this model allows to articulate a set of procedures to predict the future load of the involved computers; this information will be used for scheduling. Finally, a conceptual solution is designed, putting all the architectural elements together in order to address possible implementations.

3.1. IDS characterization

The proposed characterization is based on the collaboration and distributed models presented in previous works of the authors [35, 36]. Now, those ideas are reused in order to capture the complexity of an IDS as an application with the following properties: (a) the IDS obtains input data from network interfaces and log files; (b) the work to be performed can be decomposed into a set of individual tasks or processes, which exchange data flows across the network and are executed in parallel; and (c) the results of the processing are translated into a set of actions such as raising alarms, sending reports or blocking access.

In formal terms, the distributed IDS can be represented by a directed graph $A = T, F$ where:

- T is the set of vertices and represents a set of tasks necessary for data capture, processing, storage and actuation. Typically, the $t \in T$ tasks exchange data over a communication network and they cause an increase in the load on the computers on which they execute.
- F is the set of edges, and each $f \in F$ represents a data flow exchanged between two tasks (source and target).

The data flow diagram shown in Figure 2 depicts an example of IDS modeled according to this principle. Table 1 offers additional explanation on each proposed process.

3.2. Resource characterization

The set C is defined as the computing elements on which tasks in T can be executed. The set C effectively constitutes the computational resources in the network in which the distributed IDS is deployed. In addition to intelligent wireless sensors, the set C can include desktop computers, servers, mobile devices, etc. For example, Figure 3 shows a set of resources stemming from the IoT infrastructure, available devices, and hired cloud resources.

In order to achieve an efficient distribution of tasks, the utilization of the resources must be suitably characterized. For this purpose, L is defined as a

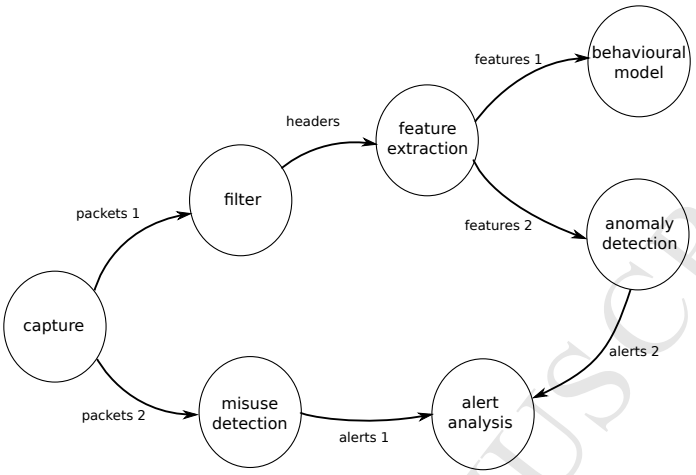


Figure 2: Decomposition in tasks and data flows for a sample IDS

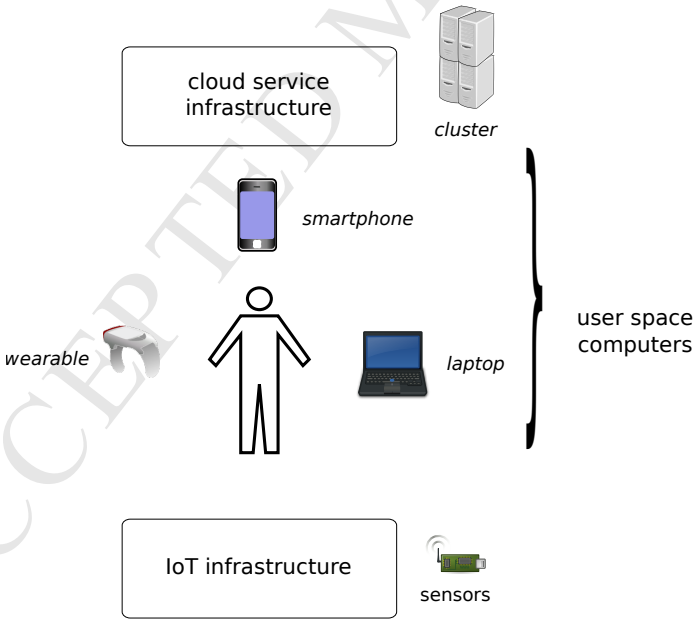


Figure 3: Various resources from IoT to Cloud service infrastructure offering computational power

Table 1: Definition and examples of the tasks for the proposed IDS

Task	**Definition**	**Example**
Capture	Acquisition of all network traffic that need to be monitored	HTTP traffic
Filter	Selection of relevant fields for further anomaly detection	HTTP headers, source and target IP
Feature extraction	Computing features for statistical pattern recognition	Number of incoming connections from a given IP per sec.
Behavioral model	Learning patterns of *normal* network traffic	Common sequences of transactions
Anomaly detection	Pattern recognition to warn about possible attacks	Classification of uncommon sequences of transactions
Misuse detection	Application of rules to detect known attacks	Detection of connections using forbidden protocols
Alert analysis	Correlation of the output from different IDS strategies to reduce false positives	Confirmation of an intrusion attempt

vector domain that groups the relevant features which determine the load of a computer. For example, L can be defined as a domain with two-component vectors in the $[0, 1]$ range with the following semantics:

$$L = Transfer\ rate \times Processor\ load$$

The L components can refer to both computing resources and other constraints such as energy consumption or the economic cost of the service. The components are expressed by a normalized value relative to each $c \in C$; in other words, the value specified for each component determines the fraction of the resource available or required for the execution of tasks in T. Therefore, the elements of L are used to characterize the load of the device, which is variable over time, and also the requirements of each task in each computer available in

the system, which can be determined at design time:

- Load modeling. Each computer $c \in C$ is in a certain state in relation to the load (computational load or other factors). This state is represented by a vector $l_{c,i} \in L$ which describes the load of the computer c at instant i, and hence its ability to execute more processes. Obviously, these values can change at every instant i, depending on the different activities in which the device is involved.

- Requirement modeling. For each computer $c \in C$ and each task $t \in T$, it is necessary to estimate how much the load would increase if c were to execute t. This increment is represented by $r_{t,c} \in L$ (requirements of a task $t \in T$ on a computer $c \in C$).

The values of $r_{t,c}$ effectively model the suitability of a device for a given task. For example, a laptop with a powerful GPU will offer low values for the *Processor load* component for those tasks that require intensive image processing. In other words, $r_{t,c}$ quantifies the degree to which a computer with GPU is suitable for performing image processing services, or the impact of that task on the computer load.

3.3. Load prediction

One of the main elements useful to design an effective scheduling policy is the estimation of the status (load) of each device if some task of the IDS is executed on it. In the proposed framework, this estimation is achieved through an internal binary operation (procedure) defined in L, represented by the symbol $\oplus$. This operator represents a specific procedure for load prediction in the following way:

$$l_{c,i+1} = l_{c,i} \oplus r_{t,c}$$

Figure 4 offers a graphic representation of this procedure.

In general, the execution of $\oplus$ could require more information than is present in its operands. For example, the precise estimation of the transfer rate relative

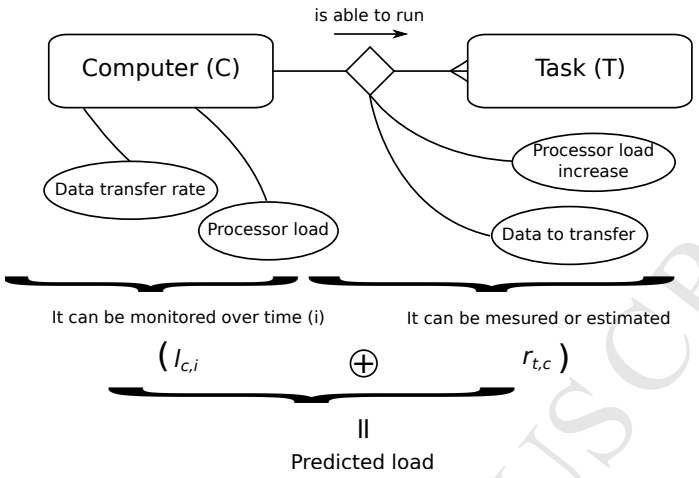


Figure 4: Load prediction coming from the monitored current state and the estimated impact when executing the task

to the available bandwidth may require a prediction of that bandwidth in the next period $i + 1$; to this end, a predictive analysis based on machine learning that uses historical data on the performance of the network can be applied.

3.4. Framework definition

The proposed solution is made up of two main components: (1) special proxy local processes run on each computer, called *switches*, and (2), a system *controller* which maintains a view of the overall system, and offers framework services such as monitoring computers, checking available bandwidth and planning the source and target computer for each required data flow. Figure 5 shows a high level view of the proposed architecture.

The scheduler component is the major element within the proposed architecture of the system controller. The remaining controller components provide the model and information required for the scheduler to determine the target computer for every required stream. The scheduler mainly decides which tasks should be executed on each device. For this, there are several possible strategies, with different level of complexity and degree of optimization in the results.

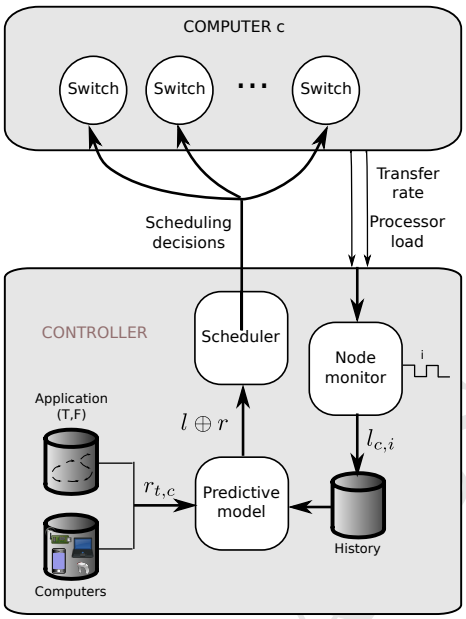


Figure 5: Architecture diagram of the proposed solution

A simple alternative is to define a feasibility function:

$$feasibility : L \longrightarrow \{0, 1\}$$

This function models the actual possibility of working with a given load on each computer. A trivial implementation would get the value 0 (non-feasible) when any of the components of $l \in L$ has any value close to 1, and would get the value 1 (feasible) otherwise. In this case, the problem is to find a correspondence

$$schedule : T \longrightarrow C$$

that satisfies the following expression (in each i cycle)

$$\forall c \in C,\ feasibility\left(l_{c,i} \oplus \sum_{\forall t | schedule(t)=c}^{\oplus} r_{t,c}\right) = 1$$

The computational cost of the optimal solution to this problem is non-polynomial, although it is possible to define heuristics that allow a significant reduction of the response time albeit obtaining suboptimal solutions.

Another key component in Figure 5 is the node monitor: it provides the performance parameters relevant to take scheduling decisions ($l_{c,i}$). This is done in terms of a fraction of free computing resources available on each computer, which can be estimated, for example, from the system load average and data transfer rate. Other user experience factors such as energy consumption could be considered [37]. In addition, the node monitor could integrate different aspects of computer unavailability; for example, when an unexpected shutdown occurs or all its resources are required for processes other than the distributed IDS ones. These situations can be modelled by conveniently expanding L and throwing proper values for $l_{c,i}$, so that the framework scheduler can redirect the corresponding data flows.

To finish with the controller components displayed in Figure 5, the computer status is logged in a historical database; this information is used as input for a predictive model, so the scheduling can be performed not only based on the result of direct measures, but also on mined knowledge (implementation of the $\oplus$ procedure). Recent research has already been done in similar environments. For example, proposals coming from [38, 39, 40], can be integrated into the presented solution.

One computer $c \in C$ is shown on top of Figure 5. Each computer can potentially run any IDS task instance sharing the corresponding data flows, and each data flow is handled by a switch process. The switch processes work as actuators for the scheduler component, transferring the scheduling decisions to the IDS tasks to effectively address data flows. Figure 6 details the control flow of a single switch process holding one data flow from t_1 to t_2.

Input data are obtained from switch processes or input devices (for example, sensors) for every task instance. Output data are also sent to a switch process, or they can be used directly to indicate an appropriate alert action (i.e. warning administrators, blocking interconnection devices, storing data for further forensic analysis, etc.). The direction of the data flow towards the corresponding task instance will be executed by every switch process and can be located either in the same or different computer. The controller is the component which takes

this decision, since it maintains the model of the overall system and runs a suitable algorithm for scheduling. In case that the controller is not available (i.e. network problems or host breakdown), the switch process executes a fallback procedure, and in that situation it uses only local information.

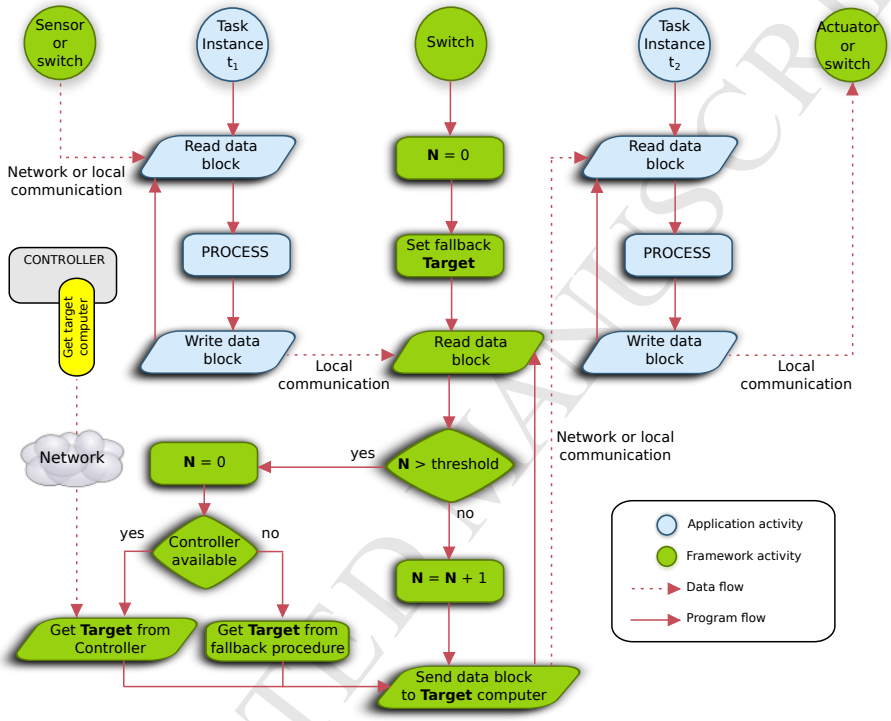


Figure 6: Flow diagram showing two task instances exchanging one data flow through one switch process. The controller, if available, decides the computer where data will be effectively processed

The data block size and the N threshold value are remarkable issues in Figure 6 The data block size may be variable and it can be specified in different ways. For example, for the capture process, one block can hold a fixed number of packet headers, or the number of packet headers captured during a fixed time period.

It is noteworthy how the values of block size and N threshold influence the performance. If both values are high the use of resources is lower, and

consequently, we obtain a better performance. However, if these values are low the accuracy of the system controller will improve; this is due to the possibility to obtain a more precise vision of the current global state, favouring the most suitable decisions.

The higher the block size and N threshold, the lower the use of resources by the framework, and therefore, better performance can be achieved. By contrast, decreasing these values improves the accuracy of the system controller, as it allows to maintain a more accurate picture of the global status, and therefore, more appropriate decisions can be taken.

The IDS processes run continuously by capturing and processing data. The capture process continuously takes data from the network interface, and other processes take their input from the output of other ones. After a number of bytes processed (or elapsed time) modelled by N, the switch architectural element must query the controller in order to find a new target for the output, according to the scheduling algorithm. Therefore, N allows to parametrize the performance/accuracy of the framework elements, and the maximum frequency of scheduling decisions. The period of acquisition of load values captured by the monitor (the difference in time between consecutive i events) must be set in accordance to this value of N, in order to avoid an extra overhead not providing any useful accuracy.

4. Experimental design and results

In this section, an experimental design for the IDS framework is presented. The objective is to build a system that shows the feasibility of the proposed ideas and work as a prototype for proof of concept. First, an experimental IDS is designed by selecting and combining a number of existing IDS solutions. Then, a prototype implementation of the proposed framework is provided. Finally, tests are performed and framework parameters are adjusted to show the feasibility of the approach.

4.1. Experimental intrusion detection system

In order to test the proposed approach, it is necessary to start from a specific IDS implementation which can be deployed using the framework. The Figure 2 can be used as a roadmap to build a distributed network IDS using select state of the art tools and techniques; each node shown in the depicted graph has been mapped into a functionality extracted from an existing project or platform, as summarized in Table 2.

Table 2: Mapping from tasks into implementations for the experimental NIDS

Task	**Functionality**	**Source project**
Capture	Acquiring target traffic from the network interface	Tcpdump
Filter	Extracting headers from selected packages using TShark	Wireshark®
Feature extraction	Preprocessing traffic to produce statistical features	MINDS
Behavioral model	Finding typical patterns in network traffic	Snort.AD
Anomaly detection	Alerting about observations that do not fit the typical patterns	Snort.AD
Misuse detection	Rules matching for known attacks	Snort®
Alert analysis	Correlating alert messages	Hadoop®

Tcpdump with its Libpcap library has been frequently used for capturing network traffic [41, 42]. The experimental configuration employs Tcpdump to acquire the data to be analysed, excluding the ones produced by the IDS tasks, so the IDS does not process data generated by itself (this processing will be useless and time consuming in most environments). The obtained data stream feeds two different IDS techniques: anomaly detection and misuse detection [43]:

- Anomaly detection.— The acquired data stream is filtered using TShark command line tool from Wireshark suite [44], producing a reduced stream (relevant packet headers) required for feature extraction. Then, the features are effectively extracted following the procedure proposed in Chandola et al. [45] and implemented in MINDS (Minnesota INtrusion Detection System) using an R [46] script. Finally, the source code provided by Snort.AD project [47] has been adapted to implement the behavioral model and anomaly detection processes in a standalone way (not integrated with Snort).

- Misuse detection.— The acquired data stream is matched against the list of rules provided by a Snort sensor [48, 49]. Since this process may produce many false positives, further alert analysis using cloud big data clusters have been proposed [50, 51]. For this reason, the experimental configuration includes a Hadoop cluster for correlating alert messages, which takes into account the results raised by anomaly detection [52].

The designed IDS shows that existing products and techniques can be integrated following a data flow model. The objective is not to build a fully operative IDS, but to test the feasibility of the proposed framework. The proposed framework provides a dynamic distribution mechanism for a number of instances of each communicating task: those instances will be run on different computers, according to the available resources.

4.2. Experimental framework implementation

The proposed framework has been validated through a prototype implementation. The prototype is based mostly on Bash [53] scripting, and SSH (Secure Shell) [54] as a communication mechanism for all services.

To begin with, a script has been built for each one of the tasks in the experimental NIDS. Those scripts provide a uniform interface for the framework, taking file names as arguments. The file names are used inside the script to

provide input and output to the corresponding tool implemented according to Table 2.

Next, an initial startup script reads a textual representation of the NIDS diagram (Figure 2), and it runs the corresponding task instances by calling the previous task scripts. New devices are added into the set, by providing a script which, remotely invoked, registers the computer configuration into a database file.

Finally, a prototype for the two main elements of the framework has been developed: the *switch* component (one instance per task instance) and the *controller* component.

4.2.1. Switch component

The prototyped switch process follows the algorithm in Figure 6. It uses SSH to periodically run a command in the controller computer, to find out the target computer for each output data stream.

According to the algorithm, if the controlling computer is not available (it does not respond), a fallback decision must be taken. In this case, the experimental prototype always decides to transfer the stream to a task running locally (this strategy could be further improved).

Figure 7 shows the operation of the switch processes by depicting two example computers and a subset of the tasks of the experimental NIDS. The prototype employs standard Unix FIFO streams along with `ssh` and `dd` (command line copying) to transfer data between task instances and their corresponding switch processes.

4.2.2. Controller component

For building the prototyped controller, the following parameters have been considered:

$$L = Transfer\ rate \times Processor\ load$$

Then, the prototyped controller is made of a set of simplified processes, that implement those represented in Figure 5:

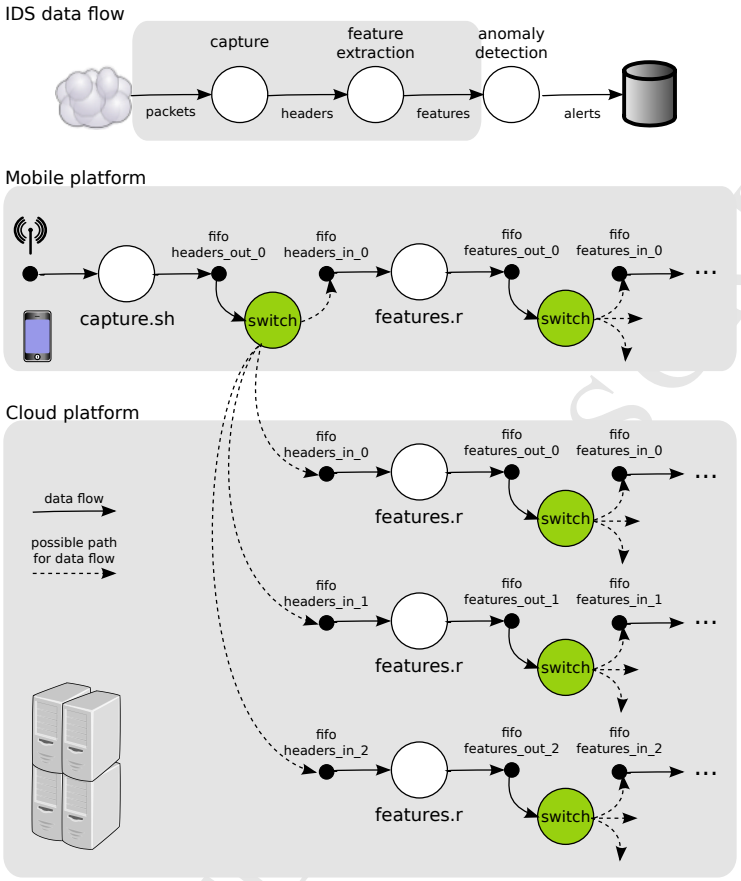


Figure 7: Processes created in two computers for a subset of the prototyped NIDS

- Node monitor.—This element must find out $l_{c,i} \in L$. Namely, it obtains the data transfer rate and the system load average by running `vnstat` and `top`/`uptime` commands, respectively on each target computer, c, using SSH (the system load average as computed by `top`/`uptime` is the average number of processes that are using the CPU, waiting to use the CPU, or waiting for some I/O access). Finally, the results are added into a historical file.

- Predictive model.—For this feasibility study, a basic estimation for $l \oplus r$ is provided: it takes the last capture available in the historical file, $l_{c,i}$, and

adds it to the estimated requirements of the task, $r_{t,c}$. This is done for the two evaluated parameters: data transfer rate and system load average. More advanced approaches could make an intensive use of the historical file, improving the accuracy of the estimation.

- Scheduler.—The decisions of the scheduler are hard-coded and modified manually for the purpose of the experiment; in fact, it just decides to target a remote computer if $l \oplus r$ is found out lower than a predefined tolerance value.

4.3. Test environment for validation

For the purpose of the test, the experimental NIDS has been deployed on a network with three computers. Two of them act as mobile devices (Raspberry Pi computers [1] [55]) and the other one as a server computer (Intel® Core™ based [56]). The hardware used to perform those roles in the network is summarized in Table 3.

Table 3: Computers deployed in the test environment. The three nodes are linked in a standard wireless local area network

Role	Mobile device	Server computer
CPU	ARMv6	Intel® Core™ i5
Memory	512MB	6GB
Computer type	Raspberry Pi	Desktop
Number of units	2	1

The three tasks shown on top of Figure 7 (capture, feature extraction and anomaly detection) have been taken into consideration for providing an illustrative example and pointing out the benefits of the approach. The requirements of each task in each computer have been estimated considering $L = Transfer\ rate \times Processor\ load$, obtaining the following matrix R:

[1] Raspberry Pi is a trademark of the Raspberry Pi Foundation

$$R = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ \\ \end{array} \begin{array}{c} \begin{array}{ccc} t_1 & t_2 & t_3 \end{array} \\ \begin{pmatrix} (0.5, 0.2) & (0.3, 0.5) & (1, 1) \\ (0.5, 0.2) & (0.3, 0.5) & (1, 1) \\ (1, 1) & (0.3, 0.1) & (0.1, 0.2) \end{pmatrix} \\ \begin{array}{ccc} \text{capture} & \text{features} & \text{anomaly} \end{array} \end{array} \begin{array}{l} \\ \text{device} \\ \text{device} \\ \text{server} \\ \\ \end{array}$$

The elements $r_{t,c} = (1, 1)$ correspond to tasks t that will never be executed on the corresponding computer c. Namely, the anomaly detection, t_3, will never be carried out by the device itself, c_1 or c_2, since it has been proven that it does not have sufficient resources, even without task occupation; similarly, the data capture (t_1) will not be done by the server (c_3), since in this test the data traffic accessible by the server is not relevant for the IDS (the data to protect are the ones reaching the devices).

In the switch processes, the framework has been configured with a variable value for N, corresponding to the data packets acquired during 60 seconds. The threshold value has been set to 0, so scheduling decisions are taken every single block, thus, once a minute (N is the data block size shown in Figure 6). This long period has been taken due to the time required by the implementation of the feature extraction process in the single-core mobile device: the deployed R script adds a constant time of 40 seconds to process any data block, including empty data blocks. However, this should not be a limitation in a real environment, as the feature extraction process can be heavily optimized: those additional seconds are not spent when running feature extraction on other platforms, and nowadays it is common to find multicore architectures even in mobile devices [57].

Another important point in the node monitor configuration is the period of acquisition of load values captured (the difference in time between consecutive i events). Certainly, this parameter affects the performance and the optimization grade of the scheduling. In the experiment, these load values are stated using commands such as `vnstat` and `uptime`, which can offer average values acquired in the last minute. Since the prototyped controller does not have to take

scheduling decisions with a frequency higher than 1/60 seconds, this provide a reasonably good estimation of the load status.

For the experiment, two additional problems must be resolved. First, network traffic needs to be simulated for the NIDS to work. Second, common user tasks must be executed on the mobile computer, in order to observe the framework behaviour. The first problem is addressed by running a script based on the GNU Wget [58] tool, randomly producing web navigation traffic; the script simulates a common web navigation session done by a user. For the second problem, this script periodically plays an MP3 audio file, by using VLC® from VideoLAN® project [59].

4.4. Performance results and discussion

The described deployment has been used as a workbench for testing the framework prototype and showing its benefits. Those benefits are given in terms of performance, as the employed tools (Table 2) implement existing techniques put in place to evaluate metrics such as occupied bandwidth or processor throughput. First, one mobile computer is considered in order to evaluate the effect of the framework activity when some extra processing load is required by the user (audio reproduction has been used as benchmark). Second, four mobile computers are considered to evaluate the effect of the framework on the server performance.

4.4.1. Framework performance evaluation on mobile clients

The upper diagram in Figure 8 shows the system load average, as computed by the `uptime` command. The average is taken over the last minute. As can be seen, at the beginning of the test, the packet header flow is sent to the feature extraction task in the mobile computer itself. After that, the system load is increased by running an MP3 audio reproduction task, reaching 3.5 from minute 5. At this point, with the framework enabled, the system load average starts a downward trend, allowing the audio stream to be properly played. This behaviour is in accordance with the framework logic: the framework scheduler

instructs the switch process to send packet headers to the server computer, where the corresponding server task will extract the features, freeing the mobile computer from this load. Finally, after finishing the audio stream reproduction, the system load average goes lower than 1, so the packet header flow is back to the mobile instance of the feature extraction task, as instructed by the framework scheduler.

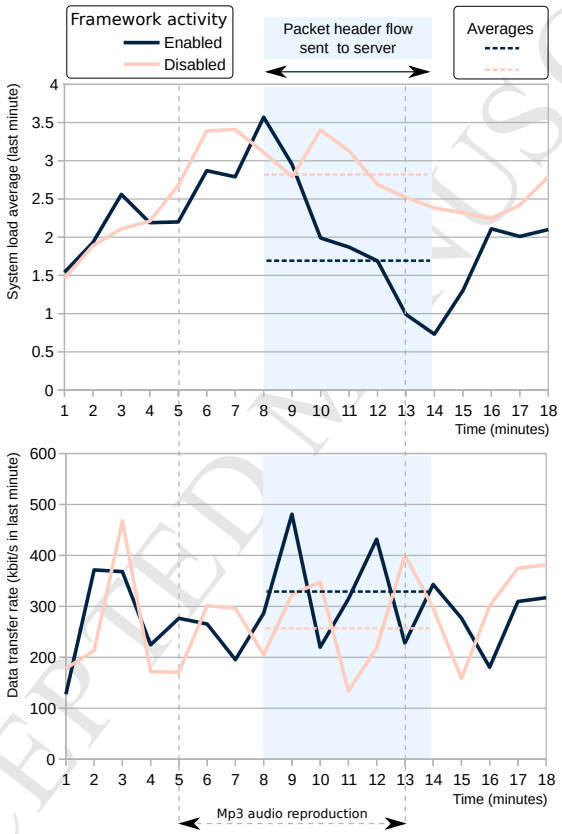


Figure 8: Mobile computer: system load average and data transfer rate, with and without framework activity

The lower diagram in Figure 8 reflects the effect of the framework activity in the bandwidth use. The data transfer rate is increased by a fraction of the analysed flow due to sending packet headers to a remote computer. Since

the wireless network supports a bandwidth of several megabytes per second in the test environment, this increase can be perfectly assumed. In more complex scenarios, other scheduling policies may be put in place, expanding L with other factors: variable free bandwidth in networks with shared transfer media, energy consumption and user preferences, among others.

In the experiment, the user navigation has been simulated by random downloads. For this reason, the experiment has been conducted several times, checking that the essential result remains the same: successful switching of the packet header stream, saving mobile processor time when required by user tasks.

4.4.2. Framework performance evaluation on server resources

In order to evaluate the framework from the server point of view, three experiments have been performed, using now four mobile units and one server, and measuring the amount of data received by the server during 10 minutes.

(a) No user activity is loaded on mobile devices: the prototyped controller decides feature extraction in the mobile computers. The amount of transferred data, TX and RX, is measured using `iftop` [60] in the server, getting 34.5 kB (transfer rate 0.46 kbit/s), which corresponds to the features and the additional information required for SSH communication.

(b) VLC is loaded on each mobile computer, during the 10 minutes of the experiment: the prototyped controller decides feature extraction on the server. In this case, the transferred data raises to 1.40 MB (transfer rate 18.67 Kbit/s), which corresponds to the traffic headers and additional information required for SSH communication.

(c) At the beginning of each minute in the experiment, VLC is loaded on each mobile computer during one minute with a probability of 0.25, 0.5 and 0.75. As expected, intermediate values are observed for the amount of transferred data.

Figure 9 depicts the results obtained.

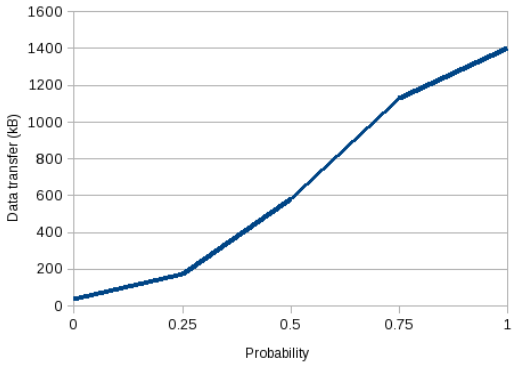


Figure 9: Server computer: data received during 10 minutes, for different levels of processor load in mobile devices

As has been shown, the proposed framework helps to reduce the occupied bandwidth in this case, by using the free processing capabilities available in mobile devices during certain time intervals.

The same experiment has been run with other metrics. For example, the processor load is expected to decrease at the server side when the feature extraction process is hosted on mobile computers. However, due to the multicore design of the computer used as server, this parameter has not been significantly impacted.

5. Conclusion and future work

The major contribution of this work is the design of a novel framework that allows convenient distribution of intrusion detection tasks taking into account security requirements, variable availability of computing resources in personal and enterprise computers, and additional capabilities coming from cloud services. In addition, the proposal integrates IDS projects built on diverse technologies and approaches, allowing modular re-use of established IDS techniques. As a derived result, the framework avoids a single point of failure or attack, by supporting multiple instances of the different tasks required for the overall IDS.

The experiments show the feasibility of the approach, and provide insight into future work. The framework itself can evolve in different directions, listed below.

From the scheduling point of view, two main problems have been identified, which can be explored further. First, considerable effort must be spent in order to adapt and test existing techniques for flow scheduling on the proposed architecture; well-known and novel methods, algorithms and heuristics should be taken into account. Second, additional research is required to integrate existing predictive models, taking the most of their capabilities in order to increase the effectiveness of the scheduler component.

Another future work line has to do with the way in which optimum framework parameters are established: in addition to simulation aided estimation, adaptive behaviour could be added to the proposed framework, incorporating the results from advanced modelling techniques using, for example, neural networks.

System resilience is another interesting issue to work on. The framework supports a fallback policy, to be activated when central control is not available or accessible. This could be expanded by adding a service discovery mechanism, allowing devices to autonomously take suboptimal decisions based on local information coming from neighbour computers.

Finally, the experimental design should be completed, integrating other relevant factors, such as main memory usage, storage requirements and energy consumption.

Acknowledgement

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness (MINECO/FEDER) under the granted Project SEQUOIA-UA (Management requirements and methodology for Big Data analytics) TIN2015-63502-C3-3-R, by the University of Alicante, within the program of support for research, under project GRE14-10, and by the *Conselleria de Educación, Inves-*

tigación, Cultura y Deporte, *Comunidad Valenciana*, Spain, within the programs of support for research, under project GV/2016/087 and AICO/2017/134.

References

[1] R. Brewer, Cyber threats: reducing the time to detection and response, Network Security 2015 (2015) 5–8.

[2] K. Elazari, How to survive cyberwar STEP ONE: Stop counting on others to protect you, Scientific American 312 (2015) 66–69.

[3] R. von Solms, J. van Niekerk, From information security to cyber security, Computers & Security 38 (2013) 97–102.

[4] D. E. Denning, An intrusion-detection model, IEEE Transactions on Software Engineering 13 (1987) 222–232.

[5] H. Debar, M. Dacier, A. Wespi, Towards a taxonomy of intrusion-detection systems, Computer Networks-the International Journal of Computer and Telecommunications Networking 31 (1999) 805–822.

[6] L. J. G. Villalba, A. L. S. Orozco, J. M. Vidal, Anomaly-based network intrusion detection system, IEEE Latin America Transactions 13 (2015) 850–855.

[7] R. Singh, H. Kumar, R. K. Singla, An intrusion detection system using network traffic profiling and online sequential extreme learning machine, Expert Systems with Applications 42 (2015) 8609–8624.

[8] M. Riecker, S. Biedermann, R. El Bansarkhani, M. Hollick, Lightweight energy consumption-based intrusion detection system for wireless sensor networks, International Journal of Information Security 14 (2015) 155–167.

[9] B. Shah, B. H. Trivedi, Improving performance of mobile agent based intrusion detection system, in: Fifth International Conference on Advanced Computing & Communication Technologies (ACCT), 2015, pp. 425–430.

[10] A. Derhab, A. Bouras, Multivariate correlation analysis and geometric linear similarity for real-time intrusion detection systems, Security and Communication Networks 8 (2015) 1193–1212.

[11] L. Wei-Chao, K. Shih-Wen, T. Chih-Fong, CANN: An intrusion detection system based on combining cluster centers and nearest neighbors, Knowledge-Based Systems 78 (2015) 13–21.

[12] A. S. Eesa, Z. Orman, A. M. A. Brifcani, A novel feature-selection approach based on the cuttlefish optimization algorithm for intrusion detection systems, Expert Systems with Applications 42 (2015) 2670–2679.

[13] M. A. Faisal, Z. Aung, J. R. Williams, A. Sanchez, Data-stream-based intrusion detection system for advanced metering infrastructure in smart grid: A feasibility study, IEEE Systems Journal 9 (2015) 31–44.

[14] F. L. Greitzer, A. P. Moore, D. M. Cappelli, D. H. Andrews, L. A. Carroll, T. D. Hull, Combating the insider cyber threat, IEEE Security & Privacy 6 (2008) 61–64.

[15] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, Future Generation Computer Systems-the International Journal of Grid Computing and Escience 29 (2013) 1645–1660.

[16] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Communications of the Acm 53 (2010) 50–58.

[17] S. X. Wu, W. Banzhaf, The use of computational intelligence in intrusion detection systems: A review, Applied Soft Computing 10 (2010) 1–35.

[18] F. Amiri, M. R. Yousefi, C. Lucas, A. Shakery, N. Yazdani, Mutual information-based feature selection for intrusion detection systems, Journal of Network and Computer Applications 34 (2011) 1184–1199.

[19] G. Folino, P. Sabatino, Ensemble based collaborative and distributed intrusion detection systems: A survey, Journal of Network and Computer Applications 66 (2016) 1–16.

[20] S. Peddabachigari, A. Abraham, C. Grosan, J. Thomas, Modeling intrusion detection system using hybrid intelligent systems, Journal of Network and Computer Applications 30 (2007) 114–132.

[21] J. Maestre, A. L. Sandoval, L. J. Garca, Alert correlation framework for malware detection by anomaly-based packet payload analysis, Journal of Network and Computer Applications 97 (2017) 11–22.

[22] H. T. Elshoush, I. M. Osman, Alert correlation in collaborative intelligent intrusion detection systems – a survey, Applied Soft Computing 11 (2011) 4349–4365.

[23] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, K.-Y. Tung, Intrusion detection system: A comprehensive review, Journal of Network and Computer Applications 36 (2013) 16–24.

[24] W. Abbas, A. Laszka, Y. Vorobeychik, X. Koutsoukos, Scheduling intrusion detection systems in resource-bounded cyber-physical systems, in: First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy (CPS-SPC), 2015, pp. 55–66.

[25] A. Patel, M. Taghavi, K. Bakhtiyari, J. Celestino, An intrusion detection and prevention system in cloud computing: A systematic review, Journal of Network and Computer Applications 36 (2013) 25–41.

[26] B. B. Zarpelo, R. S. Miani, C. T. Kawakani, S. C. de Alvarenga, A survey of intrusion detection in Internet of Things, Journal of Network and Computer Applications 84 (2017) 25–37.

[27] A. A. Gendreau, M. Moorman, Survey of intrusion detection systems towards an end to end secure Internet of Things, in: IEEE 4th International

Conference on Future Internet of Things and Cloud (FiCloud), 2016, pp. 84–90.

[28] Z. Yan, P. Zhang, A. V. Vasilakos, A survey on trust management for Internet of Things, Journal of Network and Computer Applications 42 (2014) 120–134.

[29] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, M. Rajarajan, A survey of intrusion detection techniques in Cloud, Journal of Network and Computer Applications 36 (2013) 42–57.

[30] A. Sajid, H. Abbas, K. Saleem, Cloud-assisted IoT-based SCADA systems security: A review of the state of the art and future challenges, IEEE Access 4 (2016) 1375–1384.

[31] A. Botta, W. de Donato, V. Persico, A. Pescapé, Integration of cloud computing and Internet of Things: A survey, Future Generation Computer Systems 56 (2016) 684 – 700.

[32] A. Shiravi, H. Shiravi, M. Tavallaee, A. A. Ghorbani, Toward developing a systematic approach to generate benchmark datasets for intrusion detection, Computers and Security 31 (2012) 357–374.

[33] X. Ning, C. Bin, C. Lei, H. Zi, S. Yingxia, Heterogeneous environment aware streaming graph partitioning, IEEE Transactions on Knowledge and Data Engineering 27 (2015) 1560–1572.

[34] C. Kao, W. Hsu, An adaptive heterogeneous runtime framework for irregular applications, Journal of Signal Processing Systems for Signal Image and Video Technology 80 (2015) 245–259.

[35] H. Mora, J. F. Colom, D. Gil, A. Jimeno-Morenilla, Distributed computational model for shared processing on Cyber-Physical System environments, Computer Communications 111 (2017) 68–83.

[36] J. F. Colom, H. Mora, D. Gil, M. T. Signes-Pont, Collaborative building of behavioural models based on internet of things, Computers & Electrical Engineering 58 (2017) 385–396.

[37] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, S. Pastrana, Power-aware anomaly detection in smartphones: An analysis of on-platform versus externalized operation, Pervasive and Mobile Computing 18 (2015) 137–151.

[38] A. Uthra Rajan, S. V. Kasmir Raja, A. Jeyasekar, A. J. Lattanze, Energy-efficient predictive congestion control for wireless sensor networks, IET Wireless Sensor Systems 5 (2015) 115–123.

[39] Z. Wang, X. Wang, X. Jin, T. Wang, Y. Luo, MBalancer: predictive dynamic memory balancing for virtual machines, Journal of Software 25 (2014) 2206–2219.

[40] A. Bashar, Autonomic scaling of cloud computing resources using bn-based prediction models, in: IEEE 2nd International Conference on Cloud Networking, 2013, pp. 200–204.

[41] K. Young-Hwan, R. Konow, D. Dujovne, T. Turletti, W. Dabbous, G. Navarro, PcapWT: An efficient packet extraction tool for large volume network traces, Computer Networks 79 (2015) 91–102.

[42] B. Langthasa, B. Acharya, S. Sarmah, Classification of network traffic in LAN, in: International Conference on Electronic Design, Computer Networks & Automated Verification, 2015, pp. 92–99.

[43] D. S. Punithavathani, K. Sujatha, J. M. Jain, Surveillance of anomaly and misuse in critical networks to counter insider threats using computational intelligence, Cluster Computing-the Journal of Networks Software Tools and Applications 18 (2015) 435–451.

[44] T. Ishitaki, D. Elmazi, L. Yi, T. Oda, L. Barolli, K. Uchida, Application of neural networks for intrusion detection in Tor networks, in: IEEE

29th International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2015, pp. 67–72.

[45] V. Chandola, E. Eilertson, L. Ertoz, G. Simon, V. Kumar, MINDS: Architecture & design, in: Data Warehousing and Data Mining Techniques for Cyber Security, volume 31 of *Advances in Information Security*, Springer US, 2007, pp. 83–107.

[46] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2014. URL: `http://www.R-project.org`.

[47] R. Jasek, A. Szmit, M. Szmit, Usage of modern exponential-smoothing models in network traffic modelling, in: I. Zelinka, G. Chen, O. E. Rssler, V. Snasel, A. Abraham (Eds.), Nostradamus 2013: Prediction, Modeling and Analysis of Complex Systems, volume 210 of *Advances in Intelligent Systems and Computing*, Springer International Publishing, 2013, pp. 435–444.

[48] S. Dharmapurikar, J. W. Lockwood, Fast and scalable pattern matching for network intrusion detection systems, IEEE Journal on Selected Areas in Communications 24 (2006) 1781–1792.

[49] W. Bul'ajoul, A. James, M. Pannu, Improving network intrusion detection system performance through quality of service configuration and parallel technology, Journal of Computer and System Sciences 81 (2015) 981–999.

[50] Y. Jiaxue, G. Yu, B. Yubin, Y. Ge, Scalable complex event processing on top of MapReduce, in: Proceedings of the 14th Asia-Pacific Web Conference (APWeb), 2012, pp. 529–536.

[51] Y. Shun-Fa, C. Wei-Yu, W. Yao-Tsung, ICAS: An inter-vm IDS log cloud analysis system, in: 2011 IEEE International Conference on Cloud Computing and Intelligence Systems, 2011, pp. 285–289.

[52] M. Kumar, M. Hanumanthappa, Scalable intrusion detection systems log analysis using cloud computing infrastructure, in: IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), 2013, pp. 206–209.

[53] B. Smith, Bash, the bourne again shell, Byte 17 (1992) 299–299.

[54] T. Ylonen, SSH - secure login connections over the internet, Proceedings of the Sixth Annual Usenix Security Symposium: Focusing on Applications of Cryptography (1996) 37–42.

[55] C. Edwards, Not-so-humble Raspberry Pi gets big ideas, Engineering & Technology 8 (2013) 30–33.

[56] N. Kurd, P. Mosalikanti, M. Neidengard, J. Douglas, R. Kumar, Next generation Intel (R) Core (TM) micro-architecture (nehalem) clocking, IEEE Journal of Solid-State Circuits 44 (2009) 1121–1129.

[57] T. Hubbard, R. Lencevicius, E. Metz, G. Raghavan, Performance validation on multicore mobile devices, Verified Software: Theories, Tools, Experiments 4171 (2008) 413–421.

[58] S. Neeli, K. Govindasamy, B. M. Wilamowski, A. Malinowski, Automated data mining from web servers using perl script, in: 12th International Conference on Intelligent Engineering Systems, 2008, pp. 191–196.

[59] T. Basset, Coordination and social structures in an open source project: VideoLAN, Free/Open Source Software Development (2005) 125–151.

[60] C. Binnie, Real-Time Network Statistics with Iftop, Apress, Berkeley, CA, 2016, pp. 1–12.

José Francisco Colom received the B.S. degree in Computer Science Engineering in University of Alicante, Spain, in 1998. He received the PhD degree in computer science from the University of Alicante in 2016. Currently he works as a computer teacher in vocational education, and collaborates in teaching and research activities in the faculty of the Department of Computer Science Technology at the same university. His current areas of research interest include distributed systems, data mining and computer security.

David Gil is an associated professor at the Department of Computing Technology and Data Processing at the University of Alicante, Spain. David received a Ph.D. in Computer Science from the University of Alicante (Spain) in 2008. His research interests include Applications of Artificial Intelligence, data mining, data warehouses, multidimensional databases, OLAP, design with UML, and MDA. He has published papers in high quality international conferences and highly cited international journals, and he is also a reviewer. Dr. Gil has been involved in the organization of several internationals workshops and has served as a Program Committee member of several conferences and workshops.

Higinio Mora received the BS degree in computer science engineering and the BS degree in business studies in University of Alicante, Spain, in 1996 and 1997, respectively. He received the PhD degree in computer science from the University of Alicante in 2003. Since 2002, he is a member of the faculty of the Computer Technology and Computation Department at the same university where he is currently an associate professor and researcher of Specialized Processors Architecture Laboratory. His areas of research interest include computer modelling, computer architectures, high performance computing, embedded systems, internet of things and cloud computing paradigm. He has participated in many conferences and most of his work has been published in international journals and conferences, with more than 50 published papers.

Bruno Volckaert is professor advanced programming and software engineering in the Department of Information Technology (INTEC) at Ghent University and senior researcher at IMEC. He obtained his Master of Computer Science degree in 2001 from Ghent University, after which he worked on his PhD at Ghent University on data intensive scheduling and service management for Grid computing, which he obtained in 2006. He has been involved in the inception and execution of over 30 national and European research projects dealing with reliable distributed applications and was coordinator of 3 large national interdisciplinary research tracks budgeted at 8.8 M€. He has over 85 publications in conferences and journals. His current research deals with reliable and high performance distributed software systems for City-of-Things (IoT for Smart Cities, with the creation of a living, city-wide lab in the city of Antwerpen-Belgium), automated decision support systems for UAVs, intelligent transportation applications and autonomous resource optimization of cloud-based applications.

· Novel framework for scheduling intrusion detection tasks in IoT

· Flexible integration of cloud computing and mobile computing resources

· Architecture for deployment of state-of-art methods and techniques

· System resilience achieved by allowing multiple task instances in different devices

· Experimental results show resource utilization and performance benefits