# A Novel Ad-Hoc Mobile Edge Cloud Offering Security Services through Intelligent Resource-Aware Offloading

Toufic Dbouk, Azzam Mourad, Hadi Otrok, Hanine Tout and Chamseddine Talhi

***Abstract*—While the usage of smart devices is increasing, security attacks and malware affecting such terminals are briskly evolving as well. Mobile security suites exist to defend devices against malware and other intrusions. However, they require extensive resources not continuously available on mobile terminals, hence affecting their relevance, efficiency and sustainability. In this paper, we address the aforementioned problem while taking into account the devices limited resources such as energy and CPU usage as well as the mobile connectivity and latency. In this context, we propose an ad-hoc mobile edge cloud that takes advantage of Wi-Fi Direct as means of achieving connectivity, sharing resources, and integrating security services among nearby mobile devices. The proposed scheme embeds a multi-objective resource-aware optimization model and genetic-based solution that provide smart offloading decision based on dynamic profiling of contextual and statistical data from the ad-hoc mobile edge cloud devices. The carried experiments illustrate the relevance and efficiency of exchanging security services while maintaining their sustainability with or without the availability of Internet connection. Moreover, the results provide optimal offloading decision and distribution of security services while significantly reducing energy consumption, execution time, and number of selected computational nodes without sacrificing security.**

***Index Terms*—Smart Offloading, Resource-Aware Service Offloading, Ad-Hoc Mobile Edge Cloud, Edge Computing, Cloud Computing, Mobile Computing, Sustainable Security Services, Security-as-a-Service, Intrusion Detection.**

## I. Introduction

SMART devices are being heavily used in today's life especially with the world's convergence to E-commerce and cloud-based computing. They are also being used as major and primary elements in mobile phone sensing networks to provide a wider coverage area, easier deployment, strengthened features and a social aspect [1], [2]. Since smart-devices are based on architectures that are quite similar to other computational devices such as desktops and laptops, they are subject to similar threats. The authors of [3] and [4] classify smart-phone malware into various classes such as *Virus, Worm, Scareware, Spyware, Torjan, Backdoor, Ransomware, Botnet*and, *Rootkit*. They also state the infection vectors by which smart-phone malware is capable of delivering its content such as *SMS/MMS, Bluetooth, Internet,* and, *File duplication via USB*. In addition, the work in [3] categorizes the malicious activities of smart-phone malware into *System Damage, privacy Steal, Fee Consume, Denial of Service,* and, *Remote Control*. Further, the authors of [5] manage to state some of the ways by which malware can trivially disguise and bypass its detection such as *Repackaging, Disassembling and Reassembling, Identifier Renaming, Data Encoding,* and *Junk Code Insertion*. Therefore, as stated by [6], smart-devices require special high level protection such as intrusion detection systems against such malwares, while taking into consideration the device's limited resources such as battery and memory.

However, running security services on mobile devices impose several problems such as high consumption of memory, CPU, and energy. Moreover, their performance and sustainability are hindered by the low specifications of the device's hardware. It is worth to mention that the ideas proposed in this article target security-as-a-service in general yet the focus of this study is intrusion detection systems (IDS) due to the complexity that can impose on resource constrained mobile devices. In some cases the IDS mobile agent can result in exhaustiveness and unresponsiveness of the device which negatively impacts its usability. Several approaches have been proposed to ameliorate the performance of IDS on smart-devices by offloading the analysis and scanning to off-device infrastructures such as clouds and remote servers [7], [8]. They take advantage of powerful cloud-based infrastructure to perform the detection while keeping a mobile agent to provide minimal security and communication with servers. Similarly, other approaches [9]–[11] have addressed the problem of intrusion detection in ad-hoc networks by preventing and detecting routing attacks in addition to identifying misbehaving or intruding nodes.

In this context, cloud computing consists of using distributed remote servers over the Internet to store, search, acquire, and process data instead of using local servers [12]. Such computing allows access to shared and easily provisioned computing resources in an on-demand fashion. Offloading, a technique used to migrate or offload intensive computations from a device to an external platform such as clouds, has received much attention to address resource limitation on smart devices [13]–[18]. Server-based cloud [8] consists of Internet based remote servers that provide services for smart-devices to take advantage of or to perform a specific tasks. Such clouds benefit from high computational power and resources of remote servers to augment the performance of smart devices. In other words, services on the remote servers perform most, if not all, of the high intensive tasks on behalf of the mobile device to reduce the computational load on the device itself. Therefore, server-based cloud computing serves as a solution to overcome the high computational demand of running security services on smart-devices. However, these approaches suffer from various limitations and drawbacks summarized as follows:

1) Additional cost charges due to extra quota usage, especially for mobile data, and server's subscriptions.
2) Overloading the current mobile data infrastructure by additional usage of existing communication technologies. This problem will expand further with the tremendous increase reaching around 50 Billion of connected devices by 2020 [19].

T. Dbouk and A. Mourad are with the Department of Computer Science and Mathematics, Lebanese American University (LAU), Beirut, LB.
E-mail: toufic.dbouk@lau.edu.lb, azzam.mourad@lau.edu.lb
H. Otrok is with the Department of Electrical and Computer Engineering, Khalifa University, Abu Dhabi, UAE.
E-mail: hadi.otrok@kustar.ac.ae
H. Tout and C. Talhi are with the Department of Software Engineering and IT, École de Technologie Superieure, Montreal, CA.
E-mail: hanine.tout.1@ens.etsmtl.ca, chamseddine.talhi@etsmtl.ca

3) Sketchy Internet connection and being bound by its availability.

More recently, mobile edge computing has been an area of interest for many researchers who proposed offloading computations to close mobile edge computing server MEC [20]–[24]. However these approaches suffer from the following drawbacks:

1) Require an Internet connection and a pre-set infrastructure to offload similar to cloud-based techniques.
2) Although these works reduce the energy and execution time on the devices, they impose an additional burden on the wireless networks and can lead to channel interference with other devices in the vicinity.

This article aims to address all aforementioned limitations by introducing a solution based on ad-hoc mobile edge model [25], which can be created by the efforts of nearby volunteering devices to form a cloud environment, allowing resources and computation sharing. An ad-hoc edge cloud is composed of a cluster of mobile devices known as nodes. Data and computation are offloaded from one node (device) to another for execution. This ad-hoc mobile edge cloud benefits from an intrinsic characteristic that enable it to be created anywhere and anytime due to the availability of volunteering devices, thus allowing data and computational offloading without requiring additional infrastructure. Therefore, it positively contributes in reducing the cost needed for the setup due to its spontaneous creation. Processing tasks collaboratively in mobile ad-hoc cloud has proved to be efficient in many scenarios. One of these is crowd computing, where video recordings from multiple mobile devices can be gathered to create a video that covers an entire event from different perspectives. Similar processing applies on many other situations where group of mobile devices are involved in particular activity. Moreover, Wi-Fi Direct, a technology that allows devices to connect directly to each other without an internet connection, is being used in several areas such as gaming, social media, and medical fields [26]–[28]. In similar contexts, security services can benefit from such settings, yet other aspects should be taken into consideration. Performance degradation is one of these aspects which is likely to arise when running security services on such resource constrained hardware. For instance, a security service like intrusion detection could be switched off when a phone call is ongoing not to affect the call quality.

To address the aforementioned issues, we elaborate in this paper a novel ad-hoc mobile edge based cloud that provides sustainable security-as-a-service through intelligent and efficient multi-layer computation offloading. To the best of our knowledge, this is the first work that propose ad-hoc mobile edge cloud and Wi-Fi Direct to provide security-as-a-service for mobile devices. In this work security-as-a-service is a security system, offered to a mobile device, with the ability to run it partially/totally outside the end terminal. The client is the mobile device in this case requesting the intrusion detection service and the provider is the mobile ad-hoc cloud (i.e., participating devices in the mobile ad-hoc cloud). The security service runs on a mobile device, yet in case of lack of resources on the latter, the service is offered by nearby devices participating in an ad-hoc mobile cloud, through an intelligent mechanism based on offloading techniques. The proposed method takes into consideration all of the requester status and the mobile ad-hoc cloud resources to distribute the chunks for intrusion detection over participating devices and provide the results back to the requester. A smart offloading decision supported with dynamic profiler for data collection is generated. The decision guarantees offering the needed security service (e.g., intrusion detection) by considering both the resources of the offloading node and the available nodes in the cluster. In other words, the proposed solution guarantees finishing the execution of the security service whether running on the device or in collaboration between devices within the formed mobile ad-hoc cloud while taking into consideration the resources of the requester device and those of the formed cloud. The proposed approach takes advantage of Wi-Fi Direct protocols to establish an ad-hoc mobile edge cloud that serves as a cluster of nodes to monitor and provide the security service needed on devices. For intrusion detection, each node runs an IDS that is used to detect malicious executions.

Particularly, a *Master Node* accepts an offloading request from a *Requester Node* and assigns tasks to each available *Serving Node* in an intelligent manner, which is able to reduce energy and execution time on the *Requester Node* without sacrificing security. In addition, multiple detection engines can be used to achieve a better detection rate. Each node can run a different intrusion algorithm by which data is analyzed. The framework embeds a set of interconnected modules: *Ad hoc Mobile Cloud Manager, Profiler, Offloading Manager, Communication Manager, Intrusion Detection Engine, Offloading and Distribution Controller, and Intelligent Offloading Distributor*. These modules seamlessly work together in order to analyze a given intrusion by establishing a cluster of devices that decides whether performing the detection should be locally on the device or offloaded to surrogate devices. In case the framework decides to offload, it additionally outputs the best possible node assignments to attain an optimal distribution in order to augment the device's performance. The proposed *Intelligent Offloading Distribution* approach is able to reduce the energy consumption on the device by 92%, execution time by 80% and number of selected nodes by 70% on average. The main contributions of this paper are four folds:

- Proposing a novel ad-hoc mobile edge cloud offering security-as-a-service, which uses Wi-Fi Direct as means of communication. To the best of our knowledge, none of the current approaches provide security-as-a-service over ad-hoc mobile edge cloud.
- Elaborating a multi-objective resource-aware optimization model and genetic-based solution for efficient offloading decision and distribution. The proposed Intelligent Offloading module, with the support of dynamic profiling of contextual and statistical data from the edge mobile devices, generate optimal offloading decision and distribution in order to reduce CPU and power consumption on devices, hence augmenting the sustainability and performance of the security services by reducing its resource consumption and execution time.
- Providing real-time sustainable security services in the absence of an Internet connection without levying any additional charges related to data consumption and/or subscribing to servers.
- Unburdening the mobile network infrastructure due to offloading to mobile ad-hoc edge using Wi-Fi Direct instead of mobile data.

The rest of the paper is organized as follows. Related work is summarized in section II. Section III shows the problem illustration. Section IV introduces the proposed framework. Our optimization model and heuristic algorithm are described in section V, VI, and VII respectively. The implementation and experiments are presented in section VIII. Finally, we conclude the paper in section IX and point out some future research directions.

## II. Related Work

In this section we review existing literature related to our work. Several approaches have been addressed and presented in this context to improve the performance of IDS on mobile devices which are divided into two categories: cloud-based IDS and intrusion detection in ad-hoc networks. We also present works that leverage mobile edge computing for computations offloading. Further we present the literature view showing the scope of current Wi-Fi Direct applications which is a core component in our framework.

## *A. Cloud-Based Intrusion Detection*

The authors in [29] surveyed cloud-based intrusion detection systems and highlighted various issues to its implementation. While cloud-based intrusion detection mechanisms are able to reduce the bandwidth usage, processing power needs and power consumption on mobile devices, the process of transferring data from smartphones to the cloud goes through network communication devices which as the authors emphasized, form potential security breaches where attackers can capture critical information.

[7] proposed a mechanism by which a device is replicated in the cloud and examined for malicious behaviors. Their mechanism allows multiple detection engines to run in parallel in order to gain a better detection rate by running multiple emulators of the same virtualized device. In case of failures, the mobile agent is responsible for putting the smart-device in a recovery mode synchronized with the cloud. The cloud service is responsible for the analysis and detection of suspicious behaviors.

[30] proposed an architecture for moving the IDS computation to remote servers that contain exact copies of the devices. Their technique allows for multiple detection algorithms to run simultaneously on replicated devices in the virtual environment. A minimal set of traces allowing solid replication of the device are sent by the mobile agent to the remote server via recording and replaying framework.

[8] proposed a framework that caters a real-time and strong off-device protection by continuously synchronizing the device with an emulated device on the cloud using two main components. The first component is a client agent continuously passes device's inputs and communications data to be examined on the cloud and takes actions against threats. The second component is the cloud itself that hosts emulated devices, receives events from different interfaces of the device, and ensures a periodic backup of the emulate device.

[31] introduced a hybrid IDS for smart-devices that automatically decides whether the detection should take place locally on the device or be offloaded to the cloud for analysis. However continuous communication between the mobile agent and the cloud is required. Diverse detection engines are integrated in the cloud to gain protection against a larger set of malwares. The proposed framework consists of several components most of which run on the device itself. In addition, it decides on the level, security and type of detection algorithms to be enforced on the device.

However, the above approaches impose extra power consumption when used with mobile data instead of Wi-Fi to send traces, input events, and data. In addition, mirroring the traffic and data to the cloud via mobile data overloads the mobile infrastructure and levies extra charges on the user as it consumes his mobile data quota. It also requires the usage of remote servers, that are not provided for free, which inflicts additional costs. Moreover, such approaches suffer from the intrinsic latency drawback present in current Wi-Fi technologies especially in long distance communications [32]. To the best of our knowledge, none of the proposed approaches consider using ad-hoc cloud for security services.

An alternative hybrid approach has been introduced in [25] by which a new elastic computing platform for smart-device is used. It is based on combining both ad-hoc virtual cloud and infrastructure based cloud to achieve higher scalability. An ad-hoc virtual cloud, composed of smart-devices in close proximity connected via wireless radio such as Bluetooth, work cooperatively to accomplish offloading tasks. An infrastructure cloud composed of phones performs computing intensive tasks. Nonetheless, their paper focuses more on the server-based cloud aspect and just introduce the virtual ad-hoc cloud, without illustrating or providing information pertaining to the offloading and cooperation aspects. The authors plainly state that an ad-hoc virtual cloud can be used for offloading without portraying further technical details. The paper also doesn't exhibit any experiments investigating the performance and impact of cooperative execution using the cluster of smart-devices.

## *B. Intrusion Detection in Ad-hoc Networks*

In another context, the literature review also shows a different approach to guarantee a sort of protection in ad-hoc networks.

The author in [33] compared different IDS approaches for mobile ad-hoc network; one of which implements an IDS agent on every node, while the others implement one on every clusterhead significantly reducing the energy consumption. Although such approach might overload the clusterheads, but this could be addressed with the correct load distribution which will be capable of extending the node's lifetime.

The authors of [9] proposed a zone based IDS to prevent routing attacks such impersonating and black/gray attacks. Every node in each different zone has its own local IDS. Nodes communicate together to exchange intelligence about intrusion nodes. This corporation allows nodes to have a better detection rate and raise more specific alarms in case of any intrusion.

The authors [10] of proposed an approach for dynamic intrusion detection in MANETs by which they were able to increase the detection rate and reduce the amount of false positive. Their approach is based on Genetic algorithm and artificial immune system and is capable of adapting to changes in the network topology.

In [34], the authors introduce a new statistical approach for intrusion detection in ad-hoc network by tracking node's route selection rate. Their anomaly-based intrusion detection system is implemented in the cluster head, is capable of detecting and isolating malicious nodes, and eliminating nodes with high load.

The authors of [35] present a mechanism for intrusion detection in mobile ad-hoc networks via relying on a master cluster head. A cluster head is elected in each cluster that monitors the nodes behaviour within its cluster. A master cluster head is elected from the cluster heads in order to handle intra cluster detection and elimination of malicious nodes based on the data collected from the cluster heads.

[11] propose a Dempster-Shafer based model to detect misbehaving vehicles in two phases. Nodes are motivated to cooperatively behave during the formation of the cluster based on their reputation. Misbehaving nodes are detected after cluster formation based on a cooperative watchdog model. Their approach is capable of achieving an increase in the detection rate, a decrease in false negatives rate, and a reduction in the number of selfish nodes, while at the same time, maintaining Quality of Service.

However, such approaches differ from our work as they tackle intrusion detection systems focused on routing protocols and detection of misbehaving nodes. They allow the detection of malicious nodes by analyzing their behaviors and actions. In our work, we focus on intrusion detection systems monitoring the system activities within a node.

### C. Computation Offloading in Mobile Edge Cloud Computing

In different context, offloading computations to an edge server in a mobile edge cloud computing infrastructure, is being adopted. In [20] the authors propose an opportunistic computation offloading system in which data mining tasks are efficiently executed locally or in a MEC to reduce execution time and energy consumption offering an effective load balancing mechanism between mobile devices and servers. The proposed system relies on the amount on unprocessed data, local resources, and contextual information to determine the optimal execution mode and the number of connected edge servers to be used for collaborative analytics of data mining algorithm. Furthermore, in case collaborative offloading is not possible due to the unavailability of mobile devices and edge servers, the offloading system falls back to server-based execution.

The authors of [22] propose an offloading priority function for resource allocation in a multi-user TDMA MECO system by which the Base Station decides whether an edge node should perform complete or minimum offloading taking into consideration the energy consumption under computational latency constraints. A convex optimization problem minimizing the weighted sum of mobile energy consumption is formulated to provide the offloading decision in Infinite and Finite MEC by setting priorities for users depending on channel gains and energy consumption in the form of a threshold policy structure. The paper assumes that the Base Station has a knowledge of channel gain and local energy consumption for input for all users.

An Energy-efficient computation offloading (EECO) scheme for 5G Heterogeneous MEC by which the energy consumption of the offloading system is improved while meeting tasks imposed latencies is illustrated in [23]. The authors formulate an optimization problem that minimizes the energy consumption of the compute-intensive task as well as that of the communication process. Their algorithm aims to decide whether the device should execute the task locally or offload to the edge cloud while optimizing the radio resource allocation in the network. The EECO scheme starts by classifying mobile devices based on their task execution cost. Then, it determines device priorities to be used for allocating radio resources. Finally, each offloading device is allocated channels from the base stations based on the above criteria.

The authors of [24] describe a game theoretic approach that allows a mobile device to determine, in a distributed fashion, an efficient MEC offloading decision and select the best communication channel to reduce interference that could cause an overhead in execution time. The authors also propose a distributed computation offloading algorithm that achieves highly effective performance and overhead metrics compared to a centralized approach by solving the game using a Nash equilibrium.

However, offloading computations to a MEC server require an Internet connection and a pre-set infrastructure to offload. Additionally, even though these approaches are able to reduce energy and execution time on the device/system, they impose an extra burden on the wireless networks and increase the chance of channel interference with other nearby devices.

### D. Wi-Fi Direct

In this section, we illustrate the usability of Wi-Fi Direct as a mean of communication between mobile devices in different fields. SuperBeam is an application that uses Wi-Fi Direct to enable fast, easy, and reliable file transfer between two devices [28]. BombSquad is another application that uses Wi-Fi Direct to allow users to create multiplayer games anywhere and anytime [36]. Spaceteam, an action multiplayer game, allows up to 8 players to join at the same time using Wi-Fi Direct [37]. FireChat, a social media application aimed mainly towards sending messages and photos, is built on Wi-Fi Direct and Bluetooth to provide its features. Furthermore, the authors of [26] suggest using Wi-Fi Direct as means of real-time communication and data exchange between medical applications in order to mitigate conflicts that rise from wireless interference of access points with Wi-Fi systems. They also perform experiments in the medical field using a test bed and concludes that Wi-Fi Direct results in a higher data rate by almost 65% on average. The authors of [27] propose a new framework using Wi-Fi Direct for data exchange between users of close proximity in social networks. Their approach circumvents the growing traffic of mobile data and avoids overloading the network infrastructure and ISP's.

Indeed such approaches take advantage of Wi-Fi Direct's high data rate due to less wireless interference, fast and easy setup, and availability on new devices, but none proposes or suggests using Wi-Fi Direct as a mean of communication in an ad-hoc edge cloud for intrusion detection systems. Their scope is limited to multi-player games, file sharing applications, advertising, social networks, and real time data transfer applications.

## III. Problem Illustration

Although smart-devices are getting more and more powerful, they are still impeded by their limited resources. Battery, memory, and computing power are such resources that hold back the mobile device from achieving a higher computing efficiency. In order to overcome these limitations, we present an intelligent ad-hoc offloading framework that takes advantage of nearby smart devices to augment the performance of the devices in running intensive computing tasks.

In this paper, we focus on intrusion detection as a case study for our proposed framework since it is a computational task hindered by the aforementioned constraints. We elucidate the problem of resource demanding intrusion detection by varying the data size to be scanned and the status of each device (idle, moderate, critical). All of which affect both the performance of the detection and the device itself. We focus on reducing the execution time and energy consumption on the device while taking the mobile ad-hoc cloud's structure into consideration. Table I describes the different scenarios used to achieve a real-life description of the problem.

TABLE I: Scenarios

| Scenario | Description |
|---|---|
| $Idle$ | Device is not running any user specific applications. |
| $Moderate$ | Device is running user specific applications (downloading files, gps, Bluetooth transfer...) increasing the CPU usage to $45-50\%$ |
| $Critical$ | Device is in a Moderate scenario and running additional high computational tasks overloading the CPU usage to $\approx 98\%$ |

TABLE II: Variation of Execution Time and Energy Consumption as a factor of Input Size and Node's Status.

| Input Size / Metric | Execution Time (S) | Energy Consumption (J) |
|---|---|---|
| $100KB$ | 3.9 | 3 |
| $1MB-Idle$ | 28 | 25 |
| $1MB-Moderate$ | 49.3 | 25 |
| $1MB-Critical$ | 127 | 25 |
| $10MB$ | 459 | 212 |

We conducted some experiments based on system calls. The input files consist of a collection of system traces that are generated for several input sizes. Larger files indicate a greater collection of real-life system calls. The execution time and energy consumption are captured on the device for the 3 different file sizes (100KB, 1MB and 10MB). In addition, for the 1MB file, the experiments are done while the device is under 3 different conditions (Idle, Moderate and Critical). The results are presented in Table II. The latter shows the drastic increase in execution time of the intrusion detection process accompanied with the increase in input size. Operating on a 100KB file, the detection process took 3.9 seconds to complete while it took 28 seconds for a 1MB file and 459 seconds for 10MB. A more significant observation is the radical increase in execution time associated with the device's status at the moment of detection. Table II illustrates this observation as well. For a 1MB file, it took 28 seconds to finish execution when the device is idle, however, it took 49 and 127 seconds when the device is in a medium and critical states respectively. Moreover, we show the intensive increase in energy as the data input size grows. For a 100KB, the IDS consumed $3.9J$. This consumption tends to increase to $212J$ when the input size increases to 10MB. These results divulge the inefficiency of mobile devices to handle resource hungry tasks. On account of this issue, it is crucial to embody new methods in order to reduce the execution time and other resources within a smart-device.

In our previous work [38], we showed that Equal offloading in a mobile ad-hoc cloud reduces the load on the device performing the IDS task and decreases the execution time. However, this architecture still embodies a disadvantage of wasting extra resources by using all nodes in the cloud without taking into consideration that some nodes might be low on battery or have a high CPU usage which could hinder the offloading task. Offloading equally to all devices is not necessarily the best distribution since a better execution time might be achieved by skipping nodes with heavy CPU usage.

## IV. Approach Overview and Architecture

In this section, we present an overview of our approach and discuss the architecture of the framework illustrated in Figure 1. Our approach consists of a mobile agent that is accountable for discovering nearby devices through Wi-Fi Direct and establishing a connection between them. Hence forming a mobile ad-hoc edge cloud that allows secure communication between its devices and sharing resources to reduce the computational load on devices. Communication within the cloud is limited to data needed for decision making and executing the security service (IDS). All communication is handled and routed through the *Master Node*. Each device in the mobile ad-hoc edge cloud is considered as a node. Three different types of nodes exist: *Requester Node, Master Node, and Serving Node* constituting its main elements. Different components run on each node depending on its type. They allow a seamless offloading mechanism, a managed communication system, and an intelligent distributed intrusion detection process within the mobile ad-hoc edge cloud. Below we discuss the three main components.

- **Requesting Node** is an element of the mobile ad-hoc edge cloud that requests the execution of a task. It requests from the *Master Node* to decide whether offloading is needed to augment its performance or not. In the latter, *Requesting Node* executes the task locally. It runs the following sub-components: *Ad-Hoc Cloud Manager, Profiler, Communication Manager, and Intrusion Detection Engine*.
- **Master Node** is a node that is assigned to be the Group Owner in a specific Wi-Fi Direct Group and therefore is responsible for managing the mobile ad-hoc edge cloud and requests by nodes. It manages all connected nodes in the cluster and decides on the nodes that are most suitable to handle the offload request. Also, it is responsible for communicating data in the mobile ad-hoc edge cloud between the nodes. It runs all the sub-components consisting a normal node in addition to the following sub-component: *Intelligent Offloading Distributor and Offloading & Distribution Controller*. In this work the master node is chosen in a fixed manner representing the group owner and if the cluster already exist, the master node will be the group owner of that cluster which the requester node has joined. As previously done in several works [39], the selection of the master node can be done based on several criteria (e.g., Trust, QoS, energy level) yet it is out of the scope of this article.
- **Serving Node** is a node that the *Master Node* offloads work to. In other words, it is a surrogate node that performs services, in accordance with the *Master Node*, on behalf of a *Requesting Node*. It performs the required task and returns the result back to the *Master Node*. It runs the following sub-components: *Ad-Hoc Cloud Manager, Profiler, Communication Manager, and Intrusion Detection Engine*.

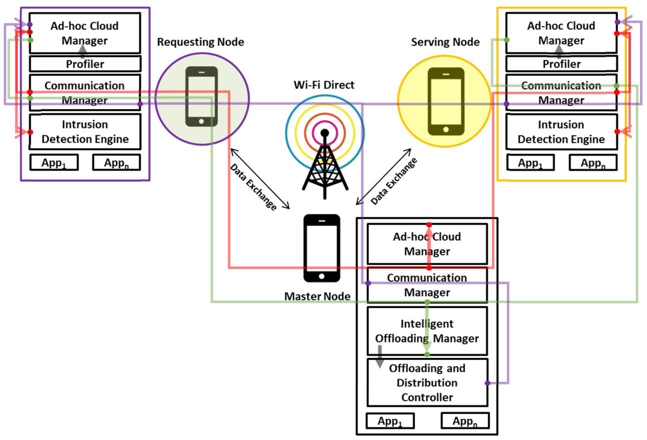


Fig. 1: Framework Architecture

Following, we list the different subcomponents and describe their interactions.

### A. Ad-Hoc Mobile Cloud Manager

The Cloud Manager acts in two different perspectives depending on the type of the node it is running on. On a *Master Node*, it is responsible for creating and monitoring the mobile ad-hoc edge cloud within a certain range. It accepts joining requests and dynamically keeps track of the number of connected nodes and their status. Furthermore, it accepts a profile of each device once it connects. The profile is updated and stored to be used later in offloading decisions. It also differentiates between the different types of nodes and assigns specific roles to each. In case of a *Requester Node* or *Serving Node*, the Cloud Manager searches for a Group Owner (GO) and requests to join its cluster. After this discovery phase, it maintains the connection with the joined cluster in order to send and receive data from the *Master Node*. Once the connection is established, the Cloud Manager sends the devices' profile (created by the *Profiler* module) to the *Master Node*. Finally, it is liable for reconnecting or disconnecting in failure scenarios.

## B. Profiler

The *profiler* is developed as a service that runs in its own process and uses native code to access the "virtual file system" and captures run-time system information on the device. It is responsible for collecting resources consumption and availability from the Kernel's information center. It collects information related to the device's resources such as CPU usage per process, memory used by each process, and battery level. It also determines the amount of free memory on the device, the name of the device, and differentiates between system and user processes. The data collected by the *profiler* is updated at a regular basis (periodically) in the mobile ad-hoc edge cloud and can be directly requested anytime by other modules in the framework (on-demand). The output of this profiler is a well structured file for easy and fast access containing all gathered information. Therefore, the *Profiler* is able to categorize the nodes in the cluster into *Idle, Moderate,* and *Critical* based on the collected information.

## C. Communication Manager

The Communication Manager is a data socket layer developed as a service to allow and facilitate the exchange of data between nodes. This layer specifies a set of protocols used to define and abstract the communication according to each node's requirement. It arbitrates two communication mechanisms, P2P and client-server, based on the type of the node and the operation to be performed by several modules of our framework. It is also responsible for sending the collected traces from the Offloading Manager module of the Requester node to the Offloading and Distribution Controller of the Master node, and then back to the Requester node.

## D. Intrusion Detection Engine

The Detection Engine is the core component for detecting any intrusion. It is in control of identifying signatures that are matched with a predefined malicious data-set. Several algorithms can be used for developing such engines such as, *Aho-Corasick Algorithm, Low Memory Keyword Trie, Wu-Manber* offered by *Snort* [40]. Genetic Algorithms and Data mining techniques are used in building such engines as well [41]. In our work, we implemented a JAVA detection engine built on top of the Aho-Corasick Algorithm to allow matching of real time gathered system calls against a data set. It constructs a string pattern finite state machine (trie) from a malicious signature database. It adds all required transitions and failure links between internal nodes of the Finite State Machine (FSM) to allow fast and efficient traversals and searching of elements. Since we are using a known database (predefined data set), the complexity of the algorithm tends to be linear in terms of input length and number of harmonized patterns. Each node, uses the constructed FSM to scan patterns in a background thread. The Detection Engine works with two types of data depending on the *Intelligent Offloading Distributor* decision. In case of a local execution, the engine scans data of the same device and acts accordingly (raise alerts, block execution etc...). However, in case of an offloading decision, the *Intrusion Detection Engine* scans data of a *Requester Node* that is provided by the *Master Node*. The Engine then communicates the result of the detection back to the *Master Node* through the *Communication Manager*. The detection engine can either be the same on all nodes or nodes can run altered versions in order to enhance the detection rate. Each engine performs a scan on the same data resulting in further and more powerful detection of malevolent data.

## E. Intelligent Offloading Distributor

The Intelligent Offloading Distributor (IOD) module is developed in Java and runs on top of a Heuristic Algorithm. The module runs on the *Master Node* and allows it to decide whether offloading the execution of a chunk augments the performance of the *Requester Node* or not. The offloading decision could either be scanning the task locally on the *Requester Node* or offloading it to a remote node for execution. In case it decides on a local execution, the *Master Node* instructs the *Requester Node* to execute the service locally and no further communication is required. On the other hand, if an offloading decision is taken, it requests the data from the *Requester Node* and reaches out to the *Communication Manager* for further handling. In order to take an intelligent offloading decision, the *IOD* first requires gathering all necessary information about the mobile ad-hoc edge cloud such as the size of the data to be offloaded, number of nodes, status of each node (idle, moderate critical), battery level, and CPU and Energy consumption on each node from the *Profiler* and other modules of the framework. If an offloading decision is taken, it then requires the *Requester Node* to send the data in order to distribute it according to the taken decision.

Moreover, our *IOD* is a multi-layered module in the sense that it outputs decisions taking into consideration both the resources of the *Requester Node* and the available resources (nodes) in the mobile ad-hoc edge cloud. To further emphasize this idea, assume a mobile ad-hoc edge cloud composed of three nodes with very high CPU usage. One node decides, based on its resources only, that offloading augments its performance. However, taking into consideration the mobile ad-hoc edge cloud resources, it is better for the node to execute its task locally rather than remotely. Such scenarios are avoided by our multi-layer *IOD* module since it initially takes into account the available nodes in the mobile ad-hoc edge cloud. The offloading decision is composed of a mapping between the set of *Serving Nodes* and the chunks to be handled by each *Serving Node*. For this mapping to be determined (where and what to offload), the module uses the NSGAII genetic algorithm to solve a multi-objective optimization problem that minimizes execution time and energy consumption on the *Requester Node*. Section V construes the optimization model in details.

## F. Offloading Distribution Controller

The Distribution Controller is a vital sub-component running on the *Master Node* only. It is developed as a service and pledged for handling the data to be distributed on other nodes and gathering the results. It receives its input from *Intelligent Offloading Distributor* and prepares it for distribution. The service runs in the background and spans one thread for each *Serving Node*. It divides the data into equal segments upon available nodes in the cluster. In some cases, where equal segmentation of the data is impossible, the last segment tends to be relatively larger than the rest. Each thread is responsible for transmitting the data, in accordance with the Communication Manager and in a guaranteed and reliable way to the node. The threads are also accountable for gathering the result from their associated nodes. The distribution process is executed in a parallel manner achieving efficient computation. In addition, the Distribution Service accepts data from spanned threads composed of the duration of the scanning process, number of malicious items, and each individual malicious item. After accepting results from all threads, the service communicates the final result back to the requester node (the node that requested the offloading process) through the Communication Manager. In case of a failure in any thread or node, the *Master Node* notifies the *Requester Node* to cover up. A *Master Node*, Group Owner, is assigned to this cluster and responsible for exchanging data between the different slaves. Moreover, the *Master Node* is in charge of the offloading decision, distribution mechanism, data transferring, and receiving the end result. In order to perform these tasks in a seamless and effective way, a service running on the *Master Node* manages the process. In addition, a service running on all nodes in the cluster coordinates with the *Master Node's* service using a set of predefined protocols. The *Master Node* receives an offloading request along with the data needed to obtain a decision. Using our *Intelligent Offloading Distributor*, the *Master Node* decides whether offloading will augment the *Requester Node's* performance or not. If an offloading decision is taken, our intelligent module selects the best distribution that will reduce energy and execution time based on a Genetic Algorithm. The framework then starts the distribution process in which the data is scattered and distributed among the *Serving Nodes* in a parallel manner. The *Master Node* waits for results from all *Serving Nodes* and informs the *Requester Node* of the final outcome. The *Requester Node* is now alarmed about any malicious data and can act upon it by removing the detected threats.

In order to generate data required by the detection engine, *strace* diagnosis utility can be used to intercept and record the system calls resulting from the interactions of applications with the Linux kernel. The execution traces are generated and saved in a log to be used by the detection engine. Due to the scope of this paper we assume that the election and movement of the *Master Node* is already established granting a well managed ad-hoc edge cloud. In addition, we assume a trust relation between the nodes allowing exchange of sensitive data.

## V. Multi-Objective Optimization for Intelligent Intrusion Detection Offloading

Though the advancements in mobile technologies, smart mobile devices are still limited in terms of CPU power, memory capacity and battery lifetime. While computation offloading in a mobile ad-hoc edge cloud releases mobile devices from extensive processing, it is indispensable to determine for each node the amount of data to be processed with respect to the latter resources and performance as well as the local node needs. We consider in this work a smart mobile device $D(C,N)$, where $C$ represents the size of data to be scanned, and $N$ represents a set of connected nearby nodes forming the ad-hoc edge cloud, such that $N = N_1, N_2, N_3, ...N_m$, where $N_j/j = 1,...m$ and $C = C_1, C_2, C_3, ...C_n$, where $C_i/i = 1,...n$ forms a chunk of computations to be processed. The aim is to scan these chunks on connected nodes in order to release resources on device D. Each Ci has its own demands with respect to execution time and energy consumption. The demands are divided into two aspects: local and remote. Local demands correspond to the execution time and energy consumption of the device $D$ itself. Remote demands correspond to the execution time and energy consumption on a remote device $N_j \neq D$ in addition to the transmission costs in terms of energy and execution time. Moreover, each of $C_i$ demands is dependent on the executing node $N_j$. The remote demands affect both Equations $F_1$ and $F_1$ corresponding to the total energy consumption and execution time respectively, which we aim to minimize. The total energy consumption is calculated based on the energy spent on processing computations locally, on transmitting data between local and remote device and by the Wi-Fi being on. As for the total execution time, beside the time needed to process data locally, the time to transmit data to remote device, receive the response back and any latency in the network are all considered in the formula. So this is how implicitly remote demands affect these equations. Moreover, each of $C_i$ demands is dependent on the executing node $N_j$. Determining the finest distribution of chunks on the different nodes leading to the best outcome is a challenging problem. Using the data collected from the Profiler module, nodes with low battery level (below threshold $t$) are excluded as performing detection tasks will put their survivability on the line. As a result, we guarantee that all surrogate nodes are capable of completing their assigned tasks. In this work, the threshold is chosen based on extensive empirical experiments, while more advanced mechanism to choose this value, is left for future work.

*Definition 1:* Given a set of chunks that need to be scanned for intrusions detection on a device $D$ which has limited resources, yet it is part of an ad-hoc network of $N$ nodes of connected mobile device, how to distribute these chunks to be scanned by participating nodes in a way to minimize the scanning time for intrusion detection, and energy consumption on device D? Knowing that each chunk $C_i$ has independently: local energy consumption $E_{c_i}^{L}$, and local execution time $T_{c_i}^{L}$, and requires different remote energy consumption $E_{c_i}^{N_j}$ and remote execution time $T_{c_i}^{N_j}$ when offloaded on nearby node $N_j$, the network is characterized by bandwidth B and Latency L, find the best distribution of these chunks such that the execution time and energy consumed on device D are minimized.

The complexity of this problem is heavily impacted by the number of possibilities a file can be divided into several chunks where each chunk is assigned to a nearby node to be scanned. This can be related to the number of ways $n$ different objects can be be distributed into $m$ different bins with $c_1$ object in first bin, $c_2$ in the second bin, etc. such that $c_1 + c_2 ... c_m = n$. In our case, the number of bins $m$ is the number of nodes in the ad-hoc mobile edge cloud since each chunk can be executed on any node.

In order to highlight more on the problem complexity, consider an ad-hoc mobile edge cloud that consists of 10 nodes where each node including the local (*Requesting Node*), is accommodating a chunk of the input data. As a result, each chunk can be executed either locally or on any of the 9 remaining nodes. Therefore, we will end up with $10^{10}$ $(10,000,000,000)$ different possible distributions. This number can significantly and radically increase as the number of nodes and chunks increases.

*Theorem 1:* Multi-objective Optimization Problem is NP-Hard.

***Proof:*** *We reduce the Multiobjective Multidimensional knapsack (MOMKP) [42] problem to our problem. The MOMKP is defined as follows: given n items $(b_1, b_2 ... b_i)$ where each one has m weights $(w_1^i, w_2^i ... w_m^i)$ and p values $(c_1^i, c_2^i ... c_p^i)$, distribute the items into different sacks such that the total p values is maximized without exceeding the m knapsack capacities. Accordingly, we construct an instance of our problem as follows:*

- *Represent each chunk to be scanned as an item in the MOMKP and setup N nodes in the mobile ad-hoc edge cloud to form the sacks.*
- *For each chunk (item) $C_i$, set its weights to be the demands in terms of execution time and energy when executed either locally or remotely on node $N_j$.*
- *For each chunk (item) $C_i$, set its values to be the objective functions and $F_1^{c_i}$ and $F_2^{c_i}$ representing the cost of the chunk in terms of energy and execution time respectively.*
- *Set the total weights of each node as $T_{time}$ representing threshold value for execution time, which the node should*

*not exceed in order to maintain good performance upon scanning assigned chunks.*

*In our problem, we choose to minimize the total cost rather than maximizing it, which is the case of MOMKP, however the problem remains essentially the same. Hence, the objective of our problem becomes to minimize the costs of chunks distributed on different nodes. A solution to it yields a solution to the multi-objective multi-dimensional Knapsack problem. Following the above reduction, an algorithm for solving our problem can be can be used to solve the MOMKP and hence our problem is NP-Hard.*

## VI. Problem Formulation

Based on the definitions given above, we formulate our problem even further by building a model that aims to minimize the energy consumption on a local device $D$ and speed up the scanning process simultaneously by reducing its execution time. Table III defines all notations used within the below model.

1) ***Minimizing Energy Consumption***
The overall energy consumption of scanning a chunk is composed of both the energy consumed locally by the scanning task and energy consumed due to offloading. The local energy is spent on CPU processing while the offloading cost is due to the energy spent on data transmission including having the Wi-Fi radio turned on.

$$F_1 = \min \sum_{N_{j=1}}^{n} x_{N_j}(E_{N_j}^{processing} + E_{N_j}^{transmission} + E_{N_j}^{idle}) \times W_E$$

2) ***Minimizing Execution Time***
The total execution time of a task is composed of its local execution time (processing) and transmission round-trip time. The time spent on transmission includes the time to send the input and receive the output, in addition to any latency in the network.

$$F_2 = \min \sum_{N_{j=1}}^{n} x_{N_j}(T_{N_j}^{processing} + T_{N_j}^{transmition}) \times W_T$$

After having declared the minimization functions, the multi-objective optimization problem becomes:

$$\begin{aligned} F &= \min\{F_1, F_2\} \\ &S.t \\ &\quad n \in \mathbf{N} \\ &\quad 0 <= x_{N_j} <= 100 \\ &\quad F_2 \leq T_{time} \\ &\quad W_E + W_T = 1 \end{aligned}$$

The first constraint ensures that the number of nodes within a cluster belongs to the set of natural numbers. The second constraint ensures that a node can be assigned any data size between 0% and 100% of the data. If $x_{N_j}$ is 0 then node $N_j$ will not be participating in the cooperation of the intrusion detection process. On the other hand, if $x_{N_j}$ is greater than 0 (say 15), then $Nj$ will be assigned 15% of the data. The third constraint ensures that the time spent on a task's execution doesn't exceed a certain threshold. The fourth constraint ensures that the assigned weights of the objective functions sums to 1. In order to make the decision adaptable with the device state and needs, each objective function can be assigned a value that represents its weight compared to the other functions. For instance, in some cases, minimizing the execution time could be of higher importance than reducing the energy consumption on the device. Therefore, a higher weight is assigned for the execution time objective function in the model indicating that minimizing the execution time is more critical and hence the distribution of chucks to be generated should answer such needs.

TABLE III: Notations

| Variable | Description |
|---|---|
| $n$ | Number of nodes |
| $x_{N_j}$ | Decision variable indicating the percentage of data to be executed by the node $j$. |
| $E_{N_j}^{processing}$ | Energy consumed locally when the task is executed by the node. |
| $E_{N_j}^{transmission}$ | Energy consumed when the task is offloaded to the node. |
| $E_{N_j}^{idle}$ | Energy consumed due to Wi-Fi being on and Wi-Fi scanning. |
| $T_{N_j}^{processing}$ | Execution time of task when executed locally. |
| $T_{N_j}^{transmission}$ | Time needed for task to be offloaded. |
| $W_E$ | Weight of the objective function minimizing energy. |
| $W_T$ | Weight of the objective function minimizing the execution time. |

## VII. Heuristic Algorithm for Intelligent Offloading Distribution

Genetic Algorithms are used to solve many search-problems in different areas by imitating the process of natural selection. Natural evolution techniques are used to generate more fitted solutions from an initial set of individuals. At first, a set of individuals forming a population are randomly selected and evaluated according to their fitness. The fitness is calculated according to a criteria that evaluates the adequacy of potential solutions. Individuals (chromosomes) with the best fitness are selected, mutated, and mated in order to generate a new population. The better fitness a solution has, the better candidate it is to reproduce offspring. Selected individuals are placed in a mating pool. Offspring is generated by performing a cross over between two parents from the mating pool. Parents could possibly undergo a mutation before the crossover in order to keep a diversity from one population to another. More fitted solutions are generated, hence providing better solutions to the problem at hand. The evolution process is terminated via several ways such as reaching a maximum number of iterations or even reaching a desired trade-off. The output of the evolution process will be a set of solutions having the best fitness, therefore being more favorable to the problem. Genetic Algorithms are being used in a wide range of applications to solve complex optimization, scheduling, and search problems by benefiting from the biological evolution process [43]–[45].

In our work, we exploit this heuristic to establish an intelligent distribution module used to solve our problem. Below, we show how we linked and related the main natural techniques of evolution to our problem. Several genetics algorithms have been proposed for solving multi-objective optimization problems. We conducted in previous work [46] experiments comparing the overhead of NSGAII, SPEA2, IBEA, MOCell, and SMSEMOA multi-objective optimization algorithms on mobile devices. The results showed that NSGAII leads to best performance, less energy and CPU consumption. In our work, we adapted NSGAII [47] that classifies solutions based on their Pareto ranking and defines proximity between classified solutions using crowding distance. The algorithm attempts to find a good trade-off between several conflicting objective functions that results in solutions that best fit the problem.

1) **Individual's Representation** We represent an Individual as an array of size $L$ where none of its cells are empty, $L$ is equal to $100/y$, and $y$ is the offloading unit. For instance, $L$ is 10 when the offloading unit is chosen to be 10% of the input data file and it is 20 when the offloading unit is chosen to be 5%. Each

**Algorithm 1** Intelligent Offloading Decision and Distribution($N$, $L$, $\mu_c$, $\mu_m$)

```
1:  Input: Population Size $N$, Individual Length $L$, Crossover Rate $\mu_c$, Mutation Rate $\mu_m$
2:  Output: Set of Fittest Individuals $S$
3:
4:  Set Initial Population Index $k := 0$
5:  Generate Random Population $P_k$ of Size $N$
6:  for $i \leftarrow 0$ to $N$ do
7:      Individual $I \leftarrow P_k[i]$
8:      Evaluate Objective Functions $F_1(I), F_2(I)$
9:  end for
10: while !Termination Criteria do
11:     Select $x$ Best Solutions from $P_k$ and add them to $P_{k+1}$
12:     Select $n$ Fittest Solutions of $P_k$ using Binary Tournament such that $n = N - x$
13:     Crossover $\mu_c$ $n$ using SBX Crossover
14:     Mutate $\mu_m$ $n$ using Polynomial Mutation
15:     for $i \leftarrow 0$ to $N$ do
16:         Individual $I \leftarrow P_{k+1}[i]$
17:         Evaluate Objective Functions $F_1(I), F_2(I)$
18:     end for
19:     $K \leftarrow K + 1$
20: end while
21: return $S$ Fittest Solution From $P_k$
```

cell contains an identifier $i$ of type integer such that $0 <= i <= n$ which identifies the node executing the chunk. Hence, each cell in the array represents a node's execution of the corresponding chunk. Finally, we find the data percentage executed by node $i$ via counting the number of occurrences of the integer $i$ in the array and multiplying it by the offloading unit $y$. For example, assuming that we have 6 nodes, which is equivalent to $L = 6$ with the following distribution: 1 2 3 4 5 1 1 5 5 6, the individual in this case will be represented as 30 10 10 10 30 10 which implies that 30% of the file will be executed by node 1, nodes $2, 3, 4, and$ 6 will each execute 10% and node 5 will execute 30% of the file. Following this representation we guarantee the distribution of the whole file and we are able to decode the distribution of remote and local execution of the nodes.

2) **Fitness Evaluation** In order to evaluate individuals, a fitness function is used in the Genetic Algorithm. This fitness is assigned to each individual, evaluated, and used as a metric to select individuals that are more fitted (have a better solution to the problem) in the population. We use in this work the previously described functions $F_1, F_2$ in order to calculate the fitness. Individual with lower fitness are better candidates to the solution since the aim is to minimize these metrics.
3) **Operators**
   - Selection: Individuals are selected from the population based on a selection operator. We use a binary tournament selection algorithm that involves selecting k (tournament size) individuals randomly and then choosing the best individuals from the tournament based on their probabilities.
   - Crossover: Selected individuals reproduce to generate new populations. We use SBX Crossover that allows to take part of parent 1 and part from parent 2 and create a new child c. For example, $p1 = 10\ 20\ 30\ 15\ 5\ 20$ and $p2 = 40\ 10\ 10\ 10\ 15\ 15$ can result in $c = 10\ 20\ 30\ 10\ 15\ 15$.
   - Mutation: Selected Individuals have a chance to get mutated (based on a probability) in order to keep a diversity in the generations. We use a mutation in which some cells' values are randomly changed indicating a change in the individual. For instance, individual A represented as 10 20 30 15 5 20 can be mutated to be 15 20 30 15 5 15.

Algorithm 1 describes the flow of our Intelligent Offloading Distributor in terms of the above described definitions. It first creates and initializes a random population composed of a list of individuals. Each individual represents a possible solution of the problem. The algorithm then performs an initial evaluation to determine the fitness of the each individual in the population by assessing the different objective functions. In the following steps, the algorithm starts to improve the overall population's fitness by adopting a survival of the fittest methodology. Individuals that survive are then added to a new population with the possibility of undergoing crossover and mutation according to defined rates ($\mu_c$ and $\mu_m$). This aims to increase the fitness of the whole population since it disregards individuals with low fitness values in favor of keeping individuals with higher fitness. The new generation of individuals is evaluated in order to determine their fitness value. This process repeats until the termination condition is met, which could be based on number of iterations, time, CPU cycles, or any relevant condition. Finally, the intelligent Offloading Distributor returns the individuals offering the best tradeoff to solve our problem.

The time complexity of this algorithm is mainly affected by the evaluation of the objective functions, population size, and termination criteria. Line number 5 has $O(1)$ complexity. Lines 6 to 9 have a complexity of $O(NL)$ where $L$ is the Individual length. Lines 10 to 18 have a complexity of $O(INL)$ where $I$ is the number of iterations in the termination condition. The return statement, line number 21, has $O(1)$ time complexity. Therefore, the total complexity sums up to $O(INL)$.

## VIII. Implementation and Experiments

In this section, we investigate the performance of our proposed model pertaining to execution time, energy consumption, and number of selected nodes. In addition, we provide a comparison of our scheme with an equal offloading approach proposed and discussed in our previous work [38].

### A. Testbed Setup

We use different types of devices running Android Operating System in our experiments; namely Samsung Galaxy S5 with the following specifications: Quad Core 2.5 GHz CPU, 2 GB RAM, 16 GB storage, and running Android OS v5.0 and LG Nexus 5 with the following specifications: Qualcomm Snapdragon 800 2.27 Ghz, 2 GB RAM, Adreno(TM) 330 @ 450 MHz, 32 GB storage, and running Android OS v5.0. We establish an ad-hoc edge cloud composed of 10 devices, using Wi-Fi P2P on Android that uses 802.11 networking standard. In our experiments, we focus on 3 aspects: performance, energy, and number of selected nodes. Performance is measured in terms of the response time of the IDS i.e. time needed for execution of the task. Energy is measured as the total energy consumed by CPU, Wi-Fi, and 3G. The number of selected nodes represents the number of nodes that are used as surrogates in the mobile ad-hoc edge cloud. We use PowerTutor [48], an android application that monitors the power consumed by the CPU, screen, and network in order to compute the energy consumed by the task at hand. In addition, we use the power profile [49] on android devices in order to get the power consumption of Wi-Fi transmission, power consumed by the CPU when it is idle, power consumed by active Wi-Fi transmission, and power consumed on screen brightness. Regarding the CPU, memory usage, and battery level on the device, we use the *Profiler* component discussed previously in section IV.

TABLE IV: Data Distribution based on the Offloading Distributor Module

| Case | Decision |
|---|---|
| $C1$ | 0 10 10 0 40 10 10 10 10 0 |
| $C2$ | 10 10 10 0 30 10 10 10 10 0 |
| $C3$ | 0 10 20 0 10 20 20 20 0 0 |
| $C4$ | 20 10 10 0 10 20 10 10 10 0 |
| $C5$ | 10 10 10 10 10 10 10 10 10 10 |
| $C6$ | 40 10 10 0 0 10 0 10 0 20 |
| $C7$ | 30 0 0 0 0 10 30 0 30 0 |

Table IV shows the distribution of the file, with the best tradeoff, in a mobile ad-hoc edge cloud of 10 nodes via our solver that implements Algorithm 1. Yet, the proposed approach is not limited to any offloading unit and still based on dynamic selection, based on the resources in the cloud. In other words, it can find any distribution of chunks that best fit the multi-objective optimization problem defined in Section VI. However, in the experiments we limit the offloading unit to be multiple of 10% to provide clear comparison for different device settings (e.g., idle, critical). However, in different experiments this value can change dynamically from one execution to another. The main objective of Table IV is to show an example of the output of the algorithm while we study the correctness of the decision in Section VIII-B. We choose the offloading unit in the experiments to be multiple of 10%. More dynamic way to choose the offloading unit will be part of future work. Since the problem at hand is a multi-objective optimization problem, our solver results in more than one solution per case. However, since every solution is not dominated by any other one, all solutions are considered equally good. Hence, we can either randomly pick a solution for each case or based on a constraint. For example, if time is a critical factor then we can pick the solution with the lowest execution time. On the other hand, assuming energy has more weight over time, we can pick a solution with the lowest energy. This trade-off is a very important feature of this algorithm, however in table IV we randomly picked the solutions. Different cases represents different scenarios of the mobile ad-hoc cloud pertaining to the status of node, battery level, and type of node. In the sequel, we highlight the main ones. Case $C1$ represents a setup of 10 nodes where the requester node #1 and surrogate node #3 & #10 are in critical state, node #4 is in idle state and the rest of nodes are a mix of idle and moderate nodes. Case $C1$ shows that the *Requesting Node* (#1 highlighted in red) is offloading all its work. The work is being offloaded to nodes number 1, 2, 4, 5, 6, 7, 8 where each node is working on $10\%, 10\%, 40\%, 10\%, 10\%, 10\%, 10\%$ respectively. Case $C7$ represents a cloud composed of 10 nodes where the requester node #1 and nodes #7, #9 are in an idle state, nodes #2, #3, #4, #5, #8, #10 are in a critical state, and node #6 is in moderate state. In case $C7$, the *Requester Node* is handling 30% of the file while the other 70% are being distributed on 3 different nodes $(5, 6, 8)$, where nodes 6 and 8 are handling 30% each and node 5 is handling the remaining 10%. Case $C5$ represents an ad hoc cloud setup composed of 10 nodes where all nodes are capable of handling a chunk to speed up the requester's node execution time. The nodes here are all in an idle state and offloading is performed equally on all nodes in parallel.

To make our experiments as realistic as possible, we allow each node to be in three different states: *Idle, Moderate, Critical*. An *Idle* node represents a node that is in standby, not performing any task other than the default running services. The CPU usage of such node is between 0 and 10%. A *Moderate* node represents a node performing some services such as GPS, networking, Bluetooth etc.. where the CPU usage doesn't exceed 55%. A *Critical* node indicates a node with a very high CPU usage, above 90%, due to running several computation-intensive applications. Therefore, we create our mobile ad-hoc edge cloud using combinations of the aforementioned scenarios. We divide the combinations into 4 different cases: *All Idle, All Moderate, All Critical, Mix*. The *All Idle* case entitles a situation where all nodes in the Mobile ad-hoc edge cloud have an Idle status. The *All Moderate* depicts a situation where all nodes in the edge cloud have a Moderate status. The *All Critical* indicates a situation where all nodes in the mobile ad-hoc edge cloud have a Critical status. Finally, the Mix scenario includes nodes of different statuses. We further divide the *Mix* scenario into multiple assignments shown in table V. In each scenario, we study 3 main situations: *Local Execution (L), Equal Offloading (E) and Intelligent Offloading Distributor (IOD)*. Local Execution depicts executing the task locally on the device. Equal Offloading represents offloading an equal percentage of data to all nodes. Hence, each node is involved in the offloading process with an equal input. Furthermore, we study 2 other scenarios as described in table VI in order to take into account extreme cases. Finally, we show the distribution generated by our Smart Solver component and compare it to the other situations. In each of these various situations we focus on 3 significant factors that affect the Requester Node and the mobile ad-hoc edge cloud itself. Particularly, we monitor the total execution time of the task, the energy consumed by the Requester Node, and the total number of selected nodes and their status. Moreover, to provide definite results regarding our approach, we perform the same set of experiments on $1MB, 10MB$, and $50MB$ sizes of input data.

TABLE V: Mix Scenario Assignments

| Assignment | Description |
|---|---|
| 7 Critical 3 Idle | The mobile ad hoc cloud consists of 7 Critical nodes and 3 Idle nodes. |
| 9 Critical 1 Idle | The mobile ad hoc cloud consists of 9 Critical nodes and 1 Idle node. |
| 5 Critical 5 Idle | The mobile ad hoc cloud consists of 5 Critical nodes and 5 Idle nodes. |

TABLE VI: Execution Scenarios

| Scenario | Description |
|---|---|
| L | Local execution on Idle Requesting Node. |
| L Critical | Local execution on Critical Requesting Node. |
| E | Equal distribution on the mobile ad hoc cloud nodes. |
| IOD | Execution using our Intelligent Offloading Distributor Component when Requester Node is in an Idle state. |
| IOD - Critical | Execution using our Intelligent Offloading Distributor Component when Requester Node is in a Critical state. |

## *B. Results and Analysis*

The following results are the average results of several runs. It is worth to mention that when the input is similar, for example for IOD and IOD2, the variation in results come from the fact that our intelligent offloading distributer outputs several results that are equally good but, with each result prioritizing different constraint energy, execution time, and/or the number of used nodes. Figure 2 shows our approach's evaluation in a scenario where all nodes of the mobile ad-hoc edge cloud are idle. We compare our Intelligent Offloading

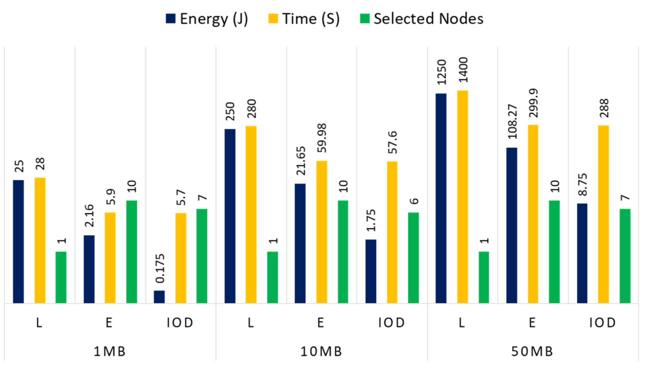


Fig. 2: Approach Evaluation Using Idle Nodes

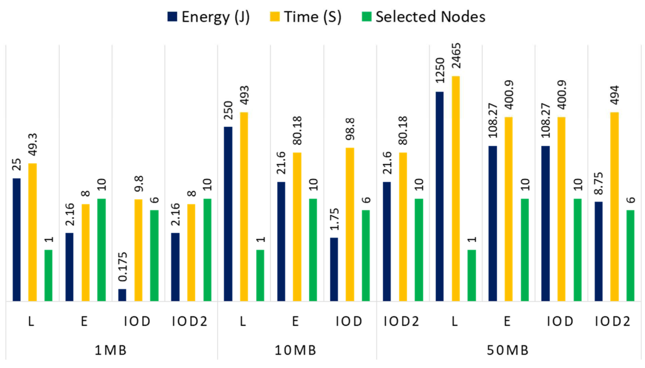


Fig. 3: Approach Evaluation Using Moderate Nodes

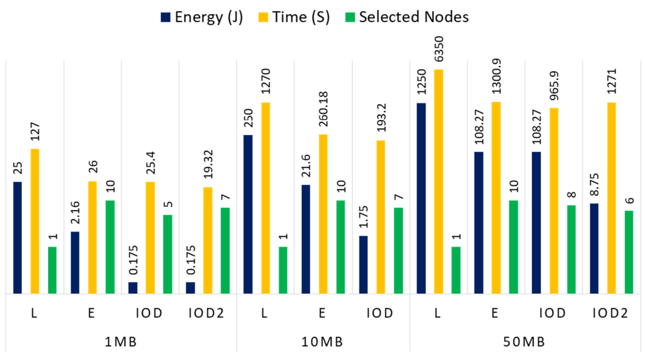


Fig. 4: Approach Evaluation Using Critical Nodes

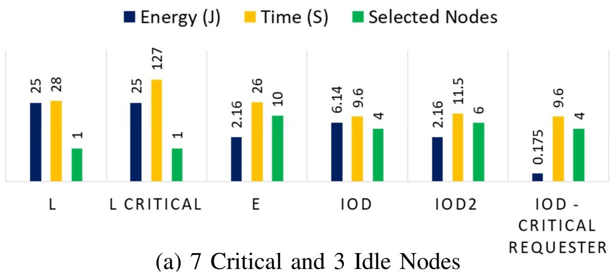
Distributor module to local execution and equal offloading. We vary the input size between 1MB, 10MB, and 50MB in order to test our IOD module under different situations. In each scenario, we monitor the execution time of the task and the energy consumption of the Requester Node. In addition, we keep track of the number of nodes used in the cloud with each of the above mentioned cases. Offloading equally to all nodes in the cluster shows a drastic improvement over local execution. For example, taking the 10MB input size, we notice a 91.3% decrease in energy consumption and 78.5% decrease in execution time. Our Intelligent Offloading Distributor, compared to the Equal offloading, enhances this result even more by reducing the energy consumption by 80.7%, decreasing the execution time by 4%, and cutting down the number of selected nodes by 40%. Hence it significantly improves the performance with respect to local execution decreasing energy consumption by 99% and time by 79.4%. Similar results are also shown with the 1MB and 50MB of input data. Therefore, using our IOD module leads to better result in which energy, time, and number of nodes are reduced.

Similarly, Figure 3 shows our approach's evaluation in a scenario where all nodes of the edge cloud are in a Moderate state. Offloading equally to all nodes in the cluster shows a drastic improvement over local execution. For example, taking the 1MB input size, we notice a 91% decrease in the energy consumption and 83.7% decrease in execution time. When comparing our IOD module with equal offloading, we notice 2 interesting outputs. The first one is represented by the IOD case and the second one by the IOD2 case. We notice that both results are actually good, however each one has its own usage. For example, assuming that energy is prioritized over time, result IOD can be adopted since it gives the minimum energy consumption, but at the cost of increasing execution time by 18.3%. Energy is reduced by 92% here due to offloading to less nodes since the number of selected nodes is lowered from 10 to 6 nodes compared to equal offloading. If we also compare this result (IOD) with local execution we notice a major improvement summarized by 99.3% and 80% decrease in energy and time respectively. Therefore, we come up with another motivation behind adopting this decision. In some situations the number of used nodes is a critical issue in a mobile ad-hoc edge cloud due to several reasons related to security such as malicious nodes, communication, trust etc. On the other hand, case IOD2 can be adopted when execution time is prioritized over energy since it results in more energy consumption but less execution time. It is worth to note here that IOD2 has the same results as that of equal offloading with regard to energy, time, and selected nodes. Very close results are also observed when the input size is changed to 1MB and 50MB. From this observation, we can perceive that equal offloading provides the best decision when it comes to execution time.

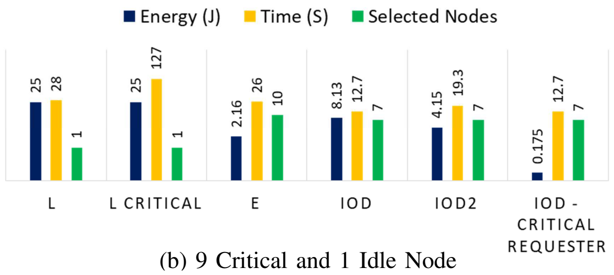
Again, Figure 4 illustrates our approach's evaluation in a scenario where all nodes of the mobile ad-hoc edge cloud are in a Critical state. Offloading equally to all nodes in the cluster shows a drastic improvement over local execution. For example, taking the 1MB input size, we notice a 91% decrease in energy consumption and 79.5% decrease in execution time while using the 10 nodes of the cloud. However, our IOD module proves again to be a better option than equal offloading. Compared to equal offloading, our IOD further reduces energy consumption by 92%, decreases execution time by 25.7%, and lowers the number of selected nodes in the cluster by 30% (from 10 to 7 nodes). Analogous outcomes are also observed when changing the input data to 1MB and 50MB. Consequently, the IOD unit decides to offload the entire file to 7 nodes only in a way that optimally reduces both energy and time.

Furthermore, assuming that nodes in an ad-hoc mobile edge cloud would always have the same computation state in real life environments is not realistic. For instance, some nodes might have a *Critical* state where several heavy applications are running, while others might be *Idle*. As such, we further extend our experiments to include such *Mix* situations where the edge cloud is composed of nodes with different states. We choose a few *Mix* assignments illustrated in table V. Figure 5 shows the variation of execution time, energy, and number of selected nodes using our Intelligent Offloading Distributor compared to Equal Offloading under *Mix* assignments and using a 1MB input size. In the first 3 cases (E, IOD, and IOD1), the Requesting Node is in an Idle state whereas it suffers from Critical situation in the last case (IOD2).

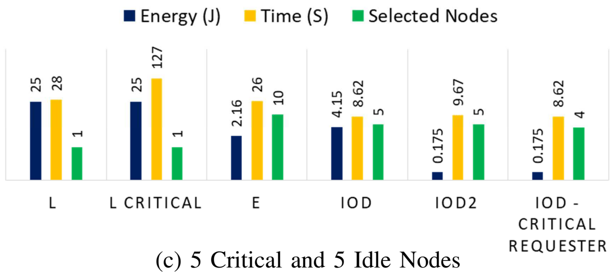
Figure 5a represents a scenario where the ad-hoc mobile edge cloud consists of 7 *Critical* nodes and 3 *Idle* nodes. In the first 3 cases (E, IOD, and IOD1), the *Requesting Node* is in an *Idle* state whereas it is *Critical* in the last case (IOD2). Equal offloading takes 26 seconds to finish the execution, consumes



(a) 7 Critical and 3 Idle Nodes

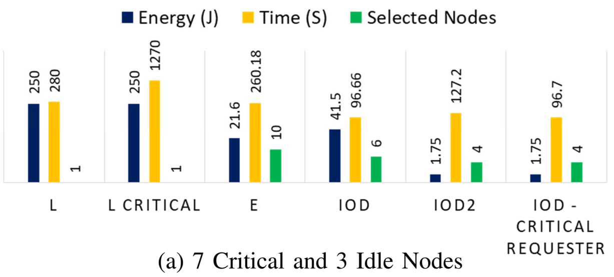


(b) 9 Critical and 1 Idle Node

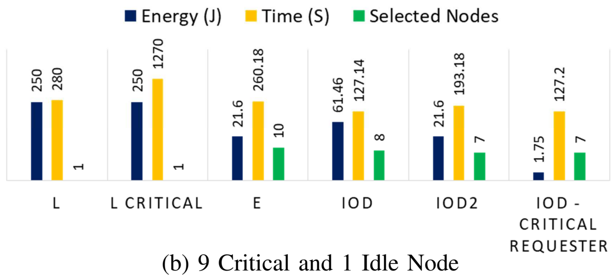


(c) 5 Critical and 5 Idle Nodes

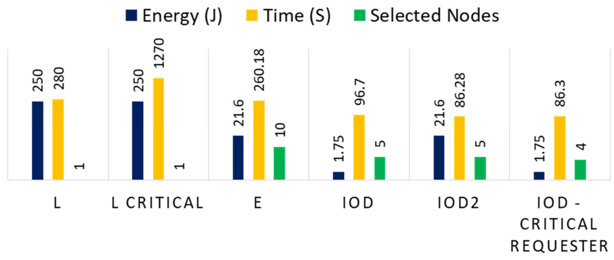
Fig. 5: shows the evaluation of our Intelligent Offloading Distributor using 1MB input under a mix assignment scenario.



(a) 7 Critical and 3 Idle Nodes



(b) 9 Critical and 1 Idle Node



(c) 5 Critical and 5 Idle Nodes

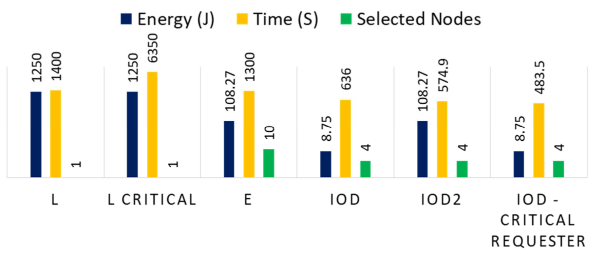
Fig. 6: shows the evaluation of our Intelligent Offloading Distributor using 10MB input under mix assignment scenarios.

$2,16J$ of energy, and uses 10 nodes. In case the *Requester Node* is *Idle*, IOD decides to use 4 nodes in the cluster instead of 10 in order to reduce the execution time by 63% at the cost of consuming $3.98J$ more energy (65% increase). This decision can be used when time is prioritized over energy. On the other hand, if energy is considered critical, then a different decision can be reached using our IOD module. Case IOD2, shows that using the same amount of energy, our *Intelligent Offloading Distributor* can still decrease the execution time by 55.7% and use 40% less nodes than an equal distribution. Moreover, In case the *Requester Node* is in a *Critical* state, our IOD module chooses to offload the entire input data on 4 nodes instead of 10 while decreasing both the energy by 91%, execution time by 65%, and selected nodes by 60%.

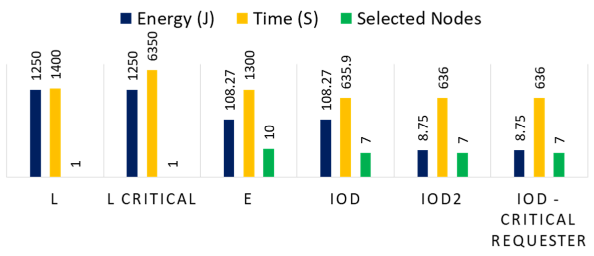
Figure 5b demonstrates a scenario where the ad-hoc mobile edge cloud consists of 9 Critical nodes and 1 Idle node. Equal offloading takes 26 seconds to finish the execution, consumes $2,16J$ of energy, and uses 10 nodes. In case the Requester Node is Idle, IOD decides to use 7 nodes instead of 10 in order to reduce the execution time by 51% at the cost of consuming 65% additional energy. This decision can be used when time is prioritized over energy. On the other hand, if energy is the more demanding factor, then a different decision can be used using our IOD module. Case IOD2, exhibits that 49% less energy can be consumed while increasing the execution time by34% and maintaining the same number of selected nodes (7) compared to the previous IOD result. Moreover, In case the Requester Node is in a Critical state, our IOD module chooses to offload the entire input data on 7 nodes instead of 10 decreasing both the energy by 91%, execution time by 51%, and selected nodes by 30%.

Figure 5c illustrates a scenario where the ad-hoc mobile edge cloud consists of 5 Critical nodes and 5 Idle nodes. Equal offloading takes 26 seconds to finish executing, consumes $2,16J$ of energy, and uses 10 nodes. In case the Requester Node is Idle, IOD decides to use 5 nodes instead of 10 in order to reduce the execution time by 66.8% at the expense of consuming 48% additional energy. However, if energy is the more demanding factor, then a different decision can be outputted using our IOD module. For example, Case IOD2, shows that offloading the entire file decreases the energy consumption by 92% and lowering the execution time by 62%. Both decisions cut down on the number of selected nodes by 50% (from 10 to 5 nodes). Comparing both IOD decisions shows that execution time is slightly prolonged by 1 seconds in IOD2 while decreasing energy by 95% compared to IOD. In the last case where the Requester Node is in a Critical state, our IOD module chooses to offload the entire input data using 4 nodes only instead of 10 decreasing both the energy by 91%, execution time by 66%, and selected nodes by 70%. To conclude, we show that our *Intelligent Offloading Distributor* significantly outperforms Equal offloading and it even performs better when the *Requester Node* is in a *Critical* state.

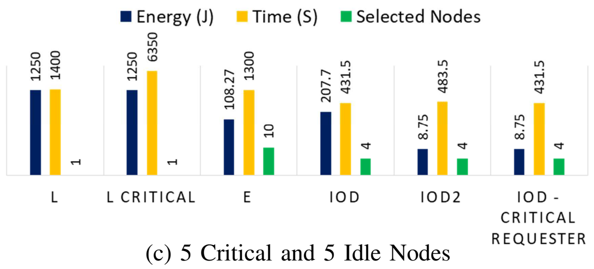
Figures 6 and 7 also show the variation of energy, time, and number of selected nodes in *Mix* assignments using 10MB and 50MB receptively where the mobile ad-hoc edge cloud consists of 9 *Critical* and 1 *Idle* node in the first figure and 5 *Critical* and 5 *Idle* nodes in the second figure. Both experiments illustrate that even with different input sizes, our Intelligent Offloading Distributor produces better results than Local execution and Equal Offloading. Analogous outcomes to that of the 1MB input size are observed in both figures.



(a) 7 Critical and 3 Idle Nodes



(b) 9 Critical and 1 Idle Node



(c) 5 Critical and 5 Idle Nodes

Fig. 7: shows the evaluation of our Intelligent Offloading Distributor using 50MB input under mix assignment scenarios.

Figure 8 portrays the efficiency and effectiveness of our Intelligent Offloading Distributor pertaining to the status of skipped nodes. In other words, in this experiment we monitor the number of skipped nodes and their status. We took a random sample from our previous experiments indicating 10 different outcomes of our IOD module. The experiment shows compelling results. 70% of the cases indicate that nodes with *Critical* status are the ones being skipped by the IOD module, while at the same time, as explained above, gaining powerful improvements over both Local execution and Equal Offloading. In the remaining 30%, the number of *Critical* skipped nodes is less than the number of skipped nodes by 1 or 2 nodes. This indicates that in some scenarios, our IOD module decides to skip some non-*Critical* nodes. This is reasonable and justified by having some situations in which the *Requester Node* is in an *Idle* state and skipped due to offloading the entire file in order to achieve a better result. In other cases, this is justified by skipping all *Critical Nodes* and some additional non-*Critical* nodes. Particularly the IOD determines to skip all *Critical* nodes to boost the effectiveness of the result, yet this effectiveness can still be enhanced by skipping other non-*Critical* nodes.

## IX. Conclusion and Future Work

This paper addressed problems related to the performance of running security services locally on smart devices. We introduced an ad-hoc mobile edge based cloud that provides security-as-a-service through efficient multi-layer cooperative offloading. Nodes forming an ad-hoc mobile edge cloud intelligently share their resources to reduce the load on the device executing the intrusion detection task. Our proposed approach is capable of reducing energy consumption and execution time via a set of components that rely on profiling, multi-objective optimization models and heuristics. The *Profiler* module is used for querying the status and current resource consumption of the nodes in the mobile ad-hoc edge cloud. The *Intelligent Offloading Distributor* module is used to allow a seamless smart offloading mechanism in the edge cloud that outputs the best offloading distribution based on optimization functions. In this context, the paper showed promising results. We were able to reduce energy consumption by 92%, execution time by 80% and number of selected nodes by 70% on average. Moreover, we illustrated the effectiveness of our IOD module with respect to the status of the skipped nodes. In more details, we showed that 90% of the skipped nodes by our IOD module are actually *Critical* Nodes.

For future work, the framework can be boosted to improve its performance and feasibility. Profiling can be enhanced especially with respect to the number of performed polling. In this work, participating nodes are assumed to be trusted, yet to target more real life scenarios, dynamic creation of the edge cloud and trust relations can be added for better management and selection of trusted nodes. Also, the number of selected nodes can be reduced in order to decrease the cost of offloading.

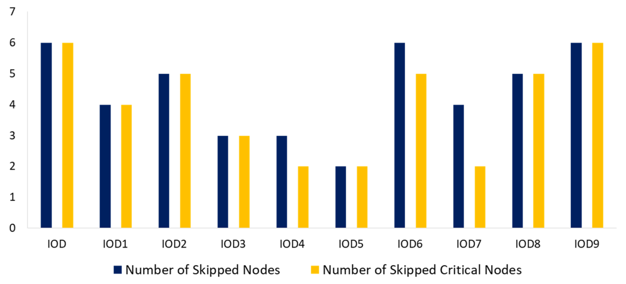


Fig. 8: Number of Skipped Critical Nodes

## References

[1] R. Mizouni and M. El Barachi, "Mobile phone sensing as a service: Business model and use cases," in *2013 Seventh International Conference on Next Generation Mobile Apps, Services and Technologies*. IEEE, 2013, pp. 116–121.

[2] R. Mizouni, A. Salah, S. Kolahi, and R. Dssouli, "Merging partial system behaviours: composition of use-case automata," *IET software*, vol. 1, no. 4, pp. 143–160, 2007.

[3] S. Peng, S. Yu, and A. Yang, "Smartphone malware and its propagation modeling: A survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 925–941, 2014.

[4] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, "Android security: A survey of issues, malware penetration, and defenses," *Communications Surveys & Tutorials, IEEE*, vol. 17, pp. 998 – 1022, May 2015.

[5] V. Rastogi, Y. Chen, and X. Jiang, "Catch me if you can: Evaluating android anti-malware against transformation attacks," *Information Forensics and Security, IEEE Transactions on*, vol. 9, pp. 99 – 108, December 2013.

[6] M. Jakobsson, "Why mobile security is not like traditional security," 2011.

[7] R. S. Khune and J. Thangakumar, "A cloud-based intrusion detection system for android smartphones," in *2012 International Conference on Radar, Communication and Computing (ICRCC)*. IEEE, December 2012, pp. 180–184.

[8] S. Zonouz, A. Houmansadr, R. Berthier, N. Borisov, and W. H. Sanders, "Secloud: A cloud-based comprehensive and lightweight security solution for smartphones," *Computers and Security*, vol. 37, pp. 215–227, September 2013.

[9] N. Soms, R. Priya, A. Banu, and P.Malathi, "A comprehensive performance analysis of zone based intrusion detection system in mobile ad hoc networks," in *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*. IEEE, March 2015, pp. 1–8.

[10] F. Barani, “A hybrid approach for dynamic intrusion detection in ad hoc networks using genetic algorithm and artificial immune system,” in *2014 Iranian Conference on Intelligent Systems (ICIS)*. IEEE, 2014, pp. 1–6.

[11] O. A. Wahab, H. Otrok, and A. Mourad, “A cooperative watchdog model based on dempster–shafer for detecting misbehaving vehicles,” *Computer Communications*, vol. 41, pp. 43–54, 2014.

[12] Q. Hassan, “Demystifying cloud computing,” *The Journal of Defense Software Engineering,(CrossTalk)*, pp. 16–21, Jan/Feb 2011.

[13] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, “Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading,” in *2012 Proceedings of IEEE INFOCOM*. IEEE, 2012, pp. 945–953.

[14] D. Chae, J. Kim, J. Kim, J. Kim, S. Yang, Y. Cho, Y. Kwon, and Y. Paek, “Cmcloud: Cloud platform for cost-effective offloading of mobile applications,” in *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2014, pp. 434–444.

[15] A. E. Eshratifar and M. Pedram, “Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment,” 2018.

[16] S. JoÅąilo and G. Dan, “Selfish decentralized computation offloading for mobile cloud computing in dense wireless networks,” *IEEE Transactions on Mobile Computing*, pp. 1–1, 2018.

[17] X. Lyu, H. Tian, L. Jiang, A. Vinel, S. Maharjan, S. Gjessing, and Y. Zhang, “Selective offloading in mobile edge computing for the green internet of things,” *IEEE Network*, vol. 32, no. 1, pp. 54–60, 2018.

[18] D. Deyannis, R. Tsirbas, G. Vasiliadis, R. Montella, S. Kosta, and S. Ioannidis, “Enabling gpu-assisted antivirus protection on android devices through edge offloading,” in *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*. ACM, 2018, pp. 13–18.

[19] D. Evans, “The internet of things: How the next evolution of the internet is changing everything,” *CISCO white paper*, vol. 1, pp. 1–11, 2011.

[20] X. Tao, K. Ota, M. Dong, H. Qi, and K. Li, “Performance guaranteed computation offloading for mobile-edge cloud computing,” *IEEE Wireless Communications Letters*, 2017.

[21] M. H. ur Rehman, C. Sun, T. Y. Wah, A. Iqbal, and P. P. Jayaraman, “Opportunistic computation offloading in mobile edge cloud computing environments,” in *Mobile Data Management (MDM), 2016 17th IEEE International Conference on*, vol. 1. IEEE, 2016, pp. 208–213.

[22] C. You and K. Huang, “Multiuser resource allocation for mobile-edge computation offloading,” in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–6.

[23] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, “Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks,” *IEEE access*, vol. 4, pp. 5896–5907, 2016.

[24] X. Chen, L. Jiao, W. Li, and X. Fu, “Efficient multi-user computation offloading for mobile-edge cloud computing,” *IEEE/ACM Transactions on Networking*, no. 5, pp. 2795–2808, 2016.

[25] W. Zhang, Y. Wen, J. Wu, and H. Li, “Toward a unified elastic computing platform for smartphones with cloud support,” *IEEE Network*, vol. 27, no. 5, pp. 34–40, 2013.

[26] C. Jin, J.-W. Choi, W.-S. Kang, and S. Yun, “Wi-fi direct data transmission for wireless medical devices,” in *The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014)*. IEEE, 2014, pp. 1–2.

[27] Y. Wang, A. V. Vasilakos, Q. Jin, and J. Ma, “A wi-fi direct based p2p application prototype for mobile social networking in proximity (msnp),” in *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing (DASC)*. IEEE, 2014, pp. 283–288.

[28] LiveQoS, “Superbeam,” https://play.google.com/store/apps/details?id=com.majedev.superbeam&hl=en/.

[29] Z. Inayat, A. Gani, N. B. Anuar, S. Anwar, and M. K. Khan, “Cloud-based intrusion detection and response system: open research issues, and solutions,” *Arabian Journal for Science and Engineering*, vol. 42, no. 2, pp. 399–423, 2017.

[30] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, “Paranoid android: versatile protection for smartphones,” in *ACSAC ’10 Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, NY, USA, December 2010, pp. 347–356.

[31] D. Damopoulos, G. Kambourakis, and G. Portokalidis, “The best of both worlds: a framework for the synergistic operation of host and cloud anomaly-based ids for smartphones,” in *EuroSec ’14 Proceedings of the Seventh European Workshop on System Security*, no. 6. ACM, NY, USA, April 2014.

[32] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for vm-based cloudlets in mobile computing,” *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[33] D. Spanos, “Intrusion detection systems for mobile ad hoc networks,” 2018.

[34] A. A. Korba, M. Nafaa, and Y. Ghamri-Doudane, “Anomaly-based intrusion detection system for ad hoc networks,” in *2016 7th International Conference on the Network of the Future (NOF)*. IEEE, 2016, pp. 1–3.

[35] P. Ramkumar, V. Vimala, and G. S. Sundari, “Homogeneous and hetrogeneous intrusion detection system in mobile ad hoc networks,” in *Computing Technologies and Intelligent Data Engineering (ICCTIDE), International Conference on*. IEEE, 2016, pp. 1–5.

[36] E. Froemling, “Bombsquad,” https://play.google.com/store/apps/details?id=net.froemling.bombsquad&hl=en.

[37] H. Smith, “Spaceteam,” https://play.google.com/store/apps/details?id=com.sleepingbeastgames.spaceteam&hl=en.

[38] T. Dbouk, A. Mourad, H. Otrok, and C. Talhi, “Towards ad-hoc cloud based approach for mobile intrusion detection,” in *2016 IEEE 12th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2016, pp. 1–8.

[39] O. A. Wahab, H. Otrok, and A. Mourad, “Vanet qos-olsr: Qos-based clustering protocol for vehicular ad hoc networks,” *Computer Communications*, vol. 36, no. 13, pp. 1422–1435, 2013.

[40] M. Norton and D. Roelker, “Snort 2.0: Hi-performance multi-rule inspection engine,” *Sourcefire Network Security Inc*, 2002.

[41] V. K. Kshirsagar, S. M. Tidke, and S. Vishnu, “Intrusion detection system using genetic algorithm and data mining: An overview,” *International Journal of Computer Science and Informatics ISSN (PRINT)*, vol. 2231, p. 5292, 2012.

[42] T. Lust and J. Teghem, “The multiobjective multidimensional knapsack problem: a survey and a new approach,” *International Transactions in Operational Research*, vol. 19, no. 4, pp. 495–520, 2012.

[43] E. Benkhelifa, T. Welsh, L. Tawalbeh, A. Khreishah, Y. Jararweh, and M. Al-Ayyoub, “Ga-based resource augmentation negotation for energy-optimised mobile ad-hoc cloud,” in *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. IEEE, 2016, pp. 110–116.

[44] P. Ghamisi and J. A. Benediktsson, “Feature selection based on hybridization of genetic algorithm and particle swarm optimization,” *IEEE Geoscience and Remote Sensing Letters*, vol. 12, no. 2, pp. 309–313, 2015.

[45] X. Zuo, C. Chen, W. Tan, and M. Zhou, “Vehicle scheduling of an urban bus line via an improved multiobjective genetic algorithm,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 1030–1041, 2015.

[46] H. Tout, C. Talhi, N. Kara, and A. Mourad, “Selective mobile cloud offloading to augment multi-persona performance and viability,” *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2016.

[47] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[48] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, “Accurate online power estimation and automatic battery behavior based power model generation for smartphones,” in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2010, pp. 105–114.

[49] Android, “Power Profile for Android,” https://source.android.com/devices/tech/power/.

**Toufic Dbouk** received BS and MSc degrees in computer science from the Lebanese American University. His current research interests include mobile computing, mobile resource management and intrusion detection systems.

**Azzam Mourad** received the Ph.D. degree in electrical and computer engineering from Concordia University, Montreal, Canada. He is an associate professor of computer science at the Lebanese American University and Affiliate Associate Professor in Software Engineering and IT department at the EÂĽcole de technologie superieure (ETS), Montreal, Canada. He served/serves as Associate Editor for IEEE Communications Letters, General Co-Chair of WiMob2016, and Track Chair, TPC member and reviewer of several prestigious conferences and highly ranked journals. He is an IEEE senior member.

**Hadi Otrok** holds an associate professor position in the department of ECE at Khalifa University, an affiliate associate professor in the Concordia Institute for Information Systems Engineering at Concordia University, Montreal, Canada, and an affiliate associate professor in the electrical department at EÂĽcole de technologie superieure (ETS), Montreal, Canada. He received his Ph.D. in ECE from Concordia University. He is a senior member at IEEE, and associate editor at: Ad-hoc networks (Elsevier), IEEE communications letters, wireless communications and mobile computing (Wiley). He co-chaired several committees at various IEEE conferences. Moreover, he is a TPC member of several conferences and reviewer of several highly ranked journals.

**Hanine Tout** received the PhD degree in software engineering at ETS, University of Quebec, Montreal, Canada. She is currently doing a Postdoc. between ETS and Ericsson, Montreal. Her research interests include 5G, IoT, security, privacy, machine learning, federated learning, mobile cloud computing, mobile virtualization, optimization, Web services, security and formal verification. She is serving as TPC member for SSCC-2018, NTMS 2016 and IMCET 2016 and a reviewer in IEEE Communications Letters, Computers and Security journal, Ad Hoc Networks journal and several international prestigious conferences.

**Chamseddine Talhi** received the PhD degree in computer science from Laval University, Quebec, Canada. He is an associate professor in the department of software engineering and IT at ETS, University of Quebec, Montreal, Canada. He is leading a research group that investigates smartphone, IoT and embedded systems security. His research interests include cloud security, secure sharing of embedded systems and IoT.