National College of Ireland

Mobile Computation Offloading to Adhoc Cloudlets with reliable security

MSc Research Project
Cloud Computing

Mahesh Prakash Ravi
Student ID: x21146331

School of Computing
National College of Ireland

Supervisor: Aqeel kazmi

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Mahesh Prakash Ravi
Student ID:	x21146331
Programme:	Cloud Computing
Year:	2022
Module:	MSc Research Project
Supervisor:	Aqeel kazmi
Submission Due Date:	1/02/2023
Project Title:	Mobile Computation Offloading to Adhoc Cloudlets with reliable security
Word Count:	7227
Page Count:	21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Mahesh Prakash Ravi
Date:	29th January 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Mobile Computation Offloading to Adhoc Cloudlets with reliable security

Mahesh Prakash Ravi
x21146331

Abstract

The most recent developments in wireless technology have led to the emergence of cutting-edge mobile phones as well as various smart devices. Handheld devices are popular among the public because of their mobility and also for their resource extensibility feature in the form of cloud technologies. While there are many benefits reaped from mobile devices, they are also a popular target for hackers who seek to access their sensitive data. Furthermore, when the task is offloaded outside of the mobile device, the critical data is in a compromised position because it becomes part of a large network. To make matters worse, adhoc computing's heterogeneous offloading environment makes the task most vulnerable to evolving threats. So it is important for application developers to defend application tasks from a variety of cyberattacks. Therefore, to address this issue, when a decision is made to offload to an adhoc host, it is ensured that the latest operating system updates are present by verifying its version. To be more specific, because operating system vendors frequently release security updates, having them installed on adhoc hosts makes them safer from attacks. This step ensures that task offloading is only performed on the upgraded adhoc hosts for small and critical tasks. As a result, even secure applications may be considered to be offloaded to ad hoc, which necessitates fundamental security. Also, through the simulations, it is also seen that the cost is reduced by almost 30 to 40% (depending on the cost set by the edge devices) when the same number of tasks are offloaded to adhoc hosts when compared to the edge cloudlets.

1 Introduction

Since technology-oriented development is adaptable and sustainable, practically every industry has embraced it. To be more specific, digital technologies have aided businesses in the rapid resolution of complex problems, saving them time and money. Anyhow, the quest for perfection in terms of infrastructure, design, performance, and other factors is continuing due to the evolution of modern gadgets like computers and smartphones. Despite the fact that computers sparked the digital revolution, mobile phones are nowadays famous, and creating applications for smartphones is the current trend. This behaviour is mainly due to the portability and user-friendliness of handheld devices, which enable consumers to utilise them at their convenience. To exemplify, the majority of services, including e-commerce shopping and stock market investing, are now supplied through mobile applications due to their ease of access. Additionally, the sensors included in mobile devices help users with navigation, motion detection, altitude awareness, etc., which

leads to sensing-oriented mobile application development.

However, mobile phones faced significant challenges due to their limited processing power and memory when it came to big data storage and complicated calculations involving many iterations. The primary explanation given for this is that handheld devices are made to be light and cannot support large-sized hardware due to their design. Additionally, there is a reliance on the lithium element for power because everlasting batteries to power mobile phones have not yet been invented. To compensate for their limitations, cloud technology was discovered to address issues related to the processing and storage power of mobile phones. The fundamental idea behind the cloud is that it offloads complex calculations and massive storage through data centers where numerous nodes operate continuously to support various applications. For instance, the most well-known cloud providers like AWS, Azure, and others provide services to many applications on a pay-as-you-go basis through large data centers across various geographical locations. However, the public cloud didn't offer a one-stop solution to every problem that devices with restricted resources encounter. Regions with poor or nonexistent technological infrastructure, particularly those with intermittent, patchy, and limited access, might exacerbate issues or cause cloud services to perform less efficiently by increasing their service time. In the case of real-time applications, the latency was supposed to be minimal for applications like video conferencing, online gaming, etc. where the cloud doesn't adhere in certain locations. As a result, Edge Computing evolved to address this issue by establishing a micro data center architecture with a small number of nodes close to the most accessible regions. Therefore, the applications that expect low latency have been far better served by this strategy than by the conventional cloud architecture. Also, due to the mobility of handheld devices, (Sun and Ansari; 2017) explains that cloudlet transfer happens along the route to support applications in order to avoid any latency deterioration.

Anyhow, in the network constraint areas, cloudlets, which are edge servers, were not always efficient and, in certain cases, not economical either. Ad hoc computing was a term used in the past to describe situations where research projects with little funding used public resources for computations. Liu et al. (2013) elaborates that whenever there is limited access to WiFi or cellular activity, adhoc mobile cloud can be used as an alternative task offloading method. In addition, Chen et al. (2015) highlights that combining mobile adhoc clouds with traditional cloudlet architecture would result in a strong offloading environment capable of serving applications seamlessly. However, it is seen as less secure because the network's nodes are unregulated and diverse in nature, involving a variety of mobile devices and PCs. Additionally, there is no service level agreement in place to ensure that everyone taking part in the pooling of resources follows all security requirements. Shila et al. (2017) asserts that security in adhoc clouds is crucial during all offloading stages, such as partitioning, device discovery, processing, and so forth. To prevent any potential vulnerabilities in the future, the proper countermeasures must be implemented even during the design phase.

This study looked into how operating system versions could be checked before task offloading to ad hoc devices such as PCs and smartphones. Based on whether it has the most recent releases for the particular operating system, the node will be chosen. This guarantees that the vendor's security fixes are available and trustworthy to offload critical tasks from the programs that demand fundamental protection. It will make use of the

open community project for updates that manages the version releases across the various platforms.

1.1 Research Question

Based on the introduction above, the project's motivation is framed by the following rephrased research question, which paves the way for the experimentation and the extraction of the expected results.

How may adhoc cloudlets be prioritized for cost savings while also taking into account fundamental security considerations to accomplish mobile task offloading?

1.2 Objectives of the research

The brief research has motivated to achieve the following objectives while designing the application.

1. To reduce the cost of offloading for atleast small size tasks.
2. With the aim of reducing offloading cost the basic security should not be jeopardized.
3. Even for the non-delay sensitive applications, latency should be kept to an acceptable level if not the real time.

The following sections are classified based on the given structure. Section 2 identifies and summarizes the literature review done on various works based on mobile task offloading. It also uses a tabulation that is created to highlight the key publications.The tools, methodology, algorithms, and research methodologies described in Section 3 provide a foundation for putting the idea into practice.The proposal is concluded in Section 4 with the experimental results and a forecast of potential future work that might be done to improve the study even more.

2 Related Work

Mobile phones have advanced significantly, but they still have problems because of their computing and battery power constraints. Mobile computing and cloud computing finally came together to solve the problems and overcame the limitations of mobile phones. These technologies facilitated the offloading of computations and storage from cell phones to larger systems like data centres and neighbouring servers. Therefore, it reduced the demand for mobile devices with low configurations and helped people who needed desktop computer-like functions.

2.1 Low latency - An Imperative factor

According to (Fernando et al.; 2013), in order to initiate timely actions, real-time apps need to be instantly responsive, which means that they require compute-intensive resources for performing complex calculations or executing repetitive tasks over a large dataset with varying data points. For instance, speech synthesis and natural language

processing are two of the most well-known real-time applications, yet they are also challenging for mobile devices to perform due to their limited computational capability. The author also emphasises how cloud computing has eliminated the aforementioned problem with mobile devices through offloading. When considering cloud services as an augmentation of the mobile device, it is possible to boost the processing capability of mobile phones by offloading data or calculations to cloud data centres. The intensive parts of computations are typically performed remotely via networks such as wide area networks.

The fundamental objective of the offloading method is to reduce communication costs between the mobile device and surrogates. But clouds are geographically far away from users, necessitating a number of hops for the client to transport or retrieve data. In addition, (Ren et al.; 2019) demonstrate that, despite its ability to scale in terms of resource delivery, cloud computing was unable to meet the needs of applications that were time-sensitive. Systems using IoT suffer from an unsatisfactory response time from the cloud as a result of the WAN's high latency. The growth of WAN traffic brought on by IoT data is another aspect that contributes to cloud restrictions. (Shu et al.; 2008) claims that when data is transmitted from fog devices, the transferable packets are typically quite large. This frequently results in transmission rates to the cloud exceeding their current bandwidth capability.

2.2 Arrival of Edge Computing

As time moved on, edge computing, which offloads tasks and storage to neighbouring cloudlets, was developed by researchers as a way to get around the problems with cloud offloading. The goal behind edge computing is to bring cloud apps, capabilities, and services closer to users—specifically, one hop away—and to the network edge, as shown in the Figure 5. (Babar et al.; 2021) studied that cloudlets are used in contexts where quick latency must be maintained, and their capabilities enable interaction with physical sensors, pattern recognition, and even video analytics to improve performance. The author also cites that even if there is an outage in the enterprise cloud, persistent connectivity can be attained through cloudlets, and pending operations like permanent can finally be synced once the cloud resumes regular operations. (Sun and Ansari; 2017) highlights that these cloudlet arrangements can be used by mobile devices that are resource-constrained to offload their tasks and prevent network connection delays. However, to avoid an overrun of cloudlet resources while offloading tasks and storage, workloads must be divided equally. Therefore, there are a number of factors that must be taken into account when making decisions about offloading, whether it is based on task or storage, and they are greatly influenced by the kind of application being used as well as the environment.

2.3 Overview of Existing Offloading Methodologies

Based on task offloading decisions, various frameworks and proposals have emerged over time. For example, Fan and Ansari (2018) explains that even though there is a minimal network latency guarantee, when the cloudlet workload is high, the offloaded processes cannot fully utilise the edge resources. This thus has an impact on the response time for an application, which should be as quick as possible given the volume. He then put out an application-aware task allocation that takes network delay and compute delay into

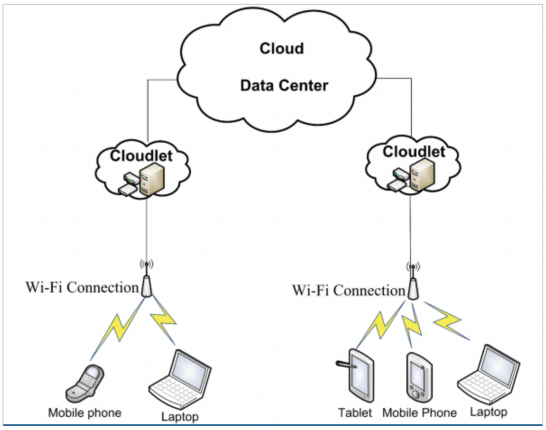

Figure 1: Basic Cloudlet Architecture(Babar et al.; 2021)

account by using the heuristic AREA algorithm. In this approach, as the request type and load vary depending on the device, choosing the cloudlets would be based on the criterion that they are not overly occupied. This ensures the quality of service agreements, which are established based on the assured response time.

Secondly, offloading tasks is always made more difficult by mobility, and under those circumstances, a wise choice must be made based on situational awareness. Xia et al. (2014) suggested a two-tiered design that would allow for access to the cloudlet via numerous Wi-Fi wireless access points, whereas in remote clouds, access is via cellular networks in order to overcome mobility difficulties. In this approach, a weighted bipartite graph is constructed such that the weighted sum of all matched edges is minimised to offload tasks onto matched edges. Using this method, it is verified that the energy consumption is fairly employed using this proposed location-aware offloading framework, extending the lifespan of their mobile gadgets without compromising the SLA requirement. Chen et al. (2016) emphasises that, due to the fact that mobile devices' coordinates vary as a result of travel, WiFi is only guaranteed to be connected in indoor settings and cannot ensure uninterrupted service for such devices from cloudlets. To address this issue, they proposed locating telecom clouds near wireless base stations and connecting them via fibre for improved responsiveness even when using mobile data. In this method, numerous requests from various mobile users could result in interference because base stations operate in multi-channel configurations. As a result, Chen et al. (2016) put forth a distributed compute offloading algorithm that converges to a Nash equilibrium of the multi-user computation offloading game within a finite number of decision slots.

In recent times, intelligence offloading mechanisms have been extensively studied, and deep learning algorithms are being used to advance offloading decision-making methods. For example, Qu et al. (2021) offered a deep meta-reinforcement learning model that combined the advantages of DRL with meta-learning. In this approach, the framework is responsible for learning the states to predict the best offloading decisions for tasks by training the model parameters. In addition, when the environment changes, it utilizes the memory playback for further learning.

2.4 Ad hoc Offloading

Similar to edge cloud, the fundamental idea behind a mobile edge cloud is that various mobile and stationary devices, such as PCs, are linked together in order to utilize them as a cloudlet in a distributed manner for computation offloading activities. According to Fernando et al. (2011), Ad hoc computing, also known as volunteer computing, is a type of distributed computing model that makes use of the unused personal components owned by different individuals either connected to the same network or using D2D communication to carry out difficult and computationally heavy tasks. At first, taking part in ad hoc networks was thought to be best accomplished via personal computers. However, because mobile phones have advanced, including smartphones and tablets, they are now included in this volunteer paradigm.

In addition, Chen et al. (2015) highlights that in order to keep the adhoc mobile cloud alive, the stable nodes can share the network with other bandwidth constrained nodes. For these reasons, combining mobile adhoc clouds with traditional cloudlet architecture would result in a strong offloading environment capable of serving applications seamlessly. However, an adhoc network is dynamic in nature, as the user may end up sharing the resources at any point. Since the volunteers are not bound by any service level agreements, fault tolerance mechanisms should be enforced through the migration to avoid any state or data loss. In order to shorten the overall duration of the offloaded operations, Sulaiman and Barker (2017) recently designed a multisite adaptive offloading system where the decision-making would be accurate even with contextual changes. Both complete and partial unloading were the subjects of the experiment, and the findings are encouraging where the task offloading is performed on the basis of scoring with respect to node capabilities. Their research supported the idea that not all activities require full offloading. For example, searching tasks in the text file shows that small and medium jobs are better off being executed locally as opposed to remotely, even if it results in a tiny gain in latency but decreases network cost.

Likewise, due to the sporadic nature of D2D connectivity, Li et al. (2014) proposed optimal and periodic access algorithms for node discovery and subtask allocation in order to conserve energy. Under an optimal access scheme, it is ensured that a task response is returned after offloading by using a heuristic access scheme where the regularity of the mobility is explored. On the other hand, the period access scheme scans intermittently for task offloading in order to avoid any privacy issues. Moving forward, Chen et al. (2015) presented an "opportunistic ad hoc cloudlet service" (OCS) that offers a compromise between the drawbacks of connected and remote cloud services (CCS). The researcher states that as the D2D connection in adhoc has short connection issues, that time is sufficient to transfer content related to computation during that period. With the OCS method, even if the server connection is lost with the client, the computation will keep going on the offloaded device until the job is finished. This type of strategy avoids the mobility issue and also has different approaches to retrieving the task result.

2.5 Security during Offloading

In remote cloud or edge computing, the service providers are mostly big players like Amazon or Azure, where they are bound by the service level agreement to ensure that they don't violate the user's privacy. Abolfazli et al. (2014) cites that these providers

use virtualization technology to create separation from other apps running on the same computer. They also frequently patch software, encrypt documents, and, at times, even use outside contractors to oversee technical aspects of their operations. This is very important because if the offloaded task often involves sensitive data, then there is a high possibility that it will become a potential threat when it is acquired through unauthorised access by attackers. According to Wu et al. (2019), task offloading in edge networks is mostly done by semi-authorized entities, which leads to data leaks. In addition, certain cloudlet providers share the resources only for certain activities, which impacts the effectiveness of the task offloading. So trustworthiness is a crucial factor not only for availing services but also for maintaining users' privacy. In order to assess the privacy risks brought on by disclosing sensitive information during offloading, Wu et al. (2019) proposed a trust-aware offloading framework based on task sensitivity that also considers latency and energy consumption. Through this framework, the author devised a classification problem using machine learning to formalise the computing activity so that trustworthy service providers can be chosen for task offloading.

Similar to this, Shila et al. (2017) presented the Autonomic Cloud Framework (AMCloud), a two-layer secure interface architecture that supports peer-to-peer discovery and includes device layer protection. Based on their prior behavior, trust system, etc., the AMCloud network's service discovery module appropriately excludes any malicious participants and also supports encryption-based task offloading. Likewise, Dbouk et al. (2019) designed a smart offloading decision using a heuristic algorithm that incorporated the dynamic profiler of the surrounding nodes. Here, fitness is evaluated through the analysis of possible solutions to the problem and also undergoes the survival test. The proposal also guaranteed the identification of intrusions through comparison with the predefined malicious dataset. Similarly, many security-related recommendations to safely offload the duties to the nearby nodes have been offered in the past. However, one must make sure that, at the very least, the providers' fundamental security updates have been updated. For instance, in order to reduce the system's exposure to the most dangerous attacks, Sawadogo et al. (2020) points out that every operating system and software provider routinely publishes patches for vulnerability rectification. Therefore, he suggested that machine learning techniques may be used to automatically identify source code modifications that genuinely reflect security updates.

To illustrate, as shown in the *Android Security Bulletin-september 2022;* (2022), Android security patches are frequently released and it is strongly advised to update them to avoid being vulnerable to attacks. Despite the fact that several studies concentrated on power, latency, and security concerns, it was discovered that not one of the articles made the choice based on the operating system version. For instance, despite the fact that the multi-adaptive framework MaMoC proposed by Sulaiman and Barker (2017) covers the majority of crucial parameters, such as power, latency, battery life, etc., no emphasis is placed on the node's trustworthy perspective. Out of all the studied research, Dbouk et al. (2019) suggestion for investigating deeply into adhoc networks comes the closest, but they only look for intrusion detection in service nodes. So, to bridge the gap, the security component will be integrated into the suggested model to increase the reliability of the offloading mechanism. As a result of the proposal, secure apps can now be trusted on ad hoc devices for task offloading, which in turn reduces latency.

3 Methodology

Mobile task offloading has a wide range of potential scenarios, and it is difficult to conduct real-time testing due to the complexity of the ad hoc configuration. Thus, the simulation method was used with the Edgecloudsim open source tool. It was chosen from a large number of open-source simulation tools because it had related classes that were offloaded and could be easily extended and customized. The Java-written code for the edgecloudsim framework serves as the foundation for the implementation. In this method, the offloading decision is influenced by the operating system of the adhoc devices being confirmed by a third-party open-source community initiative. The community is in charge of keeping track of all operating system version releases, including when updates expire. This paper demonstrates an example of how to create distributed, secured adhoc offloading with an optimal decision-making process.

3.1 EdgeCloudSim- An Overview

Simulators evolved to help researchers evaluate and optimise their proposals and designs due to the repeatability and scalability of experiments. One of the most popular frameworks on the market is EdgeCloudSim, which came to the aid of researchers looking into edge computing architectures, particularly in an offloading scenario. The key component of edge computing analysis, the orchestration of actions, is made easier by EdgeCloudsim in addition to resource provisioning. It is an open-source project that enables users to combine modules in accordance with the requirements generated by the simulation experiments and is also an extension of the CloudSim framework. As a result, it allows the researchers to push their limitations by adjusting capacity loads, making offloading choices, or even suggesting effective methods.

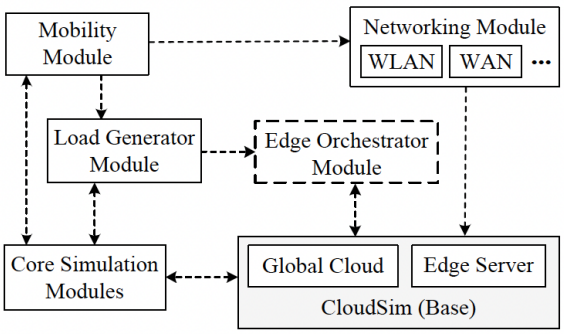

Figure 2: Relationship between EdgeCloudSim modules(Sonmez et al.; 2018)

3.1.1 Modules of EdgeCloudSim

The core components of edgecloudsim are listed below and the major change proposed is on the edge orchestrator module.

1. **The core simulation module:**The main duty of this module is to load and execute edge computing scenarios from configuration files that represent the edge computing properties. In addition to being customizable, this facility is crucial for evaluating designs.

2. **The edge orchestrator module:**The orchestrator module controls the resources to improve the efficiency of the entire system. This is accomplished by making crucial choices about the creation of new replicas, the termination of edge VMs, the management of hosts' computing resources, and the offloading of jobs to cloud or edge servers.

3. **The networking module:**By considering both upload and download data, this module specifically addresses transmission delays in WLAN and WAN. The single server queue model, which operates on the first come, first served principle, is the foundation of the networking module's default implementation which can be customized.

4. **The mobility module:**One of EdgeCloudSim's highlights is that it takes mobility into account when simulating, so the locations of mobile devices are updated in accordance with the mobility configurations. Each mobile device in our design has x and y coordinates that are updated in conjunction with the dynamically controlled hash table.

5. **The load generator module:**The load generator module is in charge of producing tasks of various sizes for the specified application setup so that they can be used for the research study. As can be seen in the figure 2, the mobility and load generator modules are the primary elements that supply input to other elements.

3.2 Outline of Fuzzy Logic Approach

Generally, things that are unclear or ambiguous are said to be fuzzy. Fuzzy logic provides extremely useful flexibility for reasoning in situations where the system is unable to tell whether a state is true or false. Due to the dynamic nature of bandwidth and utilization with respect to routes and servers, it is often challenging to demonstrate effective workload orchestration. Since the task load varies on each offload, the parameters mentioned are highly dynamic and are directly proportional to the number of users in the network as well as the varying task workload. Due to this, conventional optimization techniques cannot be used for the orchestration as the offloading requests are not known in advance.Sonmez et al. (2019)

Fuzzy logic is advised as an alternative to edge orchestration for addressing scenarios in cloud-based environments in order to avoid its limitations. This method was chosen for the research because it effectively manages uncertainty in dynamic contexts despite the absence of sophisticated mathematical models. To support the adopted approach, it previously managed the decision-making process that involved several criteria easily inside the same framework Abdullah (2013) through the form of human language.

The fuzzy logic Architecture contains four parts :

1. **RULE BASE:** This component consists of a set of rules created by a researcher to control the decision-making process based on linguistic data. There have been numerous recent advancements that have improved the rule definition approaches, and they can be selected depending on the demand.

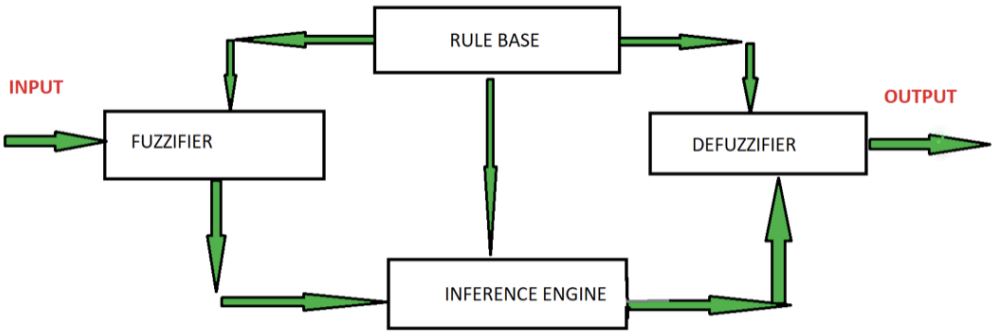

Figure 3: Fuzzy Logic Architecture*Fuzzy logic: Introduction* (2022)

2. **FUZZIFICATION:** This component can be used to transform the input integers into fuzzy sets that can be understood by humans. The variables that can be used to represent the numbers include, for example, sensor inputs or, in our case, bandwidth, job size, etc.

3. **INFERENCE ENGINE:** It calculates which rules should be fired based on the input field and the degree to which the current fuzzy input matches each rule. To create the control actions, the fired rules are then concatenated.

4. **DEFUZZIFICATION:** It's used to turn the fuzzy sets the inference engine generated into a numerical value. There are various defuzzification techniques available; the most effective one is combined with a particular expert system to minimize error.

3.3 Utilization of Open Source Community Project -EOL

Many GitHub projects were developed by a single person or a particular team to fulfill their requirements. But when a project is open source, it begins to evolve through contributions from the public, such as adding features or fixing problems, so that it can be used for more varied purposes as well.Finley (2022) As a result, the initiative ultimately becomes a community project and is frequently used as a model for other projects in different sectors once it becomes stable. These projects are self governing and self organizing and maintained through the sponsorship. However, it should be noted that some Open Source initiatives now receiving funding come from businesses, and some contributors are paid.Engard (2010)

The website endoflife.date is one such open source community project which serves for the information purpose that tracks support cycles and release cycles of over 180 products with a concise summary of the release policy. Additionally, it offers rest API endpoints for retrieving the version history in an easily accessible format, in accordance with the official releases for the relevant product. It was created with Jekyll, a Ruby static site generator that runs GitHub Pages, and the finished product was deployed to Netlify. The maintainers rigorously review the contributions, and a suitable guidance is supplied for the new product on-boarding as well as the suggested revisions.

3.4 Application Design

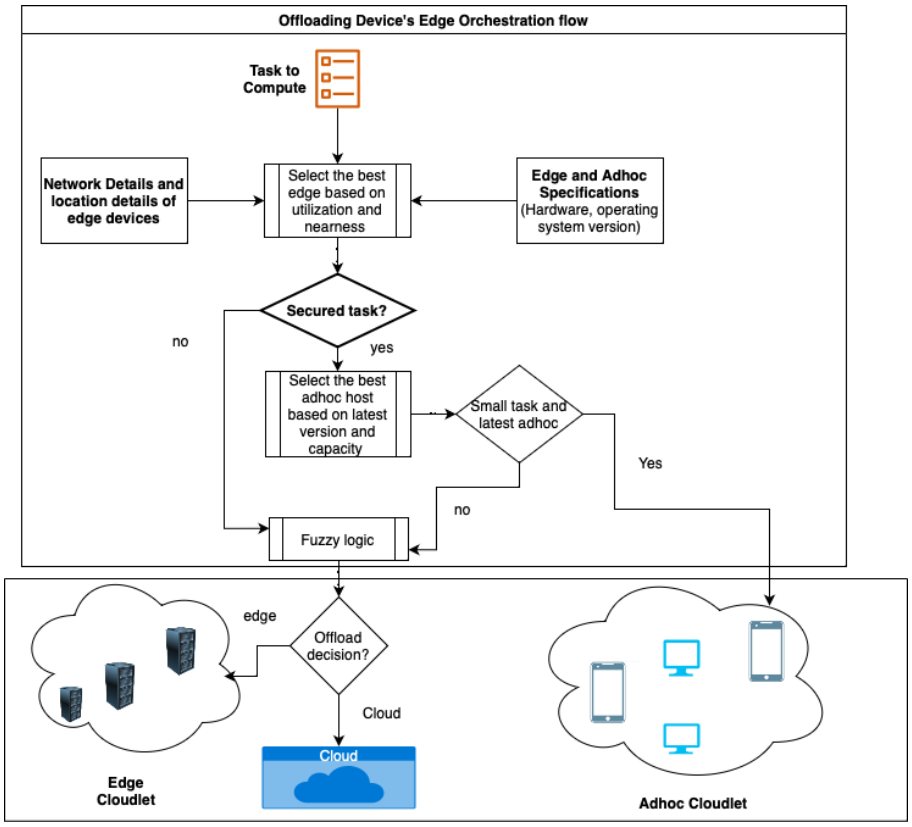

Figure 4: Secured Adhoc Task Offloading Workflow

In the proposed design, the tasks are generated by the load generator module based on the application configuration which is done in XML file. The edge server manager present in the simulation module is responsible for creating the hosts based on the edge devices configuration which also creates adhoc device cloud for the task assignment. Among the all created hosts, the best edge is selected based on the current utilization and the proximity so that later it can be chosen when the offloading decision is edge. Then the consent for secured offloading in the form of flag is taken so based on that the versions of adhoc devices are verified with the open source community project and then the best device is selected based on the capacity. In parallel, the tasks are identified based on the size and when it is small then it will be offloaded to cloud in order to get the faster response. For the other sized tasks, it will be fed into the fuzzy logic where the offloading decision will be calculated based on the various parameters which includes bandwidth, delay sensitivity and bandwidth.

4 Implementation

The proposal can be implemented in two ways: either through the development of a mobile application and then offloading to nearby devices, or by performing the same operation in a simulation environment like EdgeCloudSim. The simulation technique, however, was selected over the other two because it is difficult to quickly set up the infrastructure necessary for real-time offloading of mobile applications in real time. Also,

mobile applications require cloud formation corresponding to adhoc devices with different operating systems, which is challenging to pool accordingly, and scalability will also be limited.

4.1 Algorithm

The core algorithm for the research is defined below, and it is termed BSOAD, which otherwise called as Basic Secured Offloading on Adhoc Devices.

Algorithm 1 Basic Secured Offloading on Adhoc Devices(BSOAD)

Data: EdgeDevicesList(datacentres and AdhocDevicesList), ApplicationList, offloadingCharacteristics, adhocSecuredOffload
Result: Efficient offloading of tasks thus reducing the cost and response time

```
for i ← minNoOfMobileDev to maxNoOfMobileDev by mobileDevCounterSize do
    for i ← 0 to simulationScenarios by 1 do
        for i ← 0 to orchestratorPolicies by 1 do
            createhosts(charasteristicsofhosts);
            Generate the tasks based on application configuration
            Submit the tasks for offloading
            if adhoc device has the latest os then
                if task is small then
                    choose best capacity of adhoc and offload
                else
                    select the host by fuzzy logic( edge or cloud)
                end
            else
                select the host by fuzzy logic( edge or cloud)
            end
            Retrieve the host for offloading
            Calculate the cost and other default metrics such as time for each application
            Log the measurements in the SimLogger
        end
    end
end
```

The input of this framework is retrieved from various configuration files such as edge devices list, application list, properties file, and also additional flags such as secured, which indicates the tagged application is a critical application. Initially, It iterates based on the counter size of the mobiles and the number of mobile devices configured. The simulation scenario iteration process then begins, allowing multiple scenarios to be evaluated; in our case, "TWO_TIER_WITH_EO" is defined, indicating that we are evaluating multi-tier with edge orchestrator. After that, the orchestrator policies iteration shows that we are carrying out a fuzzy-based offloading. Following many level iterations, the subsequent stage is to generate hosts based on the device configuration described in the input files(adhoc, cloud, and edge—) configurations. The tasks are then generated based on the application setup and submitted for offloading. Finally, fuzzy logic is used to analyze the hosts depending on the various scenarios listed in the FCL definition file.

For small tasks, adhoc hosts are examined to see if they have the most recent Operating System. The best host will be selected through all of these tests and then offloaded to the appropriate hosts. Finally the cost is calculated after the successful offloading in the SimLogger utility.

4.2 Technical details

The listed Table 1 technologies, which are identical to those required by edgecloudsim because the research proposal is constructed on top of the framework, must be deployed in the development environment.

Tools and Technology	**Purpose**
Eclipse	Integrated Development Environment for simulating the application
Java	Programming language used to develop the modules of the edgecloudsim
XML	Used to define the configurations of the edge architecture
MATLAB	To display the result in the required format
EdgeCloudSim	Framework which is used is retrieved from the github and serves as a base for the simulation

Table 1: Technologies used to develop the project

According to the system where the project runs, the versions and packages (for either Intel or Mac) might change and be downloaded correspondingly from the official websites of those packages. The research was conducted on the Macbook Pro M1, which has a sufficient infrastructure setup to conduct the experiments.

4.3 Customization of Framework for Adhoc Proposal

The following are the modules of edgecloudsim where the customization is done according to the proposal.

4.3.1 Configuration files

Configuration files are the driving factor of the edgecloudsim, and based on the scenarios to be tested, the following files can be modified to measure the outcomes for research.

1. **Applications XML:** For applications to indicate whether they are secure or not, the new parameter "secured" is introduced. The flag 1 denotes a crucial application, while the flag 0 denotes a non-critical program. Even the secured tag's absence indicates that it is a non-critical application. Only the health applications were taken into account for the experiment in this study, and all other applications were left out (which was configured by default for the edgecloudsim framework).

2. **Edge Devices XML:**The adhoc devices are defined with the adhoc-specific host configuration, such as operating system name, version, and device type, which can be either computer or mobile, along with the data center list. In addition, respective

reference locations were updated to take adhoc hosts into account when creating hosts.

4.3.2 Task generation

The class IdleActiveLoadGenerator implements the fundamental load generator model, where modules generate tasks during the active time and wait during the idle period. Also, the jobs are automatically produced using a Poisson arrival mechanism. To be more specific, the task generated by the model resembles the activities carried out on the mobile device, which is essentially how real-world mobile devices operate. Additionally, the application configuration file specifies the task arrival time (load generation period), idle intervals, and active periods. As a result, the default load generator model effectively assists the suggested model using numbers with an exponential distribution.

4.3.3 Edge orchestrator

The EdgeOrchestrator implements basic algorithms, which are first/next/best/worst/random fit algorithms, while assigning requests to the edge devices. This proposal focuses mainly on this module, where the basic class is extended to orchestrate the device selection for offloading. Following are the core functions of the edge orchestrator module which were included to support the experiment.

1. **CheckDeviceVersion:** In this module, the open source community website is used to check the incoming adhoc device version for security updates as well as its recentness. The JSON response from the REST call is parsed based on the adhoc device type before being filtered for security support. Some operating systems have different JSON responses, which are parsed accordingly. For example, some vendors release minor versions, whereas some releases are like cycles. This function will accommodate all the possible scenarios.

2. **GetDeviceToOffload:**This module is in charge of picking an appropriate host for the offloading functionality. In short the proposed algorithm is implemented in this function. The VM list also fetched for the corresponding edge devices and the utilization is calculated both on total and average. The best edge device is originally chosen based on use and proximity. The task is then put through a series of scenarios using fuzzy logic to determine whether it should be delegated to the cloud or the edge.

   The rules are defined in the FCL definition class and it represents standard considerations for offloading . It is assumed that the edge orchestrator app is running on the edge devices in a distributed manner. Similar to this, the adhoc device is checked for operating system updates depending on whether the incoming task is critical or not. If devices with the most recent security updates are discovered, small tasks are offloaded to those, while larger tasks use the aforementioned technique. Since there is no guarantee that the ad hoc device would participate for a long period, this approach would be feasible.

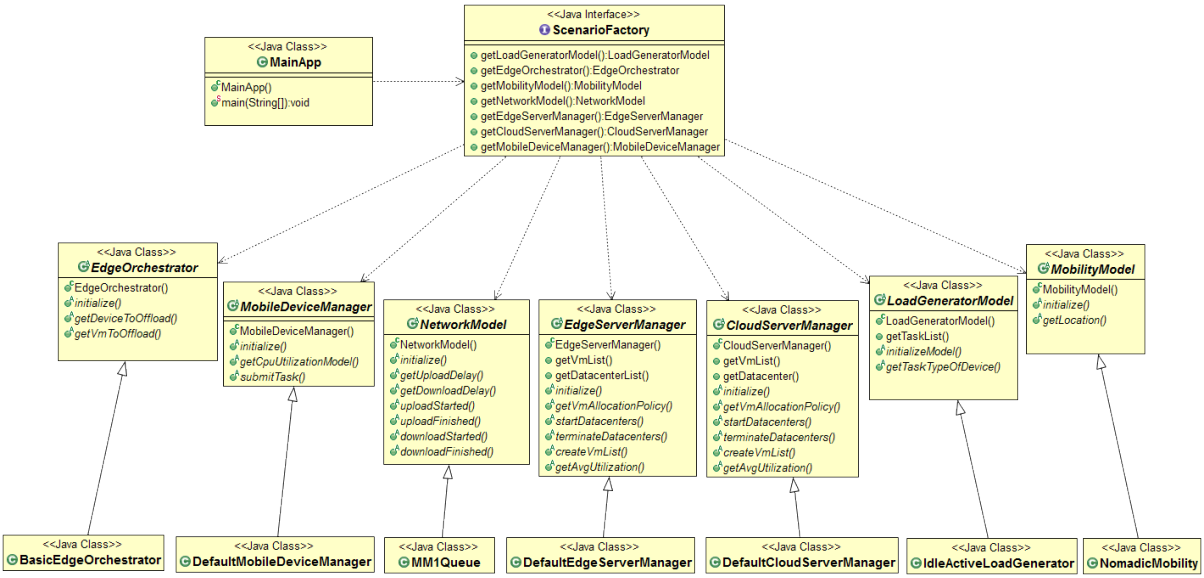

Figure 5: Class Diagram of the Implemented Classes Sonmez et al. (2018)

4.3.4 SimSettings

The parsing validations are configured to skip validations for adhoc devices , which are specific to edge devices. This can be achieved by the attribute edgevalidation, which is mentioned as false for ad-hoc devices. The datacenter and adhoc lists are merged to create a common host list. Similarly, the related locations where the host list is specified are tailored to acquaint the adhoc devices.

4.3.5 Cloud Configuration

The default framework's properties for the cloud are reused for this proposal because our objectives are around on the edge architecture. The important configuration includes cores, storage, and million instructions per cycle (MIPS), which will be parsed by the program and utilized for the cloud setup. With the edge orchestrator, the simulation scenario is divided into multiple tiers with adhoc setup where cloud is the top most tier.

4.3.6 Version Control

A completely different project is implemented separately from the EdgeCloudsim framework. The required packages are included as Java packages, and the configuration files are included in the resources package. To promote changes and enhancements, the entire code structure is hosted in the github source control management system.

5 Evaluation

Using the aforementioned implementation and environment configuration, the edgecloudsim simulation is run with the different scenarios. The simulation is run numerous times to record observations in a comparable range because the task generator generates tasks of different sizes.

5.1 Experiment 1-Without Adhoc

In the first experiment, a health application with a medium task length is offloaded to the nearby edge devices. The default configurations are reused from edgecloudsim's edge devices configuration file, which represents the datacenter list for simulations, whereas the cloud configurations are defined in the properties file. Moreover, this application is designed to be offloaded with basic security measures because the majority of the health application data is anticipated to be safeguarded in real time.

5.1.1 Parameters of Edge Devices

The following are the important configurations used for the research which are the defaults of the edgecloudsim.

1. **Task Length:**3000
2. **Cost Per second for edge datacenters:**1-3 dollars
3. **Processing Power of edge datacenters:**8000 mips
4. **Processing power of cloud:**100000

5.1.2 SimLogger logs for the task sized 210

The Figure 6 represents the simlogger output without adhoc devices.

```
HEALTH_APP
# of tasks (Edge/Adhoc/Cloud): 210(210/0/0)
# of failed tasks (Edge/Adhoc/Cloud): 0(0/0/0)
# of completed tasks (Edge/Adhoc/Cloud): 210(210/0/0)
_____
# of tasks (Edge/Cloud/Mobile/Adhoc): 210(210/0/0/0)
# of failed tasks (Edge/Cloud/Mobile/Adhoc): 0(0/0/0/0)
# of completed tasks (Edge/Cloud/Mobile/Adhoc): 210(210/0/0/0)
# of uncompleted tasks (Edge/Cloud/Mobile/Adhoc): 0(0/0/0/0)
# of failed tasks due to vm capacity (Edge/Cloud/Mobile): 0(0/0/0)
# of failed tasks due to Mobility/WLAN Range/Network(WLAN/MAN/WAN/GSM): 0/0/0(0/0/0/0)
percentage of failed tasks: 0.000000%
average service time: 0.428334 seconds. (on Edge: 0.428334, on Cloud: NaN, on Mobile: NaN, on Adhoc: NaN)
average processing time: 0.389609 seconds. (on Edge: 0.389609, on Cloud: NaN, on Mobile: NaN, on Adhoc: NaN)
average network delay: 0.038725 seconds. (LAN delay: 0.038725, MAN delay: NaN, WAN delay: NaN, GSM delay: NaN)
average server utilization Edge/Cloud/Mobile: 0.002128/0.000000/0.000000
average cost: 129.3742857142857$
```

Figure 6: SimLogger Output for the experiment without adhoc devices

Since the datacenter vendors are frequently updated with the latest security measures against vulnerable attacks they can be offloaded directly into those hosts where the tasks will be loaded into the newly created VM's. This characteristics suitable for both edge or cloud but the other fundamental factors such as latency, bandwidth and the processing power were still considered for offloading through the fuzzy logic. Based on the delay sensitivity, the host will be chosen either based on the edge devices or to the cloud. The measurements are logged in the bar chart from the readings of SimLogger as shown in Figure 7.

5.2 Experiment 2- With Adhoc obselete version

The same experiment is carried out with adhoc devices, but with outdated versions. For example, the adhoc's OS version for this Ubuntu adhoc cloud is older from the most recent version, which was retrieved from the open source website.

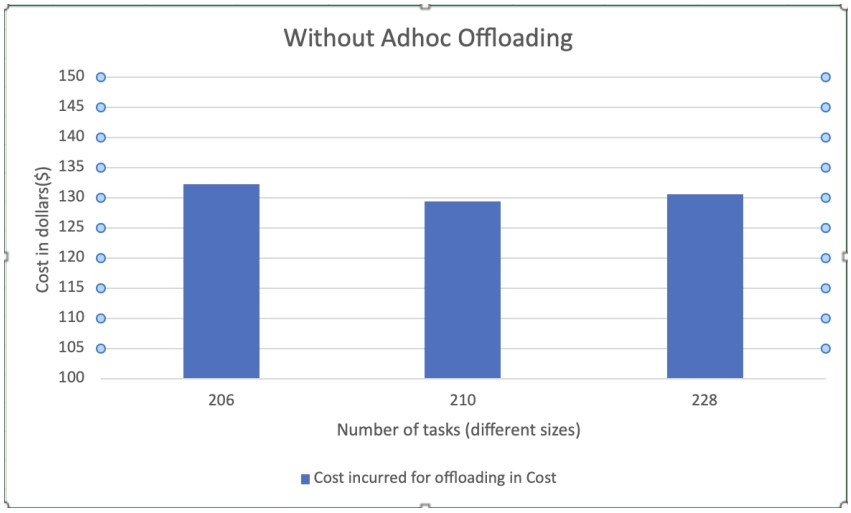

Figure 7: Simulation of EdgeCloudSim without adhoc devices

5.2.1 Parameters of Adhoc Devices

The following are the important configurations used for the research which are the parameters of adhoc devices.

1. **Cost Per second for edge datacenters:**0 dollars
2. **Processing Power of edge datacenters:**100 mips(assumption of multiple adhoc devices connected by a network.for example computer lab)
3. **Operating System:**Ubuntu
4. **Version:**2.10

In this case, the implementation will automatically ignore the Ubuntu ad hoc hosts as the health application is offloaded to be secure. Similarly, the host can be configured with many ad hoc devices, even mobile devices with the Android operating system and the corresponding version. For this experiment, all the configured ad hoc hosts are kept to be old so that the system ignores for the secured application. Along with the latest parameters, even the security support is verified through the open source website. This is achieved through the comparison of the EOL attribute with the current date. Whenever the current date crosses the support date, then it will be considered obsolete. As a result, the outcome is similar to the 5.1 when considering the cost when compared to the number of tasks. Similarly, the simlogger output also similar to Figure 7.

5.3 Experiment 3- With Adhoc Latest Version

In the third experiment, the Ubuntu operating system is configured with the latest version which is **Ubuntu 22.10** and will be validated through the open source website. Similarly, the android operating system also updated to latest which mean that they will be part of offloading system. In order to minimize the latency only the tasks of size less than 1000 considered to be offloaded in the adhoc devices.

```
HEALTH_APP
# of tasks (Edge/Adhoc/Cloud): 216(160/56/0)
# of failed tasks (Edge/Adhoc/Cloud): 0(0/0/0)
# of completed tasks (Edge/Adhoc/Cloud): 216(160/56/0)
---------------------------------------
# of tasks (Edge/Cloud/Mobile/Adhoc): 216(160/0/0/56)
# of failed tasks (Edge/Cloud/Mobile/Adhoc): 0(0/0/0/0)
# of completed tasks (Edge/Cloud/Mobile/Adhoc): 216(160/0/0/56)
# of uncompleted tasks (Edge/Cloud/Mobile/Adhoc): 0(0/0/0/0)
# of failed tasks due to vm capacity (Edge/Cloud/Mobile): 0(0/0/0)
# of failed tasks due to Mobility/WLAN Range/Network(WLAN/MAN/WAN/GSM): 0/0/0(0/0/0/0)
percentage of failed tasks: 0.000000%
average service time: 2.912139 seconds. (on Edge: 0.434858, on Cloud: NaN, on Mobile: NaN, on Adhoc:
9.990084)
average processing time: 2.873902 seconds. (on Edge: 0.399636, on Cloud: NaN, on Mobile: NaN, on
Adhoc: 9.943232)
average network delay: 0.038237 seconds. (LAN delay: 0.035249, MAN delay: 0.011527, WAN delay: NaN,
GSM delay: NaN)
average server utilization Edge/Cloud/Mobile: 0.005652/0.000000/0.000000
average cost: 86.6592592592593$
```

Figure 8: SimLogger Output for the experiment with adhoc devices

In this case, the support date will be greater than the current date when verified with the open source website. This means that the vendor still has the support for the adhoc device then it is regarded as a safe device to offload. So the decision will be the adhoc, then the host with the great processing power is selected for the better latency. The adhoc can be configured either be computer or even mobiles. The verification process will be repeated for all the configuration hosts to shortlist the secured devices. The measurements are logged in the bar chart from the readings of SimLogger as shown in Figure 9. Similarly, the simlogger output also similar to Figure 9.

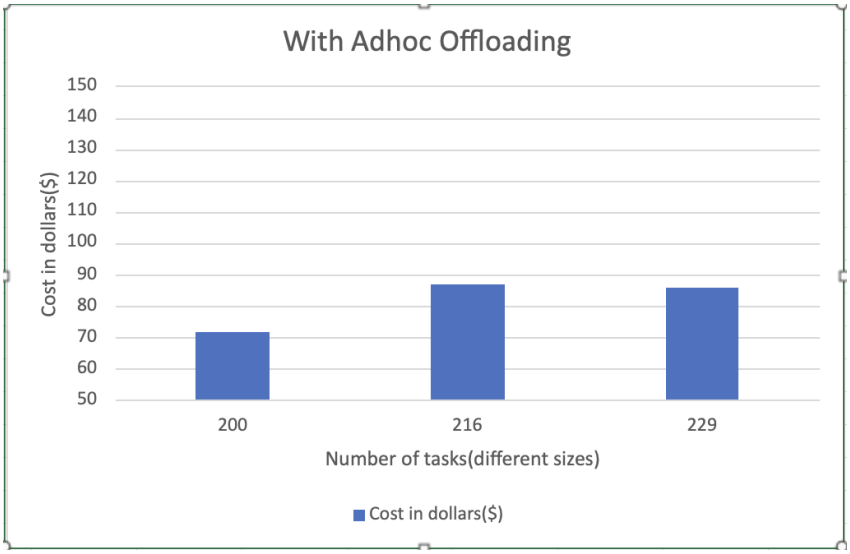

Figure 9: Simulation of EdgeCloudSim with adhoc devices

5.4 Discussion

According to the above experiments, the inclusion of adhoc devices resulted in a greater reduction in the cost of executing the offloaded tasks. To be more precise, Figure 7 represents the cost against the number of tasks that have been offloaded. The mapping shows that based on the cost per second set for different data centers, the cost is surging based on the processing time. In contrast, when looking at Figure 9 based on the task size and the number of offloaded tasks on the ad hoc devices, the cost has been reduced

by almost 30 to 40 percent. This behavior is due to the fact that there is no cost involved when executing tasks on ad hoc devices. In other words, the resources for offloading are contributed by volunteers. Even though latency is compromised to a significant degree due to the lower processing power than the data centers, ad hoc offloading can be chosen based on the less delay-sensitive applications or the tasks that can be run in the background.

Furthermore, application developers can be relieved of adhoc offloading vulnerabilities because the most recent version is always checked when the application is deemed critical. Secondly, the developers can cross-check the operating system vendor's security updates in the current releases because the most recent version is always verified for the secured apps. With this, developers can also choose whether to offload to an adhoc system or not with the introduction of the adhoc execution flag. Because of the controlled offloading approach, it is safer to offload the task to adhoc devices rather than pay the cost of processing it at the edge or in the cloud. Moreover, this cost-cutting behavior is seen even when the various applications are offloaded in the same way. In addition, it can be observed from the logs mentioned in Figure 8 that the service time is around 2 seconds longer due to the extra time consumed by the adhoc device but still in few seconds which can be accepted for delay insensitive tasks. As a result of the findings, the research's stated objectives have been met with the simulation approach. However, there is a restriction on introducing multiple hosts for an adhoc device in edgecloudsim, therefore it is believed that there is only one host per adhoc configuration. This is mostly caused by the requirement for customization of the many dependent modules, particularly its parent module, cloudsim, which has an impact on the behavior of the framework as a whole. Furthermore, as we haven't implemented real-world offloading, it is challenging to mimic attacks and demonstrate the effects of defensive behavior. This is why the security parameter was not tested.

6 Conclusion and Future Work

In the past, various offloading frameworks and ideas have been proposed, but the focus was only on limited metrics, and not enough research was done on a complete solution as a whole. The proposal aimed to improve Sulaiman and Barker (2017) research work, which was accomplished by introducing new parameters such as cost and security. This step paves the way for the developers if the application is intended to be budget-friendly while offloading tasks. Additionally, the processing capability of the ad hoc environment is further increased by encouraging additional volunteers to contribute their computing assets, such as computers or mobile devices, which enables the offloading of even medium-sized activities. In future work, this proposal can be further improved by introducing compression techniques while offloading the task, which will reduce the transfer time to the offloading environment. Since there is a limitation to performing the approach in the simulation framework and limited time, that approach was not carried out to enhance the research. Also, the same model can be further tested in real world applications to further validate the outcomes and possibly verify the defense mechanism of adhoc devices against the attacks as well.

References

Abdullah, L. (2013). Fuzzy multi criteria decision making and its applications: a brief review of category, *Procedia-Social and Behavioral Sciences* **97**: 131–136.

Abolfazli, S., Sanaei, Z., Ahmed, E., Gani, A. and Buyya, R. (2014). Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges, *IEEE Communications Surveys Tutorials* **16**(1): 337–368.

Android Security Bulletin-september 2022; (2022).
URL: *https://source.android.com/docs/security/bulletin/2022-09-01*

Babar, M., Khan, M. S., Ali, F., Imran, M. and Shoaib, M. (2021). Cloudlet computing: Recent advances, taxonomy, and challenges, *IEEE Access* **9**: 29609–29622.

Chen, M., Hao, Y., Li, Y., Lai, C.-F. and Wu, D. (2015). On the computation offloading at ad hoc cloudlet: architecture and service modes, *IEEE Communications Magazine* **53**(6): 18–24.

Chen, X., Jiao, L., Li, W. and Fu, X. (2016). Efficient multi-user computation offloading for mobile-edge cloud computing, *IEEE/ACM Transactions on Networking* **24**(5): 2795–2808.

Dbouk, T., Mourad, A., Otrok, H., Tout, H. and Talhi, C. (2019). A novel ad-hoc mobile edge cloud offering security services through intelligent resource-aware offloading, *IEEE Transactions on Network and Service Management* **16**(4): 1665–1680.

Engard, N. (2010). *Practical open source software for libraries*, Elsevier.

Fan, Q. and Ansari, N. (2018). Application aware workload allocation for edge computing-based iot, *IEEE Internet of Things Journal* **5**(3): 2146–2153.

Fernando, N., Loke, S. W. and Rahayu, W. (2011). Dynamic mobile cloud computing: Ad hoc and opportunistic job sharing, *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pp. 281–286.

Fernando, N., Loke, S. W. and Rahayu, W. (2013). Mobile cloud computing: A survey, *Future generation computer systems* **29**(1): 84–106.

Finley, K. (2022). What to do when your open source project becomes a community?
URL: *https://github.blog/2022-06-30-what-to-do-when-your-open-source-project-becomes-a-community/*

Fuzzy logic: Introduction (2022).
URL: *https://www.geeksforgeeks.org/fuzzy-logic-introduction/*

Li, Y., Sun, L. and Wang, W. (2014). Exploring device-to-device communication for mobile cloud computing, *2014 IEEE International Conference on Communications (ICC)*, pp. 2239–2244.

Liu, F., Shu, P., Jin, H., Ding, L., Yu, J., Niu, D. and Li, B. (2013). Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications, *IEEE Wireless Communications* **20**(3): 14–22.

Qu, G., Wu, H., Li, R. and Jiao, P. (2021). Dmro: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing, *IEEE Transactions on Network and Service Management* **18**(3): 3448–3459.

Ren, J., Zhang, D., He, S., Zhang, Y. and Li, T. (2019). A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet, *ACM Computing Surveys (CSUR)* **52**(6): 1–36.

Sawadogo, A. D., Bissyandé, T. F., Moha, N., Allix, K., Klein, J., Li, L. and Traon, Y. L. (2020). Learning to catch security patches, *arXiv preprint arXiv:2001.09148* .

Shila, D. M., Shen, W., Cheng, Y., Tian, X., Shen and Sherman, X. (2017). Amcloud: Toward a secure autonomic mobile ad hoc cloud computing system, *IEEE Wireless Communications* **24**(2): 74–81.

Shu, L., Zhang, Y., Zhou, Z., Hauswirth, M., Yu, Z. and Hynes, G. (2008). Transmitting and gathering streaming data in wireless multimedia sensor networks within expected network lifetime, *Mobile Networks and Applications* **13**(3): 306–322.

Sonmez, C., Ozgovde, A. and Ersoy, C. (2018). Edgecloudsim: An environment for performance evaluation of edge computing systems, *Transactions on Emerging Telecommunications Technologies* **29**: e3493.

Sonmez, C., Ozgovde, A. and Ersoy, C. (2019). Fuzzy workload orchestration for edge computing, *IEEE Transactions on Network and Service Management* **16**(2): 769–782.

Sulaiman, D. and Barker, A. (2017). Mamoc: Multisite adaptive offloading framework for mobile cloud applications, *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 17–24.

Sun, X. and Ansari, N. (2017). Latency aware workload offloading in the cloudlet network, *IEEE Communications Letters* **21**(7): 1481–1484.

Wu, D., Shen, G., Huang, Z., Cao, Y. and Du, T. (2019). A trust-aware task offloading framework in mobile edge computing, *IEEE Access* **7**: 150105–150119.

Xia, Q., Liang, W., Xu, Z. and Zhou, B. (2014). Online algorithms for location-aware task offloading in two-tiered mobile cloud environments, *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pp. 109–116.