National College of Ireland

Mobile task offloading based on bandwidth and battery availability

MSc Research Project
Msc of Science in Cloud computing

Andrew Jaskaniec
Student ID: 17132266

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Andrew Jaskaniec
Student ID:	17132266
Programme:	Msc of Science in Cloud computing
Year:	2018
Module:	MSc Research Project
Supervisor:	Vikas Sahni
Submission Due Date:	20/12/2018
Project Title:	Mobile task offloading based on bandwidth and battery availability
Word Count:	5663
Page Count:	19

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:	
Date:	25th July 2019

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 task offloading based on bandwidth and battery availability

Andrew Jaskaniec
17132266

Abstract

Mobile code offloading is very promising method of delegating tasks from mobile device to the cloud in order to reduce battery consumption and save mobile resources. There are many available methods and frameworks to offload mobile processing to the cloud including Clonecloud, Comet, Tango and MobiCop. This report describes benefits of decision engine residing on a device and running during app launch to provide response whether to offload specific task to the cloud or run it on a device. It looks at device hardware specifications, analyzing battery availability, CPU and memory usage and network availability to provide commitment to a decision. Proposed software is deployed on Android devices and shown classes can be used in any app with specific changes required based on Operating System version. With this software it can be seen that based solely on decision making engine running on mobile device, it can be decided with high confidence to offload the process to the cloud to be completed before battery is fully drained taking into consideration wireless network restrictions.

1 Introduction

With over 5 billion mobile phone users around the world, smart phones with their capabilities on par with PCs are powerful multitasking tools. Their benefits involving ease of access, portability and affordability can be limited by various factors, which users and developers have to overcome on daily basis. These main flows include large battery consumption coupled with small size of the battery and also different network bandwidth restrictions. Currently it is hard to find a device that impresses with running one full day on one battery charge with high usage. While multi-tasking and running CPU intensive processes, most of mobile phones will not last a day. Similar with network access, while on slow networks, may it be 3G or public wifi, running different tasks can be disruptive. It has been proven that code offloading helps in processing large tasks which are running on a device, when a device is not able to complete them or owner wants to transfer the task to different device for completion. Code offloading from mobile devices to cloud can mitigate current flaws in mobile devices segment, mentioned earlier battery capacity and network bandwidth. It can be seen in this report that offloading processes from mobile devices to the cloud can be done effectively by using decision making engine run during app runtime without connecting to external source. In comparison to different offloading strategies that may require copying of the app code to the cloud and predicting specific network condition, delegating task to cloud based solely on real time computation can be

as effective. This work displays benefits of decision making engine running on a device which measures connection status, speed and battery availability to send process to the cloud or make decision to finish the process on a device.

2 Related Work

2.1 Android resources usage detection

In order to make a decision whether to proceed or not with offloading scenario, service must correctly evaluate CPU, memory, wifi and battery usage of current tasks. Merlo *et al.* investigated the feasibility of constructing power-consumption-based sensors to measure consumption rates of wifi and CPU on mobile devices (1). In that work different power consumption metering approaches have been analyzed and divided into three categories: low, medium and high level measurements, based on the method of the data collection from the device. The tests have been done from a security standpoint, therefore authors concentrated on identifying threats to the device.

Low level measurements have been provided by Appscope tool and FEPMA (2), which have been both extremely invasive to kernel and tied specifically to device models. High level measurements have been done by app, PowerTutor (3), which injects high level usage data of the smart device hardware into its models to show its energy consumption. It includes 6 components: CPU, LCD as well as gps, wifi, audio and cellular interfaces, and for 10 second intervals it is accurate to within 0.8 percent and most 2.5 percent error. Last method of measurements described by Merlo *et al.* (1) is middle level measurement, in which battery usage and level are measured without connecting to kernel, as most new lithium ion batteries have this information already stored in files: sys/devices/platform/ds2784-battery/getcurrent, and sys/devices/platform/ds2784-battery/getvoltage. This means that low level information can still be reused with application software.

For the use case of task offloading and to achieve certain level of security, middle level measurement results have been chosen as the most promising for this project, while low and high level measurements will not be considered.

2.2 Code offloading

In a recent times problem of deployment different software to mobile devices, including internet of things has become much more visible, due to increased demand for high end small form factor devices and deployment of most popular services and applications to mobile devices as well. Same idea around internet of things, where all sensors and connected devices require vast amounts of processing power, which very often is not provided by only one device.

Code offloading is a concept which developers are working on, most recently Benedetto *et al.*, investigated framework for code offloading in internet of things (4). The authors present MobiCOP-IoT solution, framework that is designed to support code offloading for iOT and android Things devices using cloud and fog ecosystems. This solution consists of client libraries installed on a device and server infrastructure, which receives off loading call from a client. Whenever a task is dispatched by device, service queries integrated decision making engine to commit to decision of offloading the task, running it locally or using both approaches.

Another idea described in that paper is decision making engine, incorporated in a solution on client side, which consists of network and code profiler. Network profiler estimates and samples network every 15 min (differently on metered and unmetered network) and checks latency and reliability of connection. Code profiler, on the other hand, uses heuristic-based algorithm to measure the time taken to run given tasks on a device. After evaluation, decision is being made if code should be run on a device, offloaded to the cloud or run the code concurrently on both.

Benchmarking results of this solution prove that code offloading in cloud environment from mobile and iOT devices is improving performance while not incurring additional costs, which can be beneficial in my research.

Further exploration of task offloading, mainly concentrated on fog computing was reported by Aazam *et al.* in "Offloading in fog computing for iOT: Review, enabling technologies and research opportunities"(5). This review presents various criteria including excessive computation or resource constraint, latency requirements, load balancing, permanent or long term storage, data management and organization, privacy and security, accessibility, affordability, feasibility and maintenance. These are valid points for both iOT deployments and mobile device systems.

Authors describe types of middleware technologies used in iOT applications- cloudlets, mobile edge computing, micro data centre, nano datacentres, delay tolerant networks and Femto cloud. My research will be exploration of mobile edge computing coupled with femto cloud approach, which utilizes co-located devices in order to harvest computational or storage capabilities.

Aazon's paper also reviews technology enablers for offloading tasks in fog computing-wireless technology, which is required to properly offload task and its availability. This is crucial for my research, because slower or less reliable technologies are insufficient.

Second enabler are smart and intelligent autonomous applications, which are using machine learning and utilizing context aware services, subsequent is virtualization and containment and finally, parallelism, to enable concurrent offloading on different nodes. Authors critically analyze as well usages of fog computing and offloading processes like sensors offloading to fog, smartglasses to smartphone, fog and edge nodes offloading to cloud or cloud offloading to fog.

Finally, this review also summarizes challenges and limitations that are important in task offloading processes. This deployment could be impacted by resources allocation and scalability, so finding the right amount of resources that can be offloaded, depending on that amount system can be under utilized or offloading process trigerred too often. Second issue is scalability of the application and its handling of multiple of client connections. Another challenge mentioned in that paper is SLA between two entities (task offloaded and provider of services), although my research will not consider service level agreements. Security aspects were also considered. It plays an important role in process/tasks management and needs to be taken into consideration, especially now, with critical data residing on mobile devices. Other relevant challenges for mobile task offloading are: integration and interoperability, fault tolerance, energy consumption trade off, Incentives for offloading service and monitoring, which have to be included in a design.

In Refactoring Android Java Code for On-Demand Computation Offloading (6), Zhang *et al.* present Dpartner, tool that automatically refactors android applications to be supporting computation offloading capability. Dpartner analyzes application code to discover the parts worth offloading, and then creates two artifacts that are to be deployed on android device and server, respectively. Ranges of improvement in execution time are from

46 to 97 percent, while battery power consumption is lowered by 27 to 83 percent. Refactoring design, as proven by authors via their extensive tests using linpack, chess and 3D car game can be important for my research. However repackaging each mobile apk file to support this solution will not work on global scale with new Operating Systems and security enhancements enabled in them (to mention Google SafetyNet, unknown sources or usb debugging). This approach will cause apps to be marked as harmful to use in corporate or even personal environment.

Computation offloading has been a topic of research since the early days of computing and certainly many scientists looked at it from mobile perspective. The first practical implementation of this idea for Android Operating System was explored by Kemp *et al.* in Cuckoo: a computation Offloading Framework for Smartphones (7). To minimize impact for app developers, authors designed framework that offered very simple programming model, prepared for connectivity drops and bundled all in one package. They integrated it with existing development tools and automated large chunks of development process and offered simple way to collect remote resources.

As in Refactoring Android Java Code, Cuckoo system requires app to be developed first using specific framework to enable intelligent offloading. While it has its benefits, it requires specific standardization and redeployment of apps currently residing in Google Play (for Android) or Apple store (iOS). Framework provides two Eclipse builders and an Ant build file that can be inserted into an android project's build configuration in Eclipse. Cuckoo integrates into runtime and decides whether the method invoked should be run locally and remotely. If remotely, then it can be offloaded to any machine running Java virtual machine. For my research it is important to consider, as one of aspects of this exploration is to be able to offload tasks to another mobile device (in close proximity). Additional part of cuckoo framework is Resource manager application running on a smartphone. It uses QR code displayed on a screen to register known resource (in this case server), which will be used to offload any computation directed by Cuckoo, allowing to be used repeatedly for any application that uses Cuckoo offloading framework. In this paper authors present two projects using their framework, eyeDentify- a multimedia content analysis application that performs object recognition of images captured by the camera of a smart phone and Photoshoot, distributed augmented reality game which uses augmented reality and face detection algorithm in order to determine which player won the shootout. Both evaluation projects required little work to enable computational offloading using Cuckoo framework. Reviewed system integrates build processes significant for my work, although does not incorporate encapsulation of the software and requires, code development from the beginning (new software release).

2.3 MCC architecture

Mobile cloud computing (MCC) provides integration of cloud computing and mobile environment and overcomes obstacles related to performance, environment and security that smart phones can potentially face. Dinh *et al.* present overview of benefits and issues encountered in "MCC in Survey of mobile cloud computing: architecture, applications and approaches" (8).

General architecture of MCC (Figure 1) shows mobile devices connecting to mobile networks via satellites, access points or base receiver stations. After connection to internet is established, device is able to connect to different Cloud distributed networks, which consists of different layers. Data center layer provides physical infrastructure, linking

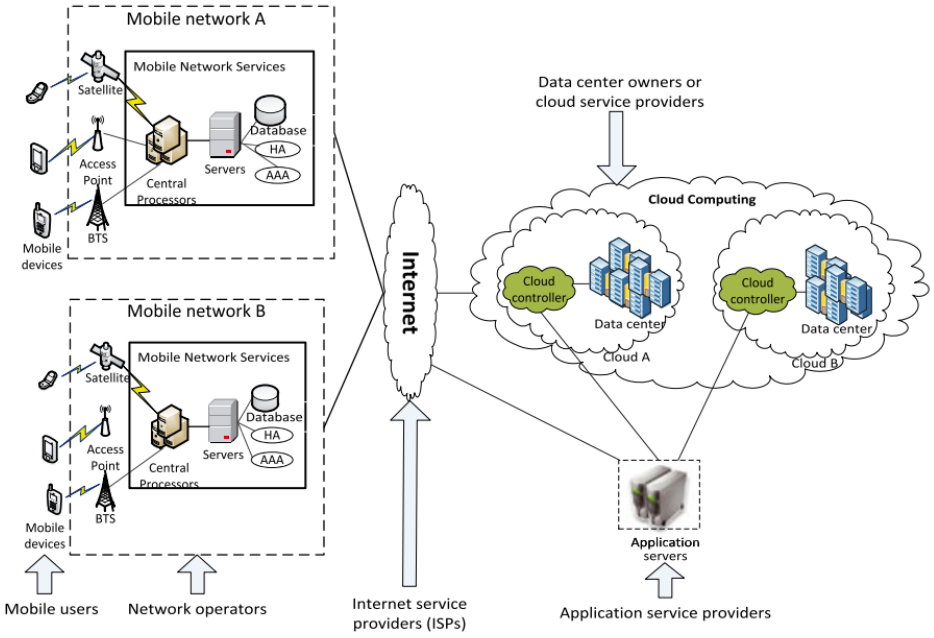

Figure 1: Mobile Cloud computing architecture (8).

servers with high speed networks, Iaas is built on top of data center layer, providing provisioning of storage, servers or networking components, while PaaS provides integrated environment for deploying applications. Many applications are being used in Software as a service mode, where user pays only for software used in cloud.

Issues discussed by authors in MCC space ((8) are very important to my research. Low bandwidth, availability of the service (in this case server side as well as other mobile client), were mentioned in ref. 1-12. Obstacle that this design must overcome is also heterogenity of mobile computing, as different mobile nodes access to the cloud through different radio technologies (GPRS, WLAN, WCDMA, among others), which causes concerns related to scalability, power efficiency or wireless connectivity. As proposed by Dinh *et al.*, there is need to intelligent context switching engine, which will qualify device eligibility to run on this service, in case of my research project, should the resources be offload to different device or not.

Another take on context-aware mobile cloud services has been described in architecture of Volare system by monitoring users device preferences and resources with provisioning of appropriate services for each user (9). Volare uses a middleware module embedded on mobile device to monitor resources and contexts, dynamically adjusting them based on user requirements (Figure 2) .

This approach can be used in this project to commit to decisions of offloading to another device through the cloud or continuing to work on current mobile phone.

2.4 Android architecture

My research project focuses on mobile devices, especially on Android operating system. In recent years Google has been trying to raise security bar for its devices or any

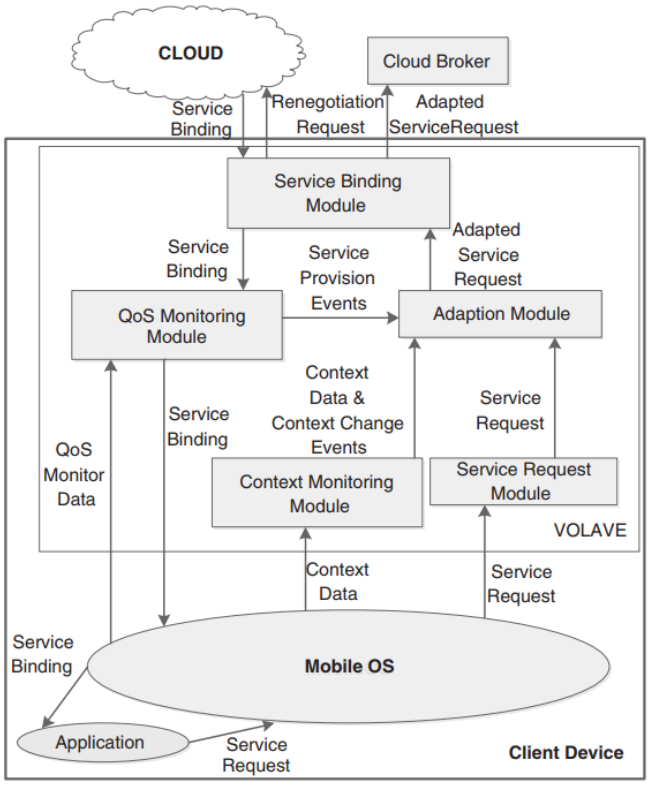

Figure 2: Volare context system architecture (9).

using Android OS with new OS updates, security patches and features. In Secure containers in android: the Samsung KNOX case Study, Kanonov and Wool (10) present a systematic assessment of security critical areas in design and implementation of secure containers in Android ecosystem. The relevance of this work is very important for this paper, as offloading system needs to take into consideration areas identified by authors as threatening, such as device being registered on untrusted networks, loss or theft of the device (physical breach) or malware that can infect the device. All these threats can be mitigated or reduced, by proper implementation of secure container on a device.

In that paper authors list different options of securing android device, by using mobile device management solution (MDM) to implement policies. It can be also achieved by root based approach via Deepdroid, which is not a go to solution as it requires device to be rooted. Finally software containers, like android enterprise (formerly android for work), which utilizes multiple user mechanism available in Android 5.0+ to allow isolation of work and personal content can be also used for this securing purpose. All listed solutions can be attacked and compromised by kernel security exploits. Answer for those exploits is obtained via software and hardware solution, incorporated in Samsung KNOX devices, which root of trust is hardware implementation of ARM TrustZone. This architecture will allow secure code offloading without data being compromised. Knox architecture consists of few components. First is SEAndroid, which is a fine grained security policy enforced by SELinux, isolating applications from each other. Second component is Trustzone-based Integrity Measurement Architecture (TIMA), which is an architecture design that contains all the security applications in the container. Finally Secure boot secures device while booting up, making sure that each step is validated starting from verification if

ROM has not been tampered with. Knox provides additional layer of encryption on top of standard Android OS and extensive VPN framework intended for enterprise applications. These KNOX features are important for this paper, as they can help with deployment and securing of any data that can be accessed or transferred from the device. As authors present, there is a danger of sharing KNOX services with user applications, even in the presence of dedicated security measures, therefore this implementation of offloading application needs to take into consideration security risks that are brought with mobile world and specially BYOD (Bring your own device) space.

2.5 MQTT Broker

For purpose of this project and offloading mechanism of proposed solution, there needs to be implemented lightweight data transport solution to support sending and receiving information between devices.

Based on research done by Thangavel *et al.* (11), bandwidth- efficient and energy-efficient Message Queue Telemetry Transport Protocol will fit in this scenario. MQTT is an application layer protocol, which has publish-subscribe architecture, allowing clients to publish messages to particular topic. Afterwards all clients which are subscribed to this particular topic will receive this message. This protocol is often used for resource constrained devices.

The authors compare MQTT protocol to Constrained Application Protocol (CoAP), which is newer and based on Representational State Transfer (REST) architecture and perform series of tests using common middleware. While MQTT uses TCP transport layer, has 3 Quality of service layers and publish-subscribe architecture, CoAP uses UDP layer, no quality of service layers and request-response architecture. Differences in layers do not show in results, as one protocol can outperform other one in different areas. Advantage of MQTT is seen in many use cases and developments for wireless distributed systems. An interesting idea is presented in Canderoid, which is a mobile system to remotely monitor travelling status of the elderly with dementia (12) . Canderoid is reliable, energy saving system, which can help caretakers to precisely and fully understand travelling status of an elderly person.

Figure 3 displays design of Broker introduced communication architecture, enabling bi-directional communication between smartphone client and caretaker platform. This deals with a problem of knowing IP address by caretaker, as broker dispatches messages to different peers. This approach is crucial for deployment of distributed system throughout different wireless networks.

Authors developed an application called "Wanderoid" for mobile devices, which made a device work as a group of sensors and record travelling status of the owner. This information was recorded and sent to the server via MQTT protocol to provide data to another client (caretaker operator) that receives message. Wanderoid architecture was tested for reliability using Android battery dog software showing only one connection break during all rounds of testing, which was recovered from in a second.

The ease of implementation of MQTT protocol for resource constrained device is further described in "MQTT-S A Publish/Subscribe Protocol For Wireless Sensor Networks" by Hunkeler *et al.* (13).

Hunkeler described, MQTT-S as an extension of the open MQTT protocol designed for Wireless sensor networks (WSN), designed in a way that can run on low power, battery operated sensors and can be used and operate over limited bandwidth wireless networks

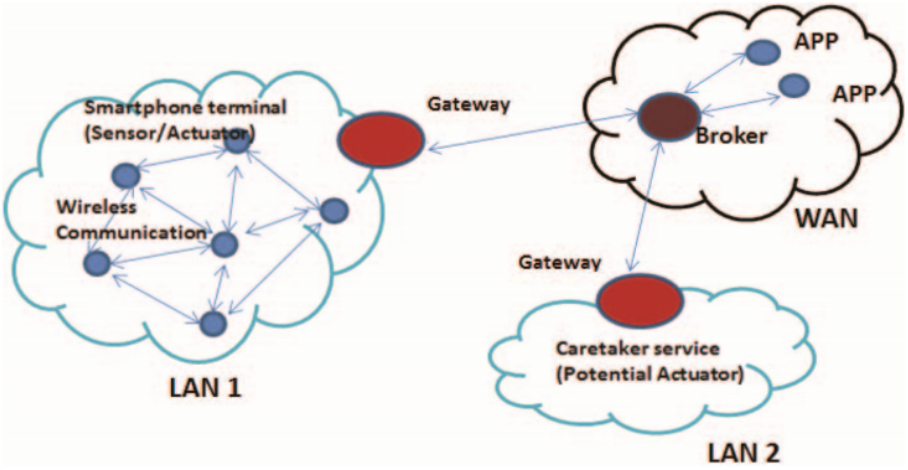

Figure 3: Canderoid Broker Introduced communication architecture(12).

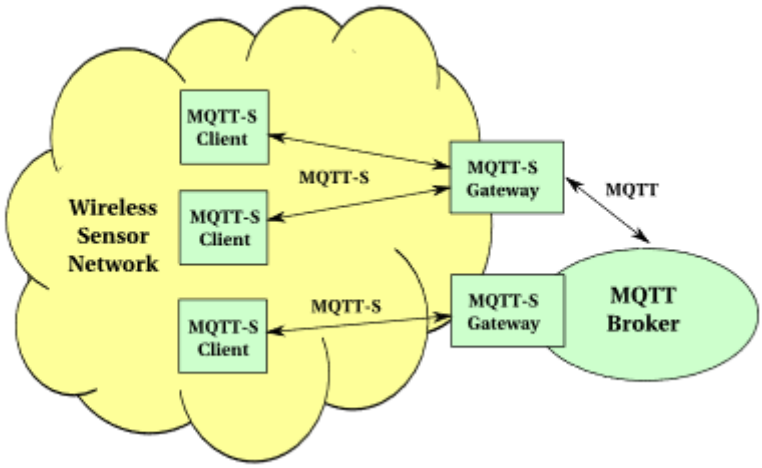

Figure 4: MQTT-S architecture(13).

(such as ZigBee). Figure 4 shows that MQTT-S architecture is similar to standard MQTT, seen in canderoid design, consists from broker, gateways and clients hosted in wireless network. In their research authors outline two different types of MQTT gateway, transparent, where for each connected client, a transparent GW sets up and maintains a MQTT connection and aggregating gateway, where there is only one gateway connection to the broker for all connected clients at once. Aggregating gateway is documented to be more helpful due to reduction in number of connection that broker has to support. The difference between MQTT and support of short message payload, (used with sensor networks by Zigbee client), is limited to 60 kb. This approach is not needed with that design as devices targeted by this research are more powerful and implementation of MQTT architecture may prove to be sufficient.

2.6 Current frameworks

As stated by Yuan Zhang *et al.* (14), current work on mobile offloading frameworks follows three approaches categorized by the granularity and partition algorithm being used; full migration, pre-calculated offline partitions and Making decision at runtime. Authors describe full migration as inflexible and not granular enough, used in early stages of research, while using pre calculated offline partitions might not simulate all offloading scenarios that can occur during real world execution.

Sample framework using pre calculated offline partitions is CloneCloud (15), however as stated by authors, real network and device conditions cannot be generalized into fixed amount of states, which means that using the pre-calculated partitions cannot cover all the offloading scenarios. Other popular frameworks used are Comet (16) and Tango (17) which suffer from issues where it is hard to replicate offloading tasks. Making decision at runtime examples of this approach are Odessa [9], MAUI [6] and Wishbone [7].

MobiCop, presented by Jose I Benedetto *et al.* (18), is an offloading platform that seeks to address the reproducibility issues of other offloading solutions by encapsulating all offloading logic in a library and offering compatibility with major IaaS providers is a newest addition to common offloading methods. It is designed to minimize execution time by invoking service application component. Developer needs to deploy custom android class and provide custom parameters to be marshalled by OS. Framework provides fully functional decision engine which predicts execution times based on previous runs. It is joined with network profiler, but is only supported if past records exist for previous executions. MobiCop's architecture consists of client library, covering android service that oversees handling of incoming tasks and connects platform components. It queries decision making engine to figure out execution context. Platform communication layer makes use of asynchronous data transfers to transfer messages via XMPP through Google's firebase cloud messaging system. Server component consists of four modules: public interface, middleware, an elastic cloud component and Android server environment. Developers are required to preinstall their applications apk into the server for the platform to be operational, which may slow the whole process down. API, application programming interface proposed by Benedetto *et al.* encapsulates offloaded tasks in runnable derived from superclass, providing utility methods for Accessing Android code.After defining class, users can declare input and apk to the platform, which provides them with id for future associated tasks. Results of these actions are being sent to local broadcast receiver. Decision Engine consists of two modules- QOS monitor and code profiler. Quality of service component takes care of profiling network- availability, latency and transfer

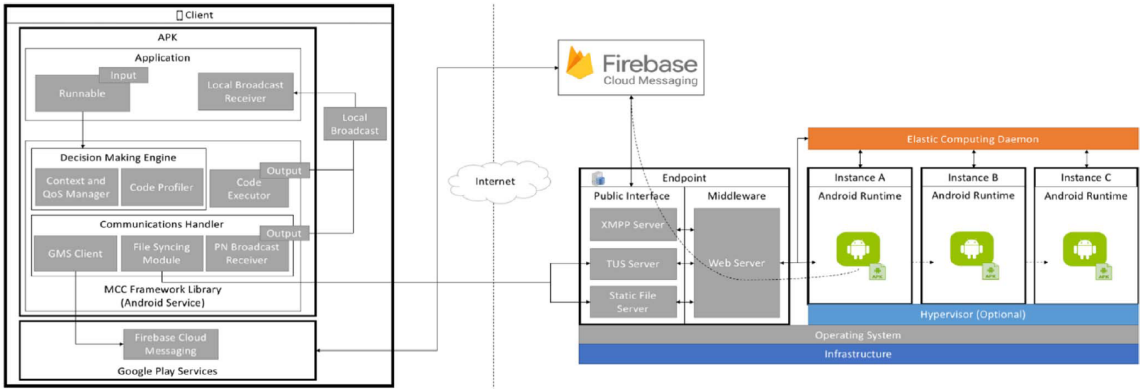

Figure 5: MobiCop architecture(18)

speed. Wifi connection speed is being sampled on connection to new network and cached for future use. This presents problem with changing network speed on different services. Network speed for mobile networks (LTE, GPRS, EDGE, etc.) is based on average transmission types. Code profiler keeps track of past task executions and predicting runtime of future tasks. In this paper there is emphasis on improving QoS part of a decision engine to incorporate analysis of hard drive space, battery left on a device and available memory and cpu. It also evaluates network currently being accessed during running of software and decision on offloading is being provided. This report will be focused on logic residing on mobile device and implementation of it using classes that can be deployed with any android apk files.

3 Methodology

3.1 Evaluation of CPU/ Memory usage on android devices.

To evaluate the CPU and Memory on android devices there need to be taken into consideration different specifications based on API availability and OS version. This research was concentrated on Android devices as iOS based devices would require also native code to analyze performance and can be taken into research in the future work. Additionally to evaluation of Memory and CPU methodology, incorporated in this research is usage of battery check, whether device is low on battery it will send process to the cloud. These three attributes help prove the concept of distribution and sending process from mobile device to the cloud. (Fig 6)

3.2 Setting threshold and limit on which action will be made.

In proposed software threshold is set to 20 percent of battery used. When triggered and battery is lower than proposed threshold, process is being sent to cloud via MQTT protocol and topic is distributed via cloud broker. Any associated clients are able to see topic of the message.

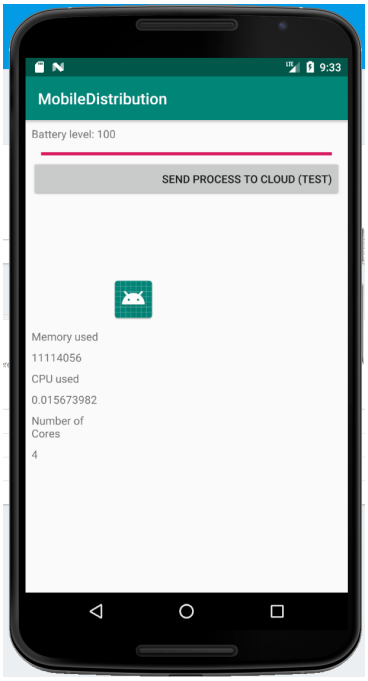

Figure 6: Nexus emulator with Android 7

3.3 Execution of the task on a device.

Task executed on a device shows animated android icon rotating consecutively during work phase of the program. When logic is met and process is sent via MQTT broker, animation stops spinning. Actual process of animating the image is not processed in the cloud in this project, as this will require further research and will be explored in my future work.

3.4 Commit decision if task is send to the cloud or is executed on a device.

For this research demo purpose, there has been added "send process to cloud(test)" button, which eliminates need to lower battery usage and bypasses this logic.

3.5 Sending task via MQTT service to the cloud.

When condition is met and battery limit reaches set threshold job is being stopped on a device and is being sent to cloud via MQTT protocol. MQTT broker connection is being established and information is being sent to another client subscribed to this thread. Fig(7)

3.6 Receiving confirmation from the cloud if the task was completed and get the result and time.

Proposed solution shows python client running on windows machine connected to cloud broker receiving information about device state from MQTT broker (Fig 9). If the

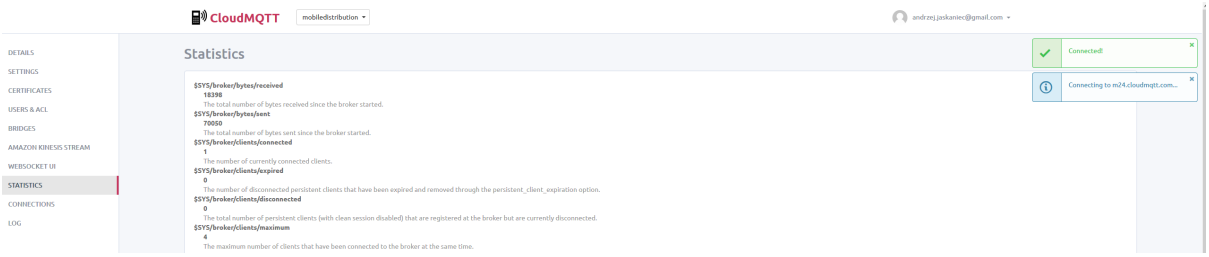

Figure 7: MQTT broker statistics information

When conditions are met with battery power being low and Wifi network accessible, proposed software stops the process and sends information about task to the broker 7. Information about task being sent is available on windows client connecting to the same topic via broker Fig(9).

```
2019-05-18 18:50:56: Saving in-memory database to /var/lib/mosquitto/13022/db.
2019-05-18 19:20:57: Saving in-memory database to /var/lib/mosquitto/13022/db.
2019-05-18 19:47:28: New connection from 109.255.139.121 on port 13022.
2019-05-18 19:47:28: New client connected from 109.255.139.121 as paho374778733940 (c1, k60, u'pflnyyaj').
2019-05-18 19:47:47: New connection from 109.255.139.121 on port 13022.
2019-05-18 19:47:47: New client connected from 109.255.139.121 as paho394297772600 (c1, k60, u'pflnyyaj').
2019-05-18 19:50:58: Saving in-memory database to /var/lib/mosquitto/13022/db.
2019-05-18 20:20:59: Saving in-memory database to /var/lib/mosquitto/13022/db.
```

Figure 8: MQTT broker connections established

battery level is under 20 percent, device will offload the process while informing clients via sending to subscribed devices message ”Battery level is under 20 percent”.

4 Design Specification

Solution for decision engine is an android app developed in Android studio, with MQTT and firebase implementation.

Design of proposed solution incorporates TelephonyManager class to establish what type of mobile service is available on a device. If connection is 2g, offloading scenario will

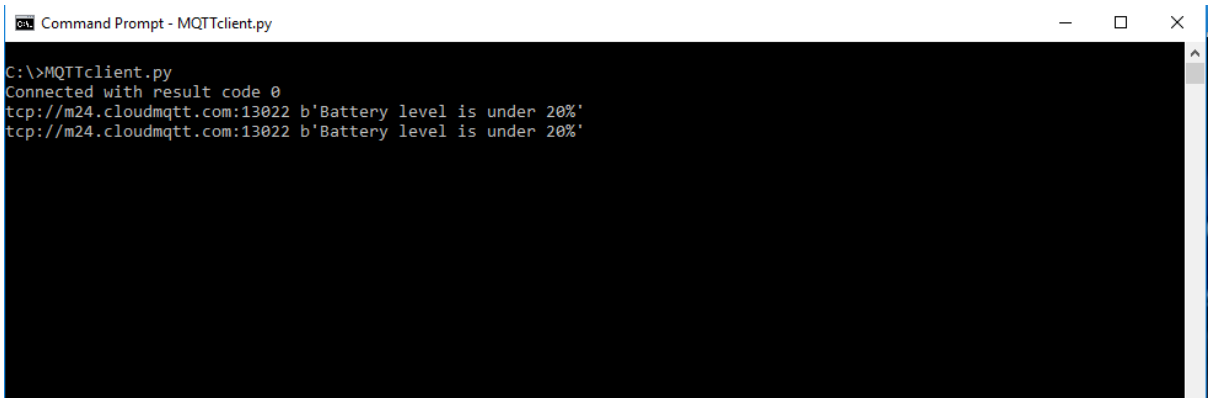

Figure 9: MQTT Client

```
//check what type of mobile network is used
public String getNetworkClass(Context context) {
    TelephonyManager mTelephonyManager = (TelephonyManager)
            context.getSystemService(Context.TELEPHONY_SERVICE);
    int networkType = mTelephonyManager.getNetworkType();
    switch (networkType) {
        case TelephonyManager.NETWORK_TYPE_GPRS:
        case TelephonyManager.NETWORK_TYPE_EDGE:
        case TelephonyManager.NETWORK_TYPE_CDMA:
        case TelephonyManager.NETWORK_TYPE_1xRTT:
        case TelephonyManager.NETWORK_TYPE_IDEN:
            return "2G";
        case TelephonyManager.NETWORK_TYPE_UMTS:
        case TelephonyManager.NETWORK_TYPE_EVDO_0:
        case TelephonyManager.NETWORK_TYPE_EVDO_A:
        case TelephonyManager.NETWORK_TYPE_HSDPA:
        case TelephonyManager.NETWORK_TYPE_HSUPA:
        case TelephonyManager.NETWORK_TYPE_HSPA:
        case TelephonyManager.NETWORK_TYPE_EVDO_B:
        case TelephonyManager.NETWORK_TYPE_EHRPD:
        case TelephonyManager.NETWORK_TYPE_HSPAP:
            return "3G";
        case TelephonyManager.NETWORK_TYPE_LTE:
            return "4G";
        default:
            return "Unknown";
    }
}
```

Figure 10: Types of mobile networks established using TelephonyManager.

not be established, while on 3g or 4g it will proceed.

Similar scenario is seen when connectivityManager class finds Wifi network connected, task can be offloaded to the cloud if meeting rest of criteria.

Functions GetCPUCores and ReadCpuusage establish number of Cores available on a device and overall CPU usage. Currently on android behaviour of these functions might change depending on API level (OS version) used. To read CPU usage access to "/proc/stat" file is needed, which from API 28 has been removed. In future work there needs to be other method evaluated.

While evaluation is completed process is being sent to cloud via MQTT protocol and stops run on a device. For the purpose of this project free version of CLOUDMQTT broker has been deployed to cloud.

5 Implementation

5.1 Description of technology used

This paper incorporates Java based services deployed in Android Studio for the purposes of evaluation of decision making engine sending offloaded process to the cloud. Java class implements broadcast receiver that checks for battery usage, wifi speed, memory and CPU usage during program run. When condition is met broadcaster send information to receiver to offload information to the cloud. For purposes of this paper there is implemented firebase plugin to monitor these statistics in a cloud as well showing overall runtime of specific processes and any failures encountered. 14 It is not a feature explored in this work, but monitoring code using firebase plugin might be interesting for the future work exploration. After committing to decision of offloading the task to the cloud, information is being passed to MQTT broker hosted at CloudMQTT portal using MQTT protocol. This information is passed on subscription topic that any client can connect to. In this research for picking up topic data is being used python client Fig(8).

5.2 Logic behind evaluation of data- reasons and thresholds for offloading processes.

Proposed software evaluates if battery is under 50 percent and makes a decision to send process to the cloud. When it is below limit, it checks for available network bandwidth to proceed with offloading scenario. If network is not sufficient, task might be stopped due to possibility of failure and not completing at all.

5.3 Explanation of service that monitors CPU/Memory on a device

Service uses Broadcast Receiver Java class to monitor CPU/Memory and battery activity. A broadcast receiver is an Android component which allows you to register for system or application events. All registered receivers for an event are notified by the Android runtime once this event happens. Current implementation monitors for changes in battery state, where if Battery is below 20 percent threshold, code is being offloaded to the cloud.

It also monitors for network that is being accessed, where offloading currently takes place only on WiFi, not 3g as it might be not reliable. Future plans are to evaluate bandwidth available on network to actively establish quality of connection available for current task.

Last monitor checks for CPU and Memory usage, but with barriers in establishing how much of the CPU current task is using, this is area for future work to explore.

6 Evaluation

6.1 CPU/Memory below threshold scenario (running process on a device)

When software is run it evaluates if the battery is in good condition to complete the process. It continues to run the process until battery reaches specified threshold. If the threshold is not met, process will be completed successfully on a device without offloading to the cloud. 12

6.2 CPU/Memory over threshold scenario (offloading to the cloud)

In proposed software, when battery reaches under 20 percent of life, service run in example is being stopped, as per animation, and information is being sent via MQTT protocol to the cloud. Network state is also evaluated, if not feasible and on low bandwidth network, process will not be offloaded. This solution also provides override of the specific scenario, which enables user to send process to cloud manually in Fig 13. Additional constraints are CPU and Memory utilization, which are analyzed in proposed software, but due to OS restrictions may need to be subject to future work for implementation on different Android versions as well as iOS and widnows implementations.

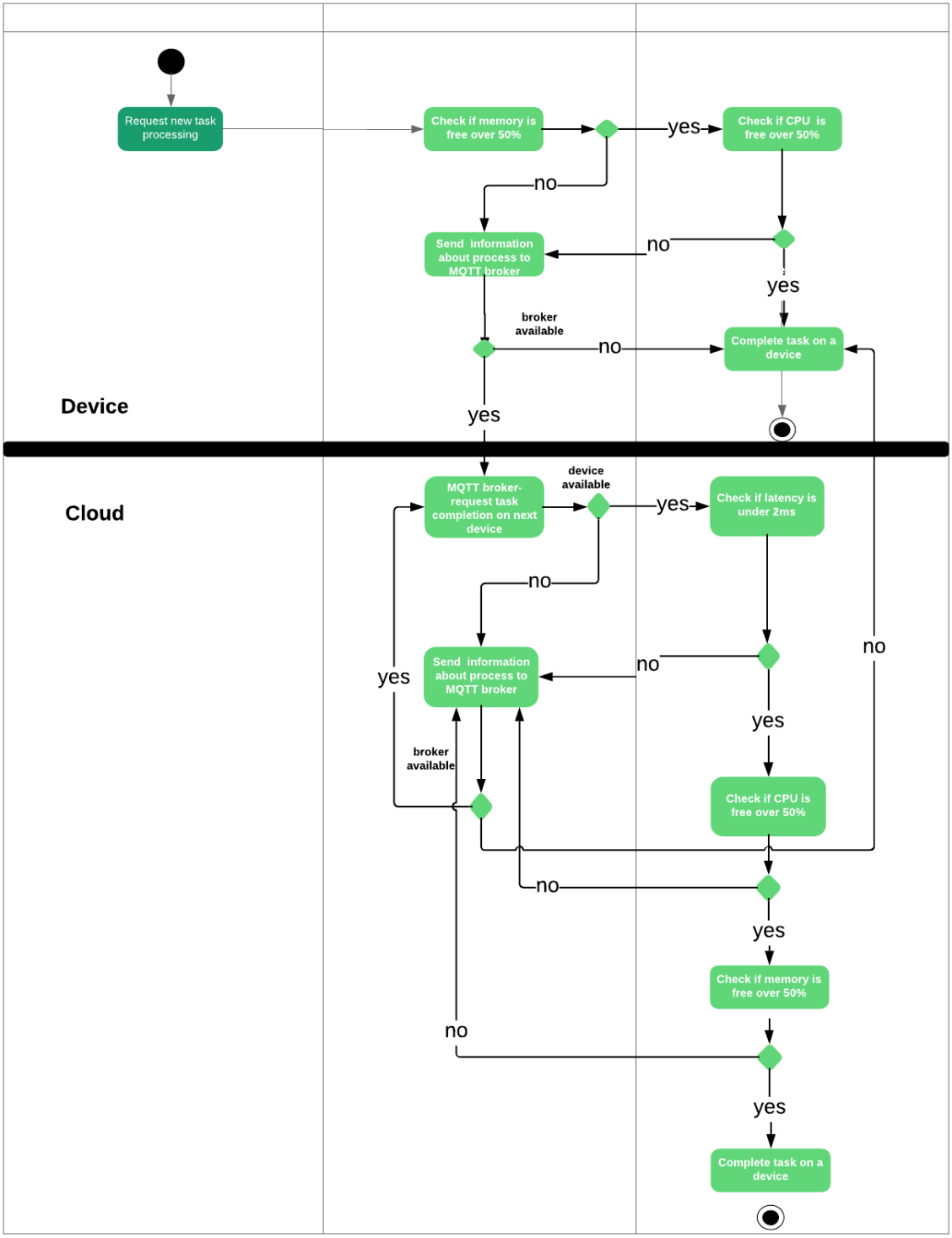

Figure 11: UML design of resources threshold analysis.

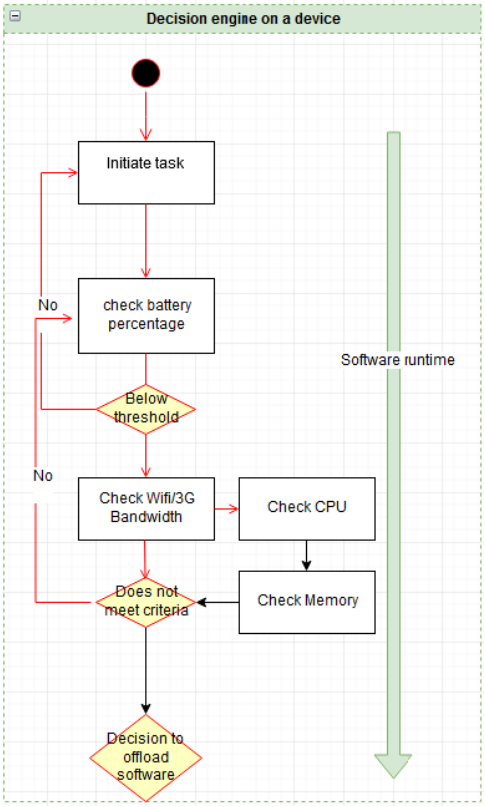

Figure 12: Current decision making engine on a device .

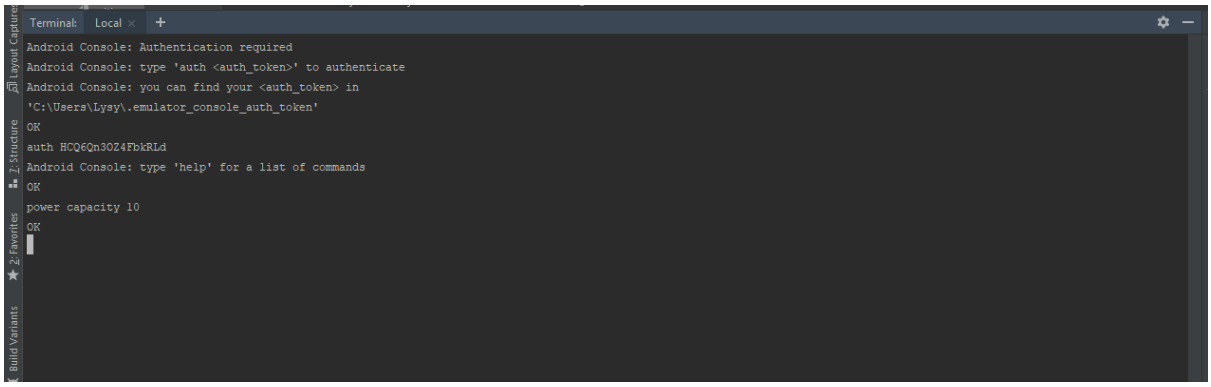

Figure 13: Power capacity set manually on android emulator (8).

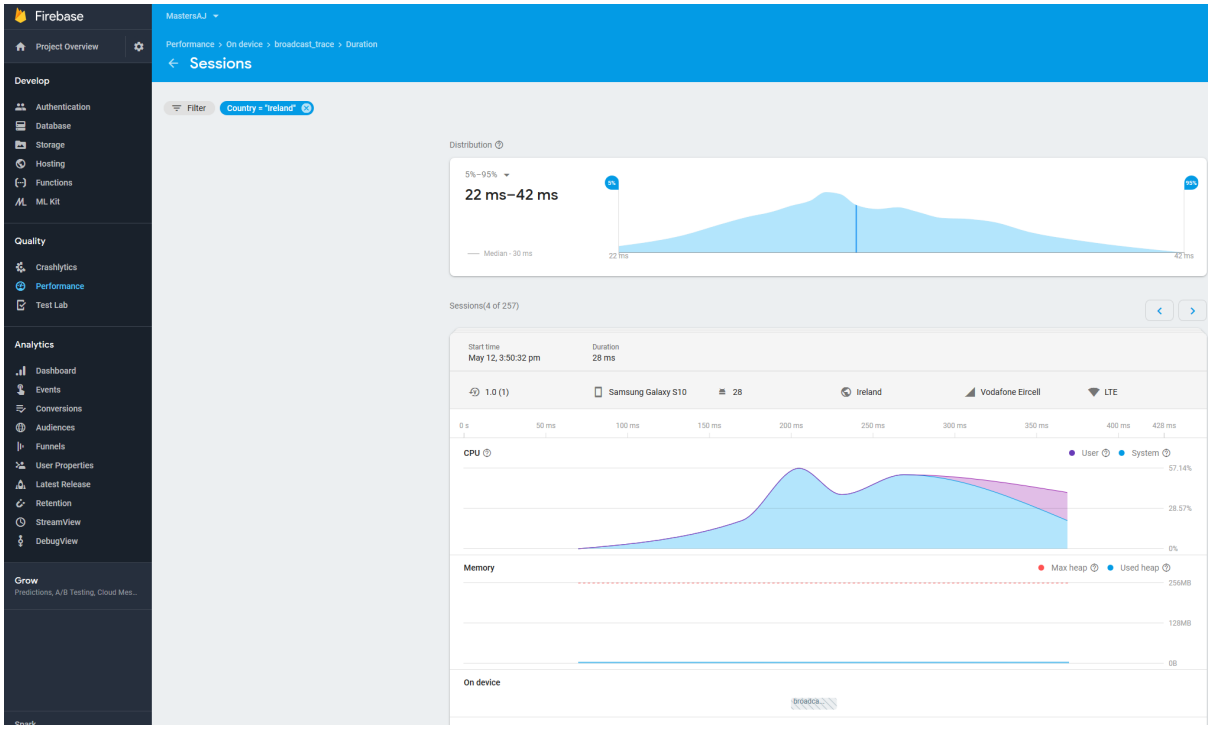

Figure 14: Firebase monitoring

6.3 Discussion

Decision making engine explored in this paper is a strong tool in computing real time hardware capability to offload specific tasks to the cloud. Running java code that can be simply incorporated in any android application makes it a very easy and familiar experience for any developers. Using Broadcast receiver, telephony manager, battery manager and connectivity manager classes, mobile devices performance and resource can be measured on demand and effectively utilized. Logic presented in this paper measures battery level, wifi/mobile data speed and availability as well as CPU and memory usage. It provides result when battery is low (under 20 percent) and connection is good, in the research only accepted offloading network is Wifi, however in the future work there might be added mechanism to offload using mobile networks as well, considering bandwidth and local tariff operators costs. Next scenarios are highly dependant on Operating systems or even devices. Measuring CPU usage based on proc/stats information is hard task on newer Android devices as from android 8 this functionality has been removed and the engine has to be deployed on android 7 devices to be fully functional. On evaluated device function can distinguish CPU used and also number of Cores, which can present specific information. Implementation of runtime calculation without incorporating prediction engine as described by Benedetto et al (4) comes with benefits but also some limitations. Engine presented in this paper is fast and does not require many additional changes during deployment of the app to the store, only addition of the classes needed for computation. Limitations are shown as code is needed to be changed for different android OS version and also extended to different systems not being tested (iOS, Windows).

7 Conclusion and Future Work

This research paper proposed mobile task offloading based on local computation using WiFi and battery availability. In described work it has been shown that decision making engine run locally on a device as a part of an application can be successful in establishing if task can be offloaded to the cloud. This computation has been done during runtime of the application using Java classes. It shows that computation based on Battery availability, mobile or wireless network access and CPU/Memory usage can be sufficient and fast for specific use cases. In comparison to other available frameworks, calculating if the process is offloaded dynamically during runtime reduces computation time and dependencies on other services, which may reduce errors. Future work will include standardization of the code between available OS versions, testing of secure containerization methodology for Android enterprise and KNOX as well as other Operating systems to fully support different mobile architectures.

References

[1] A. Merlo, M. Migliardi, and P. Fontanelli, "On energy-based profiling of malware in android," pp. 535–542, 2014. cited By 20.

[2] K. Kim, D. Shin, Q. Xie, Y. Wang, M. Pedram, and N. Chang, "Fepma: Fine-grained event-driven power meter for android smartphones based on device driver layer event monitoring," 2014. cited By 11.

[3] L. Zhang, B. Tiwana, R. Dick, Z. Qian, Z. Mao, Z. Wang, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," pp. 105–114, 2010. cited By 302.

[4] J. I. Benedetto, L. A. Gonzlez, P. Sanabria, A. Neyem, and J. Navn, "Towards a practical framework for code offloading in the internet of things," *Future Generation Computer Systems*, vol. 92, pp. 424 – 437, 2019.

[5] M. Aazam, S. Zeadally, and K. A. Harras, "Offloading in fog computing for iot: Review, enabling technologies, and research opportunities," *Future Generation Computer Systems*, vol. 87, pp. 278 – 289, 2018.

[6] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, and S. Yang, "Refactoring android java code for on-demand computation offloading," pp. 233–247, 2012. cited By 37.

[7] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A computation offloading framework for smartphones.," *Mobile Computing, Applications Services (9783642293351)*, p. 59, 2012.

[8] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611.

[9] P. Papakos, L. Capra, and D. S. Rosenblum, "Volare: context-aware adaptive cloud service discovery for mobile systems," in *Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware*, pp. 32–38, ACM, 2010.

[10] U. Kanonov and A. Wool, "Secure containers in android: The samsung knox case study," pp. 3–12, 2016. cited By 10.

[11] D. Thangavel, X. Ma, A. Valera, H. Tan, and C. K. Tan, "Performance evaluation of mqtt and coap via a common middleware," in *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pp. 1–6, April 2014.

[12] B. Xiao, M. Z. Asghar, T. Jms, and P. Pulii, ""canderoid": A mobile system to remotely monitor travelling status of the elderly with dementia," in *2013 International Joint Conference on Awareness Science and Technology Ubi-Media Computing (iCAST 2013 UMEDIA 2013)*, pp. 648–654, Nov 2013.

[13] U. Hunkeler, H. L. Truong, and A. J. Stanford-Clark, "Mqtt-s a publish/subscribe protocol for wireless sensor networks," *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, pp. 791–798, 2008.

[14] Yuan Zhang, Hao Liu, Lei Jiao, and Xiaoming Fu, "To offload or not to offload: An efficient code partition algorithm for mobile cloud computing," in *2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)*, pp. 80–86, Nov 2012.

[15] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, (New York, NY, USA), pp. 301–314, ACM, 2011.

[16] M. S. Gordon, D. Jamshidi, S. Mahlke, Z. Mao, and X. Chen, "Comet: code offload by migrating execution transparently," pp. 93–106, 10 2012.

[17] M. S. Gordon, D. K. Hong, P. M. Chen, J. Flinn, S. Mahlke, and Z. M. Mao, "Tango: Accelerating mobile applications through flip-flop replication," *GetMobile: Mobile Comp. and Comm.*, vol. 19, pp. 10–13, Dec. 2015.

[18] J. I. Benedetto, A. Neyem, J. Navon, and G. Valenzuela, "Rethinking the mobile code offloading paradigm: From concept to practice," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pp. 63–67, May 2017.