UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Bilal Abdullah

Analysis of Software Applications Computing Resources Usage on the Edge: A Case Study of Speech Recognition

Master's Thesis (30 ECTS)

Supervisor: Alo Peets, MSc

Tartu 2019

Analysis of Software Applications Computing Resources Usage on the Edge: A Case Study of Speech Recognition

Abstract:
Billions of sensors are currently deployed around the world. Some are standalone sensors while others can be found in smart-phones, wearables, cars, machinery, buildings, street lights, wind turbines and other places too numerous to mention. These sensors are connected to intermediate edge devices which provides connectivity to the core network. The amount of data generated by sensors is staggering and with the rapid growth of sensors deployed, generated data will only continue to increase. The traditional way of handling data where generated data is sent to the network for analysis and decision made is already inefficient and completely impractical in most applications. A better approach would be to perform these analytics and decision making tasks on the edge devices. But due to the very limited available resources on edge devices, it is important to first analyze the computing resource utilization of sample applications running on edge device in order to understand what computational tasks are possible on these edge devices. This thesis aims to take Speech Recognition as a case study and analyze its resource consumption on an edge device. The thesis further aims to explore the possibility of implementing long running tasks on the edge without significantly impacting the limited edge resources. Finally, we investigate the possible impact of performing additional speech analytics tasks on the edge.

Keywords:
Edge Devices, Sensors, Edge Computing Resources, Speech Recognition, Sensor Generated Data, Data Analytics, Application Resource Utilization, Application Resource Consumption, Applications on the Edge, Android.

CERCS: Computer science, numerical analysis, systems, control (P170)

Ääreanalüütika ressursikasutuse uuring kõne tekstiks muundamise näitel Android seadmes

Lühikokkuvõte:
Igal aastal võetakse kasutusee miljardeid uusi nutikaid seadmeid. Osad neist on klassikalised eraldiseisvad seadmed, kuid enamik on ühendatud internetti või integreeritud nutiseadmete, autode ja muude tarkade masinate sisse peitu. Enamik seadmeid on ühendatud lokaalsete juhtseadmetega ja läbi juhtseadmete internetiga. Andmete hulk mida andurid igal aastal toodavad kasvab pidevalt ja võib lähiajal tekitada võrguühendustes ummikuid. Klassikaline isoleeritud tark-andur ja kesksesse pilve andmete saatmine ei ole enam alati parim lahendus seega viimastel aastatel kogub populaarsust lokaalne anduri andmete töötlus enne kesksesse pilve andmete saatmist. Samas enamik seadmete riistvara jõudlus on piiratud ja sega tuleb hoolsalt jälgida töötlemiseks vajalike ressursside olemasolu või kasutust. Töös uuritakse kolme kõne tekstiks muundamise lahendust ja iga lahenduse mälu, protsessori, võrgu ja energia kasutust. Testimiseks kasutatakse Sphinx ja Google kõnetuvastuse teeke. Magistritöö tulemusena näidatakse, et Android seadmes on võimalik edukalt taustal teha aktiivset kõnetuvastust ja suurim ressursikasutus on umbes 5% akut tunnis.

Võtmesõnad:
Ääreanalüütika, andurid, hajusarvutus, kõne tuvastus, suurandmed, andmetöötlus, ressursikasutus, Android

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Contents

1 Introduction

1.1 Motivation

In 2018, a massive 2.5 quintillion bytes of data was generated every day [4] and with the rapid growth of interconnected devices with embedded sensors, generated data will only keep increasing exponentially. Sending this huge amount of generated data up to the cloud for analysis has major drawbacks. First, the current network infrastructure cannot handle this amount of data and even with the advert of 5G networks, generated data will eventually outgrow the available infrastructure. Secondly, the available cloud infrastructure to handle this quantity of data is finite. Added to this is the fact that a good percentage of sensor generated data includes noise which is not significant when there is no change in state. Thirdly, some applications require real-time data analysis and decision making which is difficult to achieve if the analysis is implemented in the cloud or remote network due to network latency. Finally, there is the issue of data privacy and handling of sensor generated data due to their private nature. For example, transmitting the data generated by a smart speaker placed in the bedroom or living room of an apartment to the cloud for analysis not only introduce privacy concerns but also exposes the data to potential data breach. Another case for moving some analytics task to the adge are lowered bandwidth usage and improved energy efficiency.

All these are issues that we hope to mitigate by implementing some analysis on the edge devices to filter the amount of data sent to the cloud and make some decision on the edge. As the major aim of edge analytics is to move data analytics close to the data source [1], this will not only reduce latency but lead to cost savings in network bandwidth and cloud storage. On the issue of data privacy particularly considering the fact that data collected from sensors could be very personal, processing the data locally on the edge ensures that it does not get into the wrong hands and the user can be sure that private data never leaves the device. Additionally, because the data is processed locally, chances of data security and privacy breaches are greatly reduced.

In order to explore the viability of moving data analytics task to the edge, there is need to first analyze the resource comsumption of sample applications running on edge devices as the available Power, CPU and Memory resource on an edge device is much limited. For this study, we settled on a Speed Recognition application running on a Samsung Galaxy S10+ smartphone. We selected speech recognition because of the rapid growth in voice interactive computing which can be found in everything from the voice assistant on our smart-phones to the smart speakers in our homes. These systems mostly depend on some remote server for speech recognition as these systems can only recognize simple commands locally although they are growing to support more sophisticated speech. In addition, due to the high computational and memory resources required for speech recognition, this makes for an ideal case study.

1.2 Goal and Problem Statement

The goal of this thesis is to analyze the computing resources used by a Speech Recognition application running on an Edge device. It is intended to carry out usage analysis of such resources as Power, CPU and Memory with the aim of identifying the usage of these resources by a Speech Recognition application running on an edge device. Additionally, we aim to explore the possibilities of constant background speech recognition on an edge device and further text analysis of the converted text.

To this end, we identify three research questions:

- What are the Power, CPU and Memory resource consumption of a speech recognition application running on an edge device?
- What is the impact on computing resources when running continous speech recognition?

We first carry out a literature review of related works in edge analytics in **Section 2** while in **Section 3** we examine the basic process of speech recognition and briefly discussed some speech recognition libraries while outlining the reasoning behind our choice of speech recognition libraries. Then in **Section 4** we lay out our experiment methodology including the test system, speech recognition application and experiment setup and in **Section 5** we discuss the application's achitecture and implementation. In **Section 6** we presented the Power, CPU and Memory resource consumption of running speech recognition application on an Android device, analyzed the extracted results and explored the possibility of running speech recognition in the background. Finally **Section 7** contains the conclusion where we discuss our findings and possible future directions.

2 Related Work

This section outlines some existing applications of analytics on edge devices.

2.1 Machine Learning on the Edge

Jianxin et al. [12] implemented the Zoo system which is a Composable Service that allows the user to pull and compose different basic Machine Learning(ML) services to form a more complex ML service. The basic idea stems from the fact that ML services are just composition of other ML services. The composed ML service can then be deployed to an edge device, the cloud, a combination of both edge device and cloud or to multiple edge devices. The system was tested by deploying an image classification service based on Google's InceptionV3 which consists of a neural-network that output a image vector class and a decoding service for ImageNet. Both services were then deployed to local Raspberry Pi devices with the option to deploy to the cloud. Automatic image tagging tasks were then ran. The system's performance was evaluated by comparing the computation tame against running similar tasks on Google ML with the results showing that the Zoo system archieved lower response latency.

2.2 Video Analytics on the Edge

Shanhe Yi et al. [11] detailed the implementation of Latency-Aware Video Edge Analytics (LAVEA) system, a Video Analytics Computing Serverless Architectural Platform running on the Edge, necessary due to the computational intensive and bandwidth hungry nature of video analysis. LAVEA is a 3-tier mobile-edge-cloud platform where the authors focused more on the mobile-edge and inter-edge design. Clients are allocated bandwidth and submit task to the platform then these tasks are then offloaded to the edge nodes. Sharing workload, managing queue priorities and scheduling tasks is automatically handled by the platform. The nodes run light computing tasks while using inter-edge collaboration between nearby edge nodes to speed up processing of tasks by offloading computationally intensive tasks to nearby edge nodes that are less busy. The functionality of the system was demostrated by building and deploying the platform on a network consisting of four (4) edge computing nodes and implementing an application for Automated License Plate Recognition (ALPR). The execution time of these tasks were then analyzed and compared against running similar tasks on the cloud which showed a 1.7x increase in processing times when processing tasks locally on the edge.

2.3 Augmented Reality on the Edge

In this paper, Marco et al. [10] presented the NEAR solution aimed at providing realtime AR features on lightweight IoT devices without any changes to the device or client application. NEAR, which is NFV at the Edge for transparent Augmented Reality, uses network function virtualization (NFV) to reprogram the network and transparently integrate functionalities on intermediate network nodes. The solution can be deployed could be deployed anywhere in the network using hardware devices with adequate computational power and made accessible via SOCKS proxy. The general operational flow includes retrieving the video stream, decoding, processing and providing the augmented video to the client. While testing, the authors reported AR acceleration gains compared to streaming the videos to remote servers.

While cases of video analytics, machine learning, data analysis and augmented reality on the edge abounds, there are only few cases of voice analysis on edge devices. This could be due to the complex nature of voice recognition on devices with limited computing power and this can also be seen in most of the voice recognition libraries that have to stream the audio to the cloud for recognition due the computing resources required. New advances in the speech recognition field had also been relatively slow although this is no more the case with the recent announcement by Google's of the All-Neural On-Device Speech Recognizer. The lack of voice analytics research and the complex nature of speech recognition with high computing resource requirements makes speech recognition an ideal case study in analyzing computing resource consumption.

3 Background

In this section we provide background information on the basics of Speech Recognition and discuss some Speech Recognition libraries with the reasons behind our choice of speech recognition libraries used in developing the Speech Recognition Android application.

3.1 Speech recognition

Speech recognition is a complex process and due to the lack of clear boundaries between words, speech to text translation is probabilistic and never 100% accurate. Speech is made of continuous audio stream with different states and simlar classes of sounds make up phones which in turn makes up words. The waveform of a phone can vary greatly which in turn makes phones vary from their representation due to various factors. In the actual speech recognition process, the sound waveform is first split by silence into parts and then each part is decoded into words. To decode the words, all possible combination of words is matched with the part and the best matching combination is selected.

In the actual speech recognition process, the sound waveform is first split by silence into parts and these parts are further splitted into frames of about 10 milliseconds in length. From the frame, a group of 39 numbers, called the feature vector, is extracted from the speech from to represent the speech. A speech model containing common attributes of the spoken word is necessary and for speech recognition, a generic model such as the Hidden Markov Model could be applied. For speech recognition, three(3) types of models are required. First is the **acoustic model** which contains the most probable feature vectors for each phone, secondly is the **phonetic dictionary** which maps words to phones and finally is the **language model** which mainly restricts the matching process to consider only probable words thus greatly reducing the search space complexity.

3.2 Speech Recognition Libraries

Here we outline four major speech recognition libraries with focus libraries that are accurate, offer online or offline speech recognition, have customization options, easy to integrate into existing application, support muliple programming languages and have up-to-date documentation. While Microsoft Bing Voice Recognition, IBM Watson Speech-to-Text and Google Cloud Speech-to-Text offer almost similar functionalities, we selected Google Speech-to-Text service due to recent Google's cutting edge advancements in the field of speech recognition, the option to perform offline speech recognition and the inbuilt availability of Speech-to-Text library in Android. Our second choice of the CMUSphinx library is due to the open-source nature of the project which

provides far more customization options and the focus on offline only speech recognition. Additionally, CMUSphinx Android integration is easy using the PocketSphinx library.

3.2.1 Microsoft Bing Voice Recognition

Microsoft Bing Voice Recognition [9] is a subscription based Microsoft product which is part of the Cognitive Speech Services offering realtime Speech-to-Text, Text-to-Speech and Translation services. The Speech-to-Text service can convert speech to text from audio files or streaming audio source and additionally identify participants in a conversation using the conversation transcription service. The recognition process is completely online as the audio source is sent to the cloud and the partial or complete text result returned as response. The service allows the user to customize or create new langauge models for used in the speech recognition process. In addition to a REST API for speech recognition, there are libraries in all major languages including Java(Android) with support for the service core the functionalities.

3.2.2 IBM Watson Speech-to-Text service

IBM Watson Speech to Text is a subscription based Watson service [] providing a realtime Speech-to-Text convension service with support for pre-recorded or streaming audio in various formats. There are options to recognize individual speakers, spot specified keywords and select a pre-built or customize the language or acoustic model used in the recognition process. The service is cloud based with HTTP REST, Websocket and Asynchronous HTTP programming interface for easy integration.

3.2.3 Google Cloud Speech-to-Text

Google Cloud Speech-to-Text [8] uses neural network models to convert speech to text with support for short or long form audio. The service can transcribe realtime streaming or pre-recorded audio with support for 120 language, automatically identify spoken language and identify individiual speakers in a conversation. The service offers the option to select specific pre-built speech recognition models based on the target use-case and context. The user can also specify words are phrases that are likely to be spoken in the speech. The Google Cloud Speech-to-Text service as the name implies is a cloud based service where the audio is first sent to the cloud for recognition and the text result sent back as respones.

Android Speech Recognizer in the android.speech package [7] provides access to the speech recognition service for Android applications using the core Google Cloud Speech-to-Text service. Due to the high consumption of battery and bandwidth, the recognizer is not suitable for continous speech recognition because of the need for stream-

ing of the recorded audio to remote servers to perform the actual speech recognition although the recognizer can also perform speech recognition locally using downloaded speech models.

3.2.4 CMUSphinx

CMUSphinx [2] is a an open-source offline speech recognition engine developed at Carnegie Mellon University and used for developing speech related applications with support for multiple programming languages such as C and Java. The suite contains tools for speech recognition and training acoustic models. There are extensive customization options including the option to specify the language and acoustic models in addition to making changes to the source code.

PocketSphinx Android [3] is a lightweight speech recognition library based on the CMUSphinx library and specifically tuned for smaller devices. It is written in Java and converts speech recordings into text using the CMUSphinx acoustic models with support for US English and some other languages. PocketSphinx depends on the SphinxBase library which provides common functionality across all CMUSphinx projects and can be deployed on Linux, Windows, on MacOS, iOS and Android platforms. For integration in the Speech Recognition Application, Android application, we include the Android Archive library containing binary files and independent Java code.

4 Experiment Methodology

Here we outlined the library, systems and configurations used in performing the experiments to evaluate the utilization of computational resources of a speech recognition application running on an edge device. The experiments were divided into two groups. The first group is an edge device running a speech recognition application that is completely offline and all the recognition is performed on the edge device while the second group is running a speech recognition application on the same device but this time speech is sent over the network to the cloud where the actual recognition is performed. Apart from analyzing the resource utilization of each group, they are compared against each other to better understand the trade-offs in resource utilization. Our main focus is on energy, memory and processing computation resources. Speech recognition accuracy was not a focus of this thesis, although we took that into consideration when comparing the results between the two groups.

4.1 System

The edge device used in the experiment was a Samsung Galaxy S10+ with a Exynos 9820 (8 nm) chipset, 2x2.31 GHz Cortex-A75 CPU and 8 GB RAM. Network communication on the device is provided by both a LTE cellular connectivity and a Wi-Fi 802.11 a/b/g/n/ac/ax wireless controller. We used Cellular for network connectivity during the experiment with wireless connection as a backup. The device was running version Android 9.0.

4.2 Application

In developing the speech recognition application, we used the open-source PocketSphinx Android library [3] developed at Carnegie Mellon University for local speech recognition on the edge device and Google Text-to-Speech [8] framework for cloud based speech recognition. While the Google Text-to-Speech framework offers little customization options, the PocketSphinx library is open-sourced and we could use custom dictionaries, acoustic and language models. But in keeping the experiment simple, we used the default dictionaries, acoustic and language models. The implementation of the aplication is discussed in **Section 5**.

4.3 Experiment setup

The speech recognition application was installed in the edge device, with battery percentage at above 50% and no other visible application running in the background and the speech recognition application was then ran on the device lasting 10 minutes where a sound recording containing a looping speech was played on a separate device. The

speech recognition are divided into two sets - offline and cloud-based speech recognition. We ran the cloud-based speech recognition using the Google Speech Recognizer as it supports supports both offline and cloud-based recognition, while for the offline speech recognition we ran it using both PocketSphinx and Google Speech Recognizer. We also ran a longer experiment lasting an hour in exploring the possibility of running speech recognition in the background.

4.4 Resource Consumption Measurement

The focus was in measuring the CPU, Memory and Energy resource consumption of the Speech Recognition Application. Here we outline each resource, terms and measurement metrics.

4.4.1 Energy

The device contains a Li-Ion 4100 mAh battery which we aim to measure actual energy consumption in mAh of the running application the energy consumption of the running application. This information is extracted by running the Dumpsys tool with the batterystats to generate a battery statistics report which is then saved to an output file. Additionally, we note the computed and actual energy drain which could be used in finding cases where the application's power consumption is inconsistent with the actual power drain.

4.4.2 Memory

The device contains 8 GB of RAM and we aim to measure the total memory allocation in Kilobytes of the application at any point in time. While Android Profiler contains a memory view where detailed information about an application's memory allocation is displayed, the same information could also be extracted using the Dumpsys tool with the meminfo option although here the information is not as detailed.

4.4.3 CPU

In measuring the CPU usage, the main focus is on the allocated CPU time in seconds. The information is extracted either using the Android Profiler or using the Dumpsys tool with the cpuinfo option.

4.5 Resource Consumption Measurement Tools

We identified two major ways of extracting information about resource consumption of a running application on Android. These tools are Dumpsys and Android Studio Profiler.

The Android Studio Profiler tool presents a high level of information collected in a single interface and the option to easily compare results, while Dumpsys tool provides us a more indepth information about resource usage although in a less readable format. So for more accurate results, we used both tools in collecting the resource comsumption data.

4.5.1 Dumpsys

Dumpsy [5] is an Android tool that is used in extracting information about system services which can be called from command line using the Android Debug Bridge (ADB) to get diagnostic report about services running on a connected device. The command can be customized to output information about specific system services such as RAM, battery, network, CPU and many more. Combining dumpsys with the meminfo, batterystats, procstats, cpuinfo and netstat options we can generate statistical data about the division of the application's memory between different types of RAM, battery, memory and cpu usage respectively. Using such tools as Battery Historian, which is a tool to visualize system and application level events, view aggregated statistics and compare reports, we can visaulize some of the generated data.

4.5.2 Android Profiler

Android Profiler [6] provides realtime data about an application CPU, Memory, Network and Battery usage. The tool by default shows a shared timeline view which shows the timeline graphs for CPU, Memory, Network and Energy. Detailed information about any of these can then be accessed by clicking on the corresponding graph. We can record and save profiler data to later compare the data between sessions. There are also options for advanced profiling where we can view detailed information about number of allocated objects, garbage collected events and details about files transmitted over the network.

5 Application Implementation

In this section we outline the architecture, implementation, functionalities and user interface of the Android Speech Recognition Application the aim of which is to measure the resource consumption of a Speech Recognition application running on an edge device which in this study is an Android smartphone. The Android Application was developed in Java using the Android Studio IDE with minimum target SDK 26 (Oreo 8.0.0) and target SDK 28 (Pie 9).

5.1 Architecture

The Speech Recognition Application implemented is made of the two major components which are the external speech recognition libraries discussed in **Section 3** used for speech recognition and the application component containing the speech recognition services.

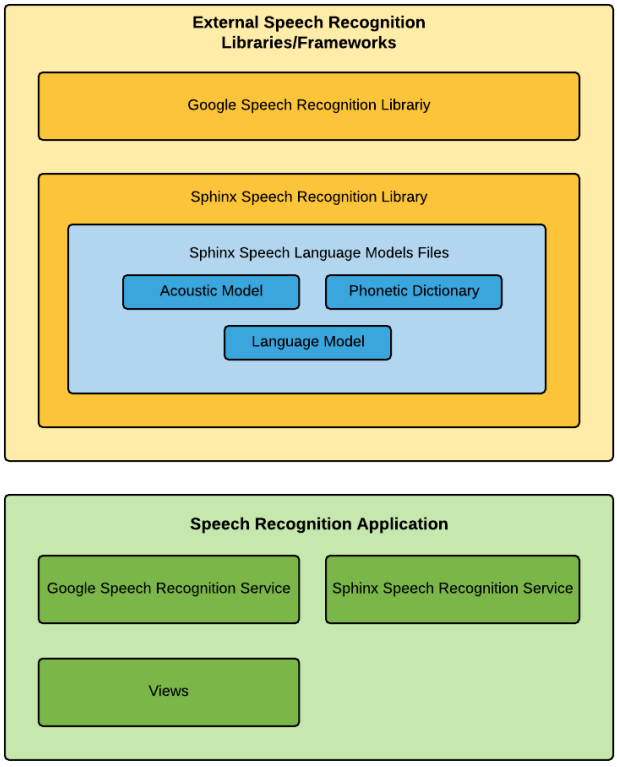

Figure 1. Application Architecture

The speech recognition application is an Android application writen in Java using the Android Studio IDE.

Google Speech Recognition library is used in the application for both online and offline speech recognition. The android.speech is already included in the android package by default so we do not need an additional step to access the library functionality. Using the library customization options, we customized the Speech Recognizer by setting the language model to en-US as this is our target language. We aim to recognize general speech so the language model is set to free-form while the minimum speech length is set to the maximum value as we need to recognize long utterances. The library does not have an option to set offline or cloud-based speech recognition. As a walkround to enable only offline speech recognition, we first ensure that the respective language file has been pre-downloaded on the device and then disable network connection on the device before running speech recognition application.

The PocketSphinx android library is used for offline speech recognition. The library which is distributed as Adroid Archive including both binary files and java code is added as a library and then added to the build.gradle file as a dependency. Language model files are required and these are included by copying the model files to the assets folder of the application while adding the gradle command to build the assets.xml file. The Speech Recognizer is created using the SpeechRecognizerSetup builder which also allows us to configure the main properties of the Speech Recognizer. we configured the language and dictionary Recognizer to use the language and dictionary models included in the app. To complete the setup, we added the digits and custom Grammar Search models.

5.2 Views

Launching the application the first time presents a permission dialog to allow the application to use the phone's microphone which is required for speech recognition and the application exits if the permission is denied. The main page of the application contains two buttons each for Google and Sphinx Speech Recognition respectively.

Clicking on any of the buttons takes you to the respective Speech Recognition page where there are options to select the language model, write the extracted text to file, run the speech recognition in the background and start the actual speech recognition. A running speech recognition can be stopped at anytime to change any of the configurations. The speech recognition is continous and restarts at end of each recognition sequence so the speech recognition could non-stop in the foreground or background.

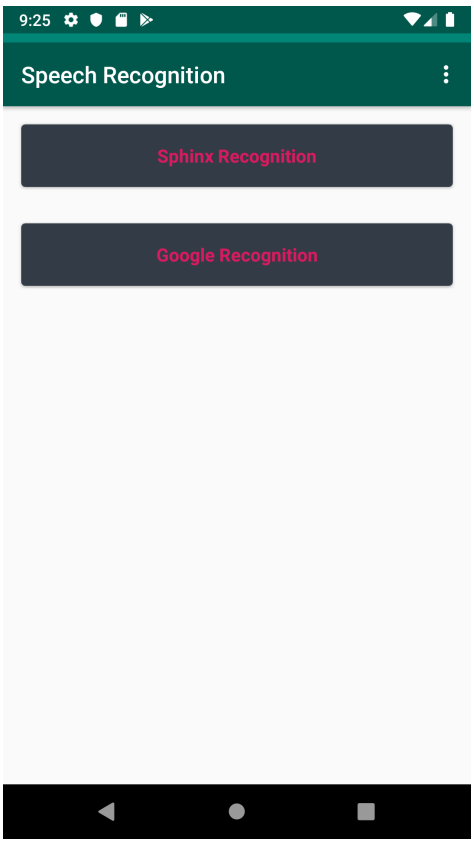

Figure 2. Application Main Page

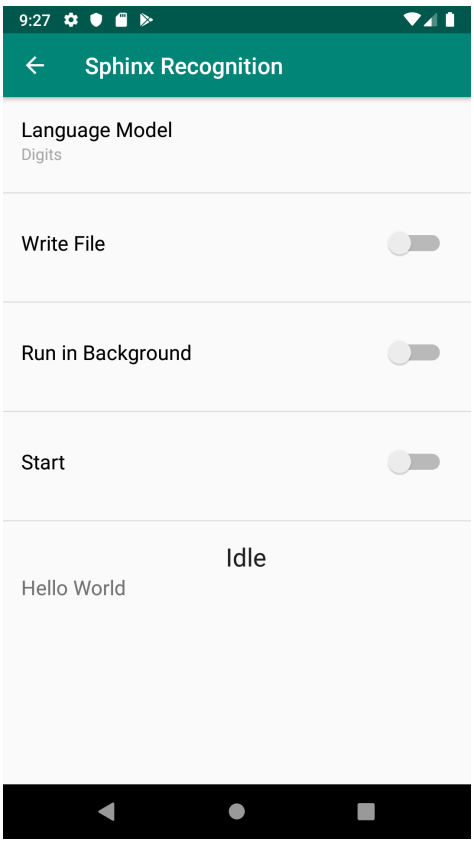

Figure 3. Application Sphinx Speech Recognition Page

6 Evaluation

This section presents the result of running the speech recognition application on the target device.

6.1 Results

6.1.1 Google Speech Recognition (Offline)

The results of running the Google Speech Recognition in offline mode are shown below. Results extracted using the Dumpsys tool are shown in Figures 4 while those extracted using the Android Studio Profiler Tool are shown in Figures 5, 6, 7 and 8.

```
Estimated power use (mAh):
  Capacity: 4000, Typical: 4100, Computed drain: 23.2,
  actual drain: 0-40.0
  Uid u0a332: 0.137 ( cpu=0.137 )
  Including smearing: 0.244 ( proportional=0.107 )
  Screen: 3.43 Excluded from smearing
  Idle: 2.37 Excluded from smearing
  Cell standby: 1.44 ( radio=1.44 ) Excluded from smearing
  u0a332:
    Wake lock *launch* realtime
    Foreground activities: 21s 856ms realtime (2 times)
    Top for: 21s 871ms
    Top Sleeping for: 10m 33s 425ms
    Cached for: 47ms
    Total running: 10m 55s 343ms
    Total cpu time: u=3s 860ms s=1s 80ms
    Proc ee.bilal.dev.speechrecorder:
      CPU: 3s 330ms usr + 1s 230ms krn ; 2s 160ms fg
      1 starts
Total PSS by process:
    36,945K: ee.bilal.dev.speechrecorder (pid 23218 / activities)
```

Figure 4. Google Speech Recognition Resource Usage (Dumpsys)

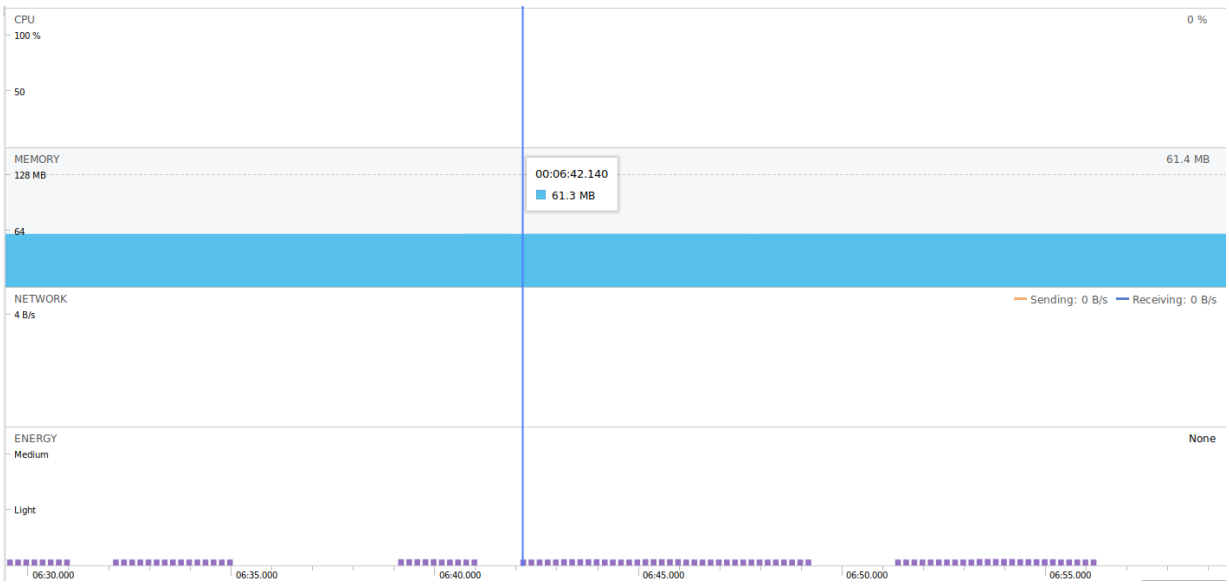

Figure 5. Google Speech Recognition Cosumption Overview (Profiler)

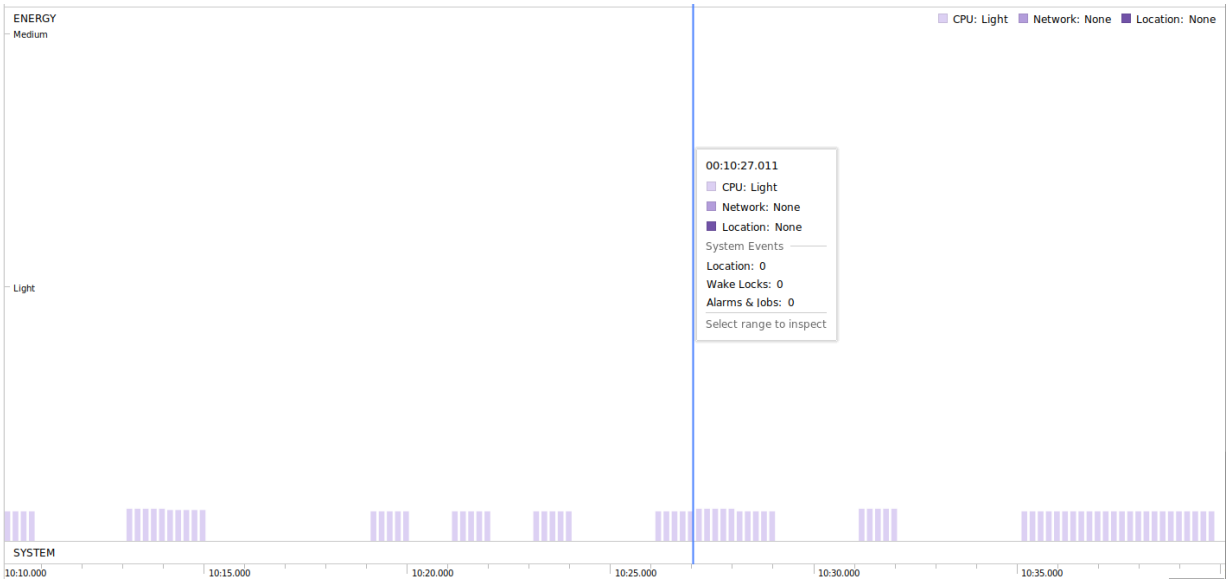

Figure 6. Google Speech Recognition Energy Cosumption (Profiler)

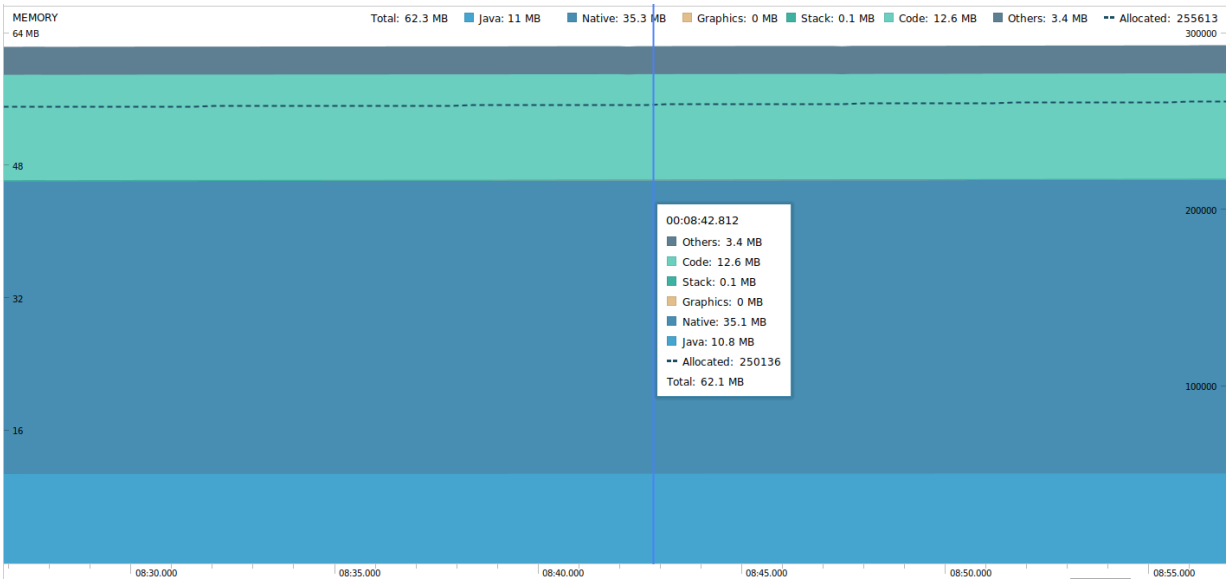

Figure 7. Google Speech Recognition Memory Usage (Profiler)

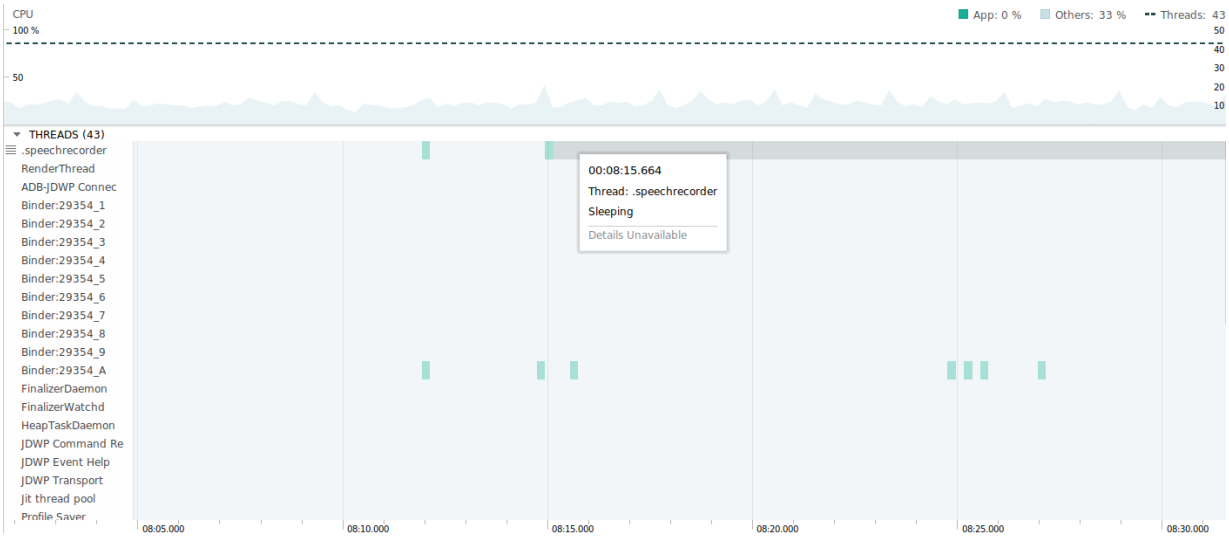

Figure 8. Google Speech Recognition CPU Usage (Profiler)

6.1.2 Sphinx Speech Recognition (Offline)

The results of running the Sphinx Speech Recognition (Offline) are shown below. Results extracted using the Dumpsys tool are shown in Figures 9 while those extracted using the Android Studio Profiler Tool are shown in Figures 10, 11, 12 and 13.

```
    Estimated power use (mAh):
      Capacity: 4000, Typical: 4100, Computed drain: 33.4,
      actual drain: 0
      Uid u0a332: 9.03 ( cpu=0.852 wake=2.28 )
      Including smearing: 11.8 ( proportional=2.82 )
      Screen: 5.58 Excluded from smearing
      Idle: 4.82 Excluded from smearing
      Cell standby: 2.45 ( radio=2.45 ) Excluded from smearing
  u0a332:
    Wake lock AudioIn: 6m 51s 27ms partial (8 times) max=545742
        actual=603064, 10m 3s 40ms background partial (8 times)
        max=545742 realtime
    Wake lock *launch* realtime
    Wake lock WindowManager: 11s 182ms full (6 times) realtime
    TOTAL wake: 11s 182ms full, 6m 51s 27ms blamed partial,
    10m 3s 64ms actual partial,
    10m 3s 40ms actual background partial realtime
    Audio: 10m 5s 930ms realtime (9 times)
    Foreground activities: 11s 916ms realtime (2 times)
    Top for: 11s 934ms
    Background for: 12s 131ms
    Top Sleeping for: 9m 51s 482ms
    Cached for: 70ms
    Total running: 10m 15s 617ms
    Total cpu time: u=26s 574ms s=25s 977ms
    Proc *wakelock*:
      CPU: 23s 262ms usr + 25s 505ms krn ; 0ms fg
    Proc ee.bilal.dev.speechrecorder:
      CPU: 3s 70ms usr + 310ms krn ; 1s 630ms fg
      1 starts
Total PSS by process:
    61,835K: ee.bilal.dev.speechrecorder (pid 29811 / activities)
```

Figure 9. Sphinx Speech Recognition Resource Usage (Dumpsys)

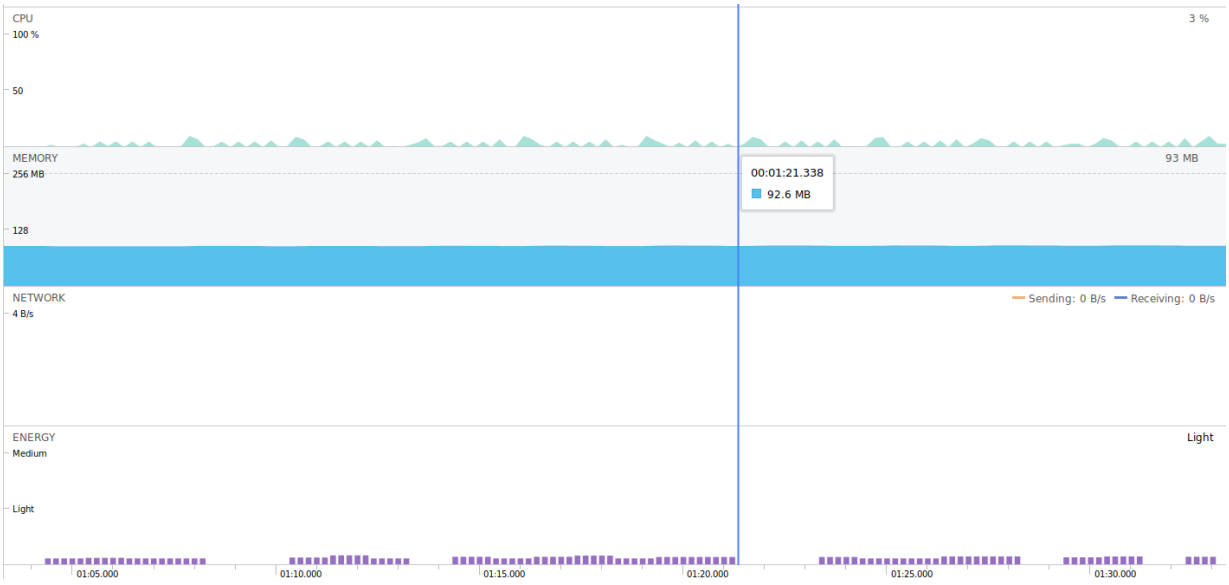

Figure 10. Sphinx Speech Recognition Resource Cosumption Overview (Profiler)

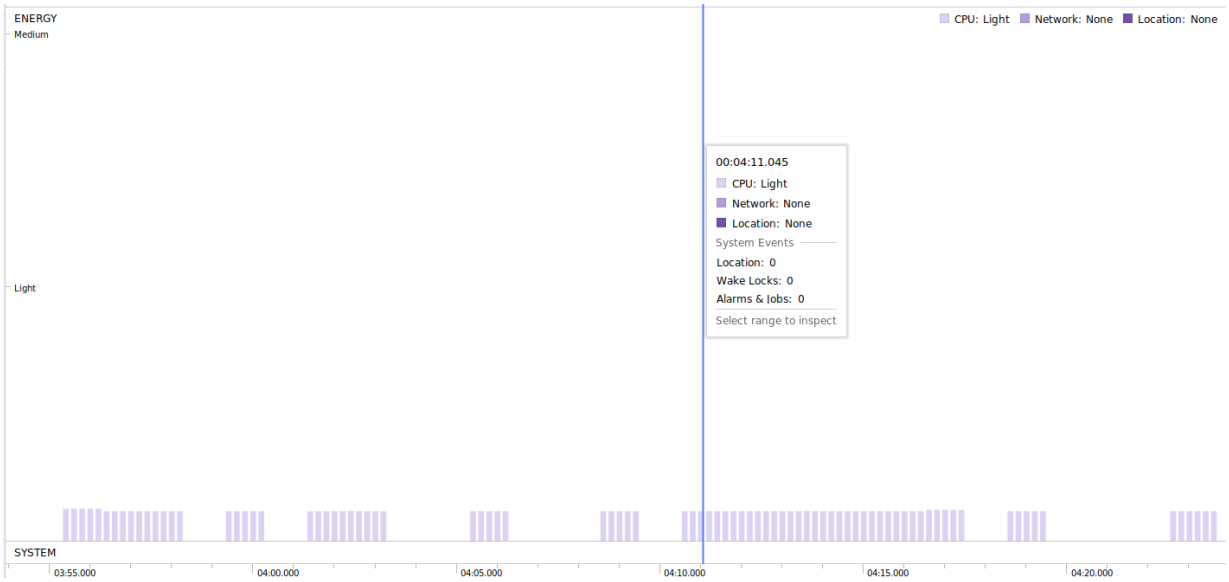

Figure 11. Sphinx Speech Recognition Energy Cosumption (Profiler)

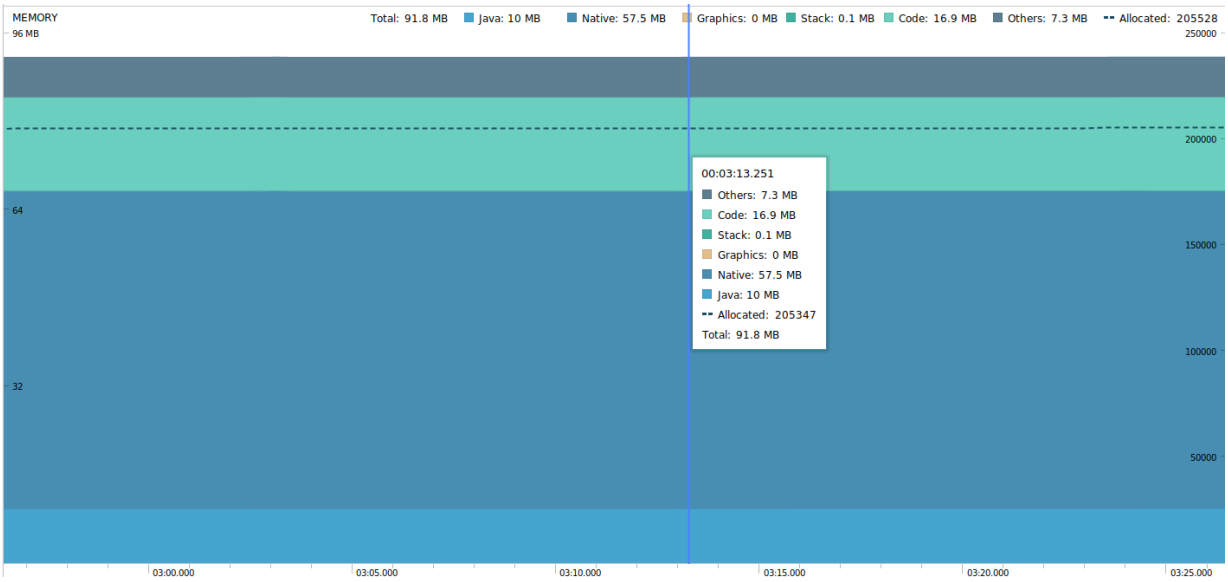

Figure 12. Sphinx Speech Recognition Memory Usage (Profiler)

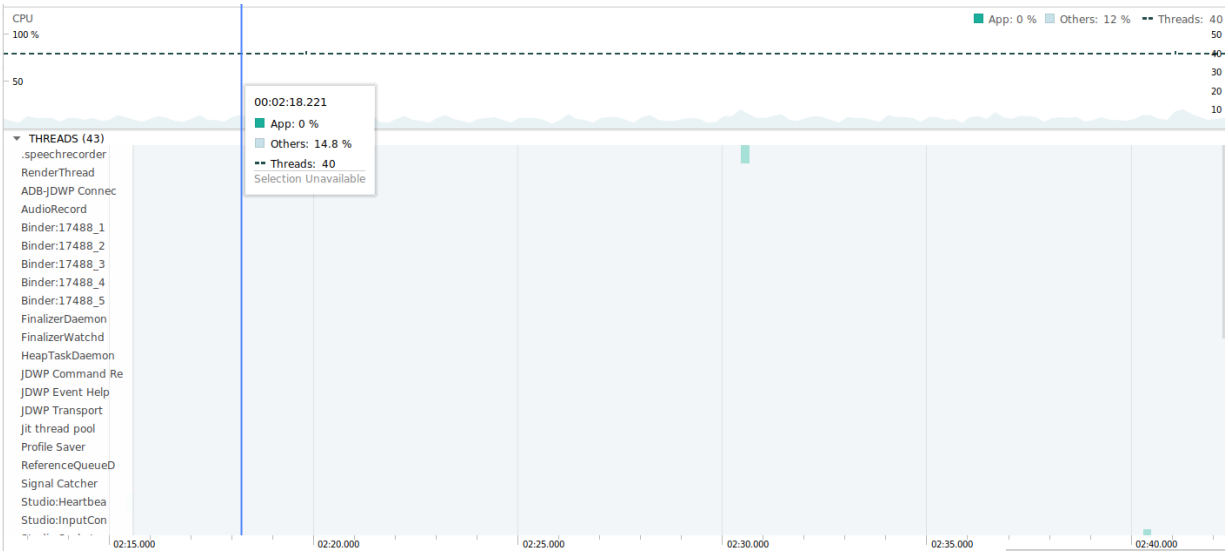

Figure 13. Sphinx Speech Recognition CPU Usage (Profiler)

6.1.3 Google Speech Recognition (cloud)

The results of running the Google Speech Recognition using cloud services are shown below. Results extracted using the Dumpsys tool are shown in Figures 14 while those extracted using the Android Studio Profiler Tool are shown in Figures 15, 16, 17 and 18.

```
Estimated power use (mAh):
  Capacity: 4000, Typical: 4100, Computed drain: 70.6,
  actual drain: 0-40.0
  Uid u0a332: 0.169 ( cpu=0.169 )
  Including smearing: 0.203 ( proportional=0.0337 )
  Idle: 4.37 Excluded from smearing
  Screen: 2.10 Excluded from smearing
  Cell standby: 1.38 ( radio=1.38 ) Excluded from smearing
   u0a332:
     Wake lock *launch* realtime
     Foreground activities: 21s 164ms realtime (4 times)
     Top for: 21s 215ms
     Top Sleeping for: 9m 59s 933ms
     Cached for: 1s 180ms
     Total running: 10m 22s 328ms
     Total cpu time: u=4s 947ms s=2s 134ms
     Proc ee.bilal.dev.speechrecorder:
       CPU: 6s 700ms usr + 3s 310ms krn ; 1s 210ms fg
       1 starts
Total PSS by process:
    34,744K: ee.bilal.dev.speechrecorder (pid 31012 / activities)
```

Figure 14. Google Speech Recognition Resource Usage (Dumpsys)

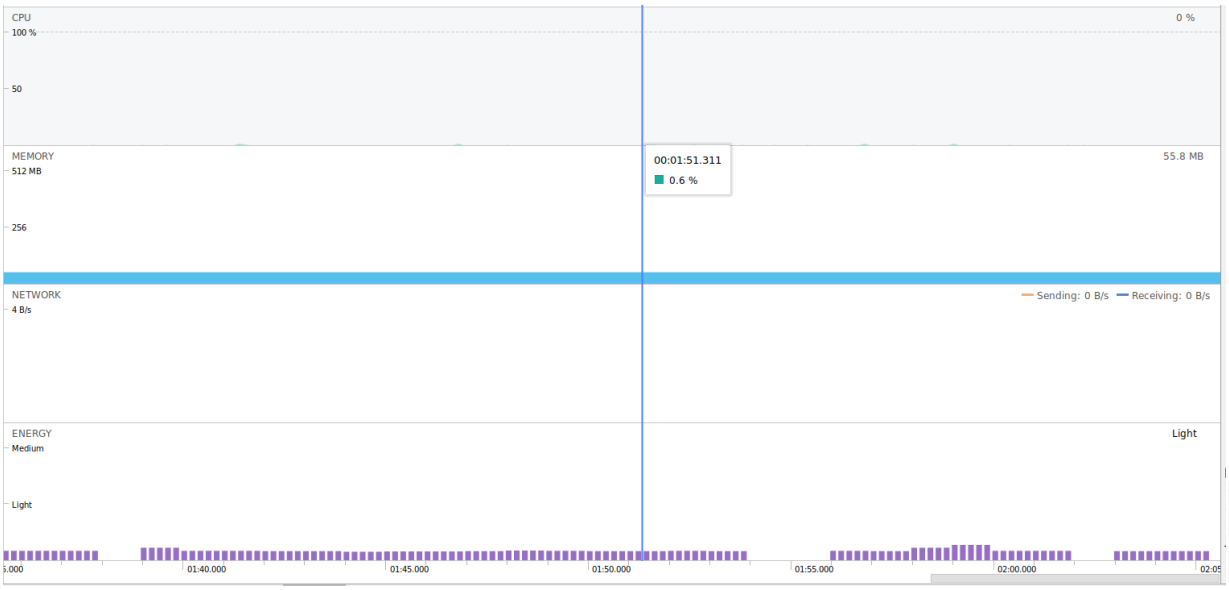

Figure 15. Google Speech Recognition Cosumption Overview (Profiler)

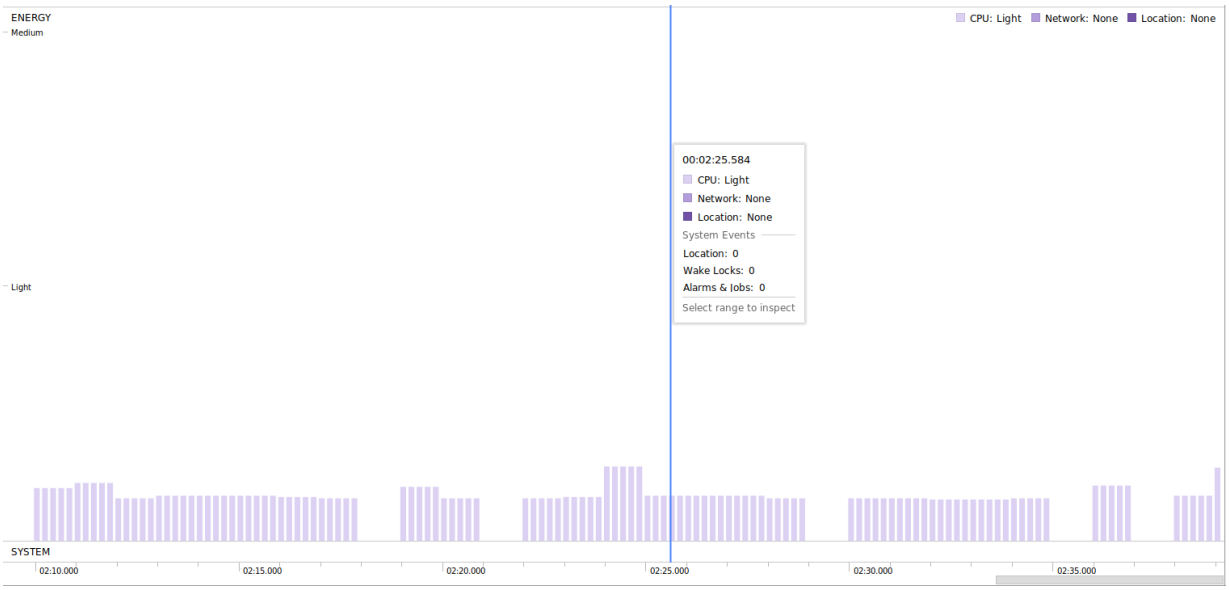

Figure 16. Google Speech Recognition Energy Cosumption (Profiler)

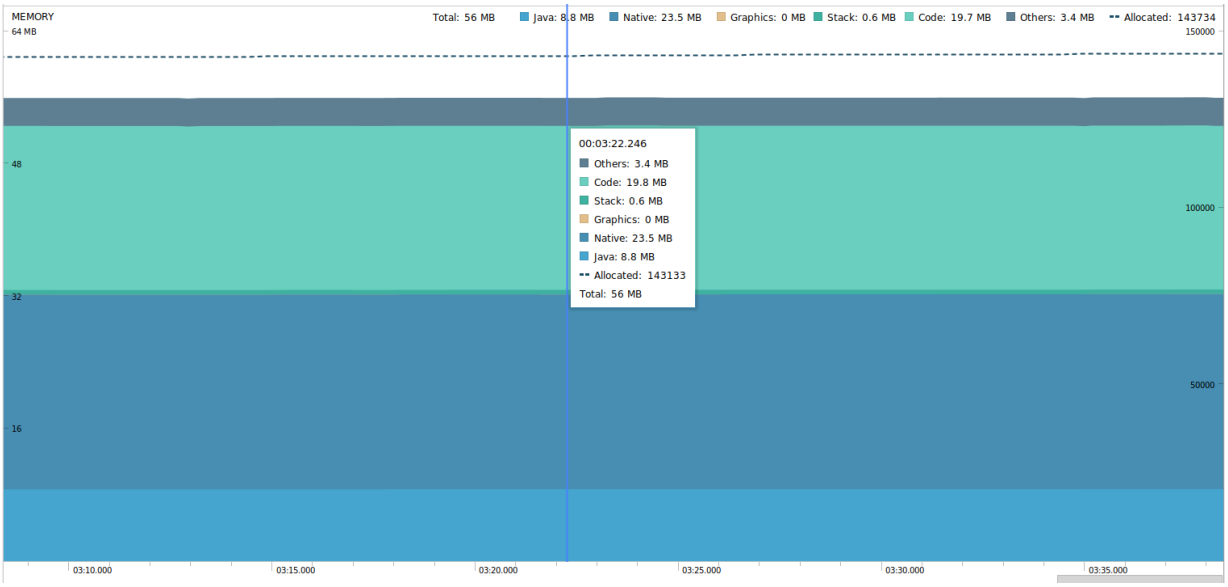

Figure 17. Google Speech Recognition Memory Usage (Profiler)

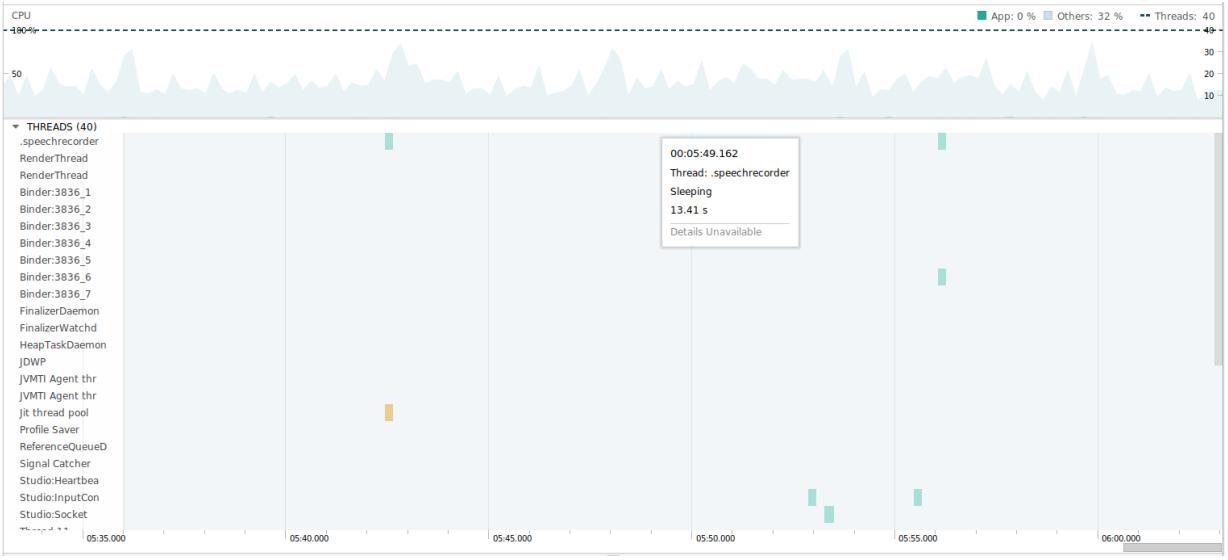

Figure 18. Google Speech Recognition CPU Usage (Profiler)

6.2 Long Running Speech Recognition Results

Here we present the results of running Offline and Cloud based Speech Recognition for an hour to represent cases of long running background speech recognition. The focus here is on the battery and memory usage so only the results obtained using the Dumpsys tool is presented here.

```
Estimated power use (mAh):
  Capacity: 4000, Typical: 4100, Computed drain: 184,
  actual drain: 120-160
  Uid u0a332: 0.694 ( cpu=0.694 )
  Including smearing: 0.925 ( proportional=0.231 )
  Idle: 26.6 Excluded from smearing
  Cell standby: 8.19 ( radio=8.19 ) Excluded from smearing
  Screen: 3.89 Excluded from smearing
   u0a332:
     Wake lock *launch* realtime
     Foreground activities: 45s 125ms realtime (3 times)
     Top for: 45s 202ms
     Top Sleeping for: 1h 2m 51s 978ms
     Cached for: 1s 282ms
     Total running: 1h 3m 38s 462ms
     Total cpu time: u=23s 105ms s=14s 471ms
     Proc ee.bilal.dev.speechrecorder:
       CPU: 38s 450ms usr + 25s 850ms krn ; 2s 480ms fg
       1 starts
Total PSS by process:
  42,198K: ee.bilal.dev.speechrecorder (pid 26634 / activities)
```

Figure 19. Long Running Google Speech Recognition Resource Usage (Offline)

```
Estimated power use (mAh):
  Capacity: 4000, Typical: 4100, Computed drain: 142,
  actual drain: 80.0-120
  Uid u0a332: 61.0 ( cpu=6.92 wake=19.3 )
  Including smearing: 75.1 ( proportional=14.1 )
  Idle: 24.9 Excluded from smearing
  Cell standby: 11.2 ( radio=11.2 ) Excluded from smearing
  Screen: 4.16 Excluded from smearing
u0a332:
  Wake lock AudioIn: 58m 0s 410ms partial (11 times) max=1192855
      actual=3524325 (running for 0ms), 58m 44s 309ms
      background partial (11 times)
      max=1192855 (running for 0ms) realtime
  Wake lock *launch* realtime
  Wake lock WindowManager: 7s 779ms full (4 times) realtime
  Wake lock Window:Toast: 44ms draw (1 times) realtime
  TOTAL wake: 7s 779ms full, 58m 0s 410ms blamed partial,
      58m 44s 325ms actual partial,
      58m 44s 309ms actual background partial, 44ms draw realtime
  Audio: 59m 30s 378ms realtime (15 times) (running)
  Foreground activities: 14s 319ms realtime (2 times)
  Top for: 14s 335ms
  Background for: 8s 85ms
  Top Sleeping for: 59m 20s 3ms
  Cached for: 60ms
  Total running: 59m 42s 483ms
  Total cpu time: u=3m 13s 458ms s=3m 41s 866ms
  Proc *wakelock*:
    CPU: 3m 10s 464ms usr + 3m 41s 348ms krn ; 0ms fg
  Proc ee.bilal.dev.speechrecorder:
    CPU: 2s 920ms usr + 400ms krn ; 2s 120ms fg
    1 starts
Total PSS by process:
    62,131K: ee.bilal.dev.speechrecorder (pid 26881 / activities)
```

Figure 20. Long Running Sphinx Speech Recognition Resource Power Usage

```
Estimated power use (mAh):
  Capacity: 4000, Typical: 4100, Computed drain: 407,
  actual drain: 360-400
  Uid u0a332: 0.603 ( cpu=0.603 )
  Including smearing: 0.712 ( proportional=0.110 )
  Idle: 26.2 Excluded from smearing
  Cell standby: 8.33 ( radio=8.33 ) Excluded from smearing
  Screen: 3.82 Excluded from smearing
  u0a332:
    Wake lock *launch* realtime
    Foreground activities: 13s 600ms realtime (2 times)
    Top for: 13s 620ms
    Top Sleeping for: 1h 1m 56s 265ms
    Cached for: 41ms
    Total running: 1h 2m 9s 926ms
    Total cpu time: u=19s 118ms s=11s 814ms
    Proc ee.bilal.dev.speechrecorder:
      CPU: 34s 640ms usr + 22s 130ms krn ; 1s 100ms fg
      1 starts
Total PSS by process:
    36,338K: ee.bilal.dev.speechrecorder (pid 10079 / activities)
```

Figure 21. Long Running Google Speech Recognition Resource Power Usage (Cloud)

6.3 Results Analysis

6.3.1 Experiment Results

From running the Sphinx and Google Speech Recognition, we were able to extract estimated Memory, CPU and Battery usage of the application running on our target device. Of most importance to us is the battery usage due to the increased storage capacity and computing power of edge devices compared to the minimal increase over the years in battery capacity of edge devices. Additionally, looking at the exttracted resources consumption results shows relatively insignificant CPU and Memory consumption which are also almost constant throughout the application's lifetime unlike the power consumption which is higher the longer the application runs.

Table of the resource consumption results obtained from the three experiments is presented below:

Name	**Battery Usage** (mAh)	**CPU** (s)	**Memory** (K)	**Time Running** (m)
Sphinx (Offline)	9.03	26.6	61,835	10
Google (Offline)	0.137	3.3	36,945	11
Google (Cloud)	0.169	6.7	34,744	10

Table 1. Resource Consumption

From the extracted results, we obtained far lower CPU, Memory and Battery usage when using the Google Speech Recognition library (Offline) compared to the Sphinx Recognition library. In Memory usage, while the Sphinx Speech Recognition used 61,835K of memory, with the Google Speech Recognition libray (Offline), the memory usage was 36,945K which is about 40% less than the Sphnix Speech Recognition library memory usage. We see the same situation in the CPU usage where the Sphinx Speech Recognition library recorded a total CPU time of 26.6s, while for the Google Speech Recognition the CPU time was 3.3s which is 12.4% of the CPU time used by the Sphinx Speech Recognition library. The battery usage follows the same pattern. The Sphinx Speech Recognition library recorded a battery usage of 9.03mAh for a total running time of 10.2m while that of the Google Speech Recognition (Offline) was 0.137mAh with total running time of 10.6m which is just 1.52% of the power used by the Sphinx Speech Recognition library although this is to be expected considering the higher CPU time of the Sphinx Speech library. We see that both offline and cloud-based Google Speech Recognition have similar resource consumption values with the highest difference in the CPU usage.

While the low resource consumption of the Google Speech Recognition library is impressive more so considering the fact that while the Sphinx Speech Recognition

library had only basic speech models with limited vocabulary the Google Speech Recognition library had full speech models with far wider vocabulary support, taking into account only the resource usage of our running application could be deceiving. When we ran the Google Speech Recognition for a longer period of an hour, we discovered that while the Speech Recognition application used very little Power and CPU resources, the Google Speech Recognition library made use of core Google Services which were actually much more resource hungry in terms of power and CPU time. This fact can be deduced by looking at the raw output data of the Dumpsys batterystats tool where we observed that the computed drain was much higher for a situation where only insignificant power was drawn. The long running Google Speech Recognition (Cloud) reported an actual power drain of 360-400mAh and looking at the power consumption of the underlining Google Service we found a power usage of about 6% which translates to an actual power usage of about 246mAh. The long running Google Speech Recognition (Offline) also reported an actual power drain of 120-160 with the Google Services reporting a battery usage of about 4% which gives us actual power usage of 164mAh. In contrast, when we ran the Sphinx Speech Recognition for the same time period, the resource consumption was as expected based on our previous result of 9.03mAh and we have actual power consumption of 61mAh. Also, we recorded the lowest actual power drain running the Sphinx Speech Recognition. From these results, the Sphinx Speech Recognition library ultimately have better performance in terms of power usage. As the Sphinx Speech Recognition library contains all required libraries and we could accurately track its resource usage, its resource consumption can be used as the benchmark for the minimum Power, Memory and CPU resource requirements to run Speech Recognition on an Edge Device´.

6.3.2 Continous Speech Recognition

Using the consumption statistics gethered from running the Speech Recognition, we explored the possibility of having a continuous speech recognition service running in the background by running a continuous speech recognition for an hour and present the obtained results below. Of note here is the low battery usage recorded by the Google Speech Recognition which does not reflect the actual battery consumption of the application which uses Google core services for the actual speech recognition. In this case, the Power Darin gives a more accurate picture of the of the actual battery consumption.

Name	**Battery Usage** (mAh)	**CPU** (s)	**Memory** (K)	**Time Running** (m)	**Power Drain** (mAh)
Sphinx (Offline)	61.0	193	62,131	63	80-120
Google (Offline)	0.694	23.1	42,198	59	120-160
Google (Cloud)	0.603	19.1	36,338	62	360-400

Table 2. Long Running Speech Recognition Resource Consumption

Considering that Sphinx Speech Recognition library has the lowest cumulative battery drain with an average power requirement of 9.03mAh per 10 minutes, we expected an average power consumption of 55mAh per hour or 1320mAh per day which equates to about 32% of a 4100mAh battery. In practice, we obtained a power consumption of 61mAh or about 1.5% of the total battery capacity. With these findings, running a Speech Recognition application in the background will have significant effect on the edge device total power consumption in the range of 30% to 40% of the total battery capacity and having a speech recognition running in the background for most of the day would significantly increase the device's battery drain. But in cases where speech recognition is of high priority, continous background speech recognition could run for about two days on a single charge. An alternative could be to modify the application with a sleep feature where continous speech recognition is automatically activated in moments of speech and deactivated in moments of silence. We assume that this will lead to great improvements in the battery consumption and could even lead to barely noticeable affect on the power consumption of the device.

6.3.3 Additional Processing Task

In speech recognition, there might be need to process the extracted text and saving or sending the process text to the cloud. From the consumptions results obtained, additional analytics tasks on the converted text could be implemented where the speech recognition is not continous as the resource consumption here is mostly insignificant. Where we have continous background speech recognition, the speech recognition already takes significant power and implementing further processing task would significantly increase the battery consumption.

7 Conclusion

In this thesis we carried out analysis of the computing resource usage of applications running on the edge with the aim of mitigating the various issue facing the traditional method of sending sensor generated data to the cloud for analysis and decision. To explore the possibilities of implementing additional processing tasks and reduce dependance on the cloud, there was need to first analyze the computing resource usage of a sample application running on an edge device. In analyzing the computing resource usage, we selected the speech recognition domain due to the growth in voice interactive devices and the high computational and memory resources required for speech recognition.

An Android speech recognition application was implemented using the Google and Sphinx Speech Recognition libraries. We then ran 10 minutes and 1 hour long offline and cloud-based speech recognition to analyze the application's memory usage. The Dumpsys tool and Android Studio Profiler tool were used in extracting the computing usage data. From the analysis of usage data, we were able to predict that we could implement continous speech recognition on an edge device although this will significant affect the battery consumption. Alternatively, we could modify the speech recognition application to sleep in between silence utterances. We proposed that adding additional processing tasks would significantly affect the power consumption of continous speech recognition applications. On the other hand, we could implement additional analytics tasks where we have short running speech recognition as the power consumption there is almost insignificant.

Our future direction is to explore the possibility of developing a continous and background speech recognition application running on the edge while greatly reducing battery consumption. Next, we could also implement text processing on the speech recognition application running on the edge while keeping battery consumption constant or with only slight increase. From our findings in here where an application that is given permission to use the microphone and started by user can continue to use the microphone in the background even when the screen is locked, it is possible to develop a stealthy spy device that uses background continous speech recognition to spy on the user. The possibility of using the resource usage footprint to identify such an application that is maliciously running speech recognition in the background could be investigated.

8 Acknowledgement

First I would to thank my family for the wonderful love and support they shown me throughout my study period at the university. I would not be here without your support, thank you very much. Secondly, I would like to thank my supervisor Alo Peets for his support, suggestions, corrections, patience and generally going the extra mile, thank you very much. I would also like to thank all the lecturers at the University of Tartu that I was priviledged to study under, I would not have made it this far without your guidance. A big thank you to my friends and colleagues at work for all the wishes and encouragements. Finally, I would love to thank the University of Tartu for giving me the opportunity to study at such a great institution.

References

[1] ALO, P., AND SATISH, S. Comparison of openfaas and fogflow for edge analytics.

[2] CMU. Cmusphinx @ONLINE. https://pypi.org/project/SpeechRecognition/, 2019.

[3] CMU. Pocketsphinx android @ONLINE. https://cmusphinx.github.io/wiki/tutorialandroid/, 2019.

[4] DOMO. Data never sleeps @ONLINE. https://www.domo.com/solution/data-never-sleeps-6, 2018.

[5] GOOGLE. Android dumpsys @ONLINE. https://developer.android.com/studio/command-line/dumpsys, 2019.

[6] GOOGLE. Android profiler @ONLINE. https://developer.android.com/studio/profile/android-profiler, 2019.

[7] GOOGLE. Android speech recognizer @ONLINE. https://developer.android.com/reference/android/speech/SpeechRecognizer, 2019.

[8] GOOGLE. Google cloud speech-to-text @ONLINE. https://cloud.google.com/speech-to-text/, 2019.

[9] MICROSOFT. Microsoft bing voice recognition @ONLINE. https://azure.microsoft.com/en-us/services/cognitive-services/speech-to-text/, 2019.

[10] TRINELLI, M., GALLO, M., RIFAI, M., AND PIANESE, F. Transparent ar processing acceleration at the edge. In *Proceedings of the 2Nd International Workshop on Edge Systems, Analytics and Networking* (New York, NY, USA, 2019), EdgeSys '19, ACM, pp. 30–35.

[11] YI, S., HAO, Z., ZHANG, Q., ZHANG, Q., SHI, W., AND LI, Q. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing* (New York, NY, USA, 2017), SEC '17, ACM, pp. 15:1–15:13.

[12] ZHAO, J., MORTIER, R., CROWCROFT, J., AND WANG, L. Privacy-preserving machine learning based data analytics on edge devices. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society* (New York, NY, USA, 2018), AIES '18, ACM, pp. 341–346.

Appendix

I. Glossary

Android Speech Recognition Application source code on GitHub.

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Bilal Abdullah**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Analysis of Software Applications Computing Resources Usage on the Edge: A Case Study of Speech Recognition

   supervised by **Alo Peets**

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Bilal Abdullah
15.08.2019