The University of Birmingham
School of Computer Science
MSc in Advanced Computer Science

Dissertation Report

Understanding User Behaviour by Mining Smartphone Usage Patterns & Exploring Them to Improve User Experience

Karthikeya Udupa Kuppar Manjunath

Supervisor: Mirco Musolesi

September 2014

Understanding User Behaviour by Mining Smartphone Usage Patterns & Exploring Them to Improve User Experience

Karthikeya Udupa Kuppar Manjunath, University of Birmingham
Mirco Musolesi, (Supervisor) University of Birmingham
Veljko Pejovic, (Supervisor) University of Birmingham

Mobility as a technology has seen advancement in the past two decades like no other rising to a level where it has become a necessity of modern day's daily life. From communication, entertainment to healthcare and fitness, mobile devices caters to all our needs in one way or other. Being such a key aspect of our lives, mobile presents us with interesting challenges that, if solved, can help improve our daily lives.

The foremost goal of this research is to capture user's interaction with his smartphone and the applications on it and gather information about the contextual conditions that lead the user to perform the interaction in the first place. We then use the data gathered to look for interaction patterns and factors that can ultimately be used to improve user's experience with the mobile device.

The research also aims at modelling the collected data which based on various factors, time and location among others, will try to predict application usage/application category usage. With the understanding of user behaviour, this model can further help us improve users experience. We also look at the modern day networking arena and provide a novel idea which would use *Software Defined Networking* alongside the intelligence gathered from the data about application usage to provide a better networking solutions for the mobile users. We will also look into what further improvements can be made in the domain based on the evidence we have gathered which would further benefit the user.

General Terms: Mobile Architecture, Smartphone Monitoring, Smartphone Usage Analysis, Network Optimisation

Additional Key Words and Phrases: Anticipatory System, Context Sensing Framework, Android, Smartphone Usage, Activity Monitoring, Network Optimisation, SDN, OpenFlow

1. INTRODUCTION

Over the past decade, we have seen several technological advancements that have created a huge impact on the mobile platforms, allowing them to evolve not only to be more usable and reliable but also have made them economically feasible for the general masses making it a necessity in our day-to-day lives from being just a luxurious commodity. Trends in several emerging markets in the recent years have indicated that there is not only a tremendous increase in web activity originating from mobile platforms but also provides us a clear indication that the people are moving from traditional computing platforms to mobile devices, making mobile platform a very exciting and vital research area.

Smartphones and other wearable computing devices such as smart watches and glasses are the most recent additions in the field of mobile communication devices [Oulasvirta et al. 2012]. Even after being this recent, the overall smartphone penetration however has surpassed every possible expectation, from 46% in 2012 it has now reached to 70% penetration among adults in the US, all this within 7 years of the launch of popular smartphone such as iPhone [Smith 2012; Asymco 2014]. This rapid adaptation of smartphones throughout the masses is a clear indicator of not only its popularity but also the need it fulfills by empowering the users with communication capabilities, internet access, multimedia capabilities, navigation etc, justifying it as the most important technological advancement in the last decade [Ross 2011].

These smartphones and other wearable technologies that have emerged recently have blended into out lives to become an extension of ourselves, allowing us to look into the the revolutionary ideas proposed by pioneers in this field in a new light [Weiser 1991]. They also get a glimpse of who we are, the minute decisions that we take everyday, our preferences in terms of shopping, dinning, travel or even in the people that we meet, ultimately helping us to manage and simplify our daily lives and focus on important things then getting entangled in managing technology.

However, various technologies that we use in parallel with the mobile platform still are being developed and setup as they were before. This although provides a working model of, but hinders the boost mobility can really provide. One such area is the networking platforms [Lee et al. 2014], mobile networks still work on a traditional networking systems with not many significant upgrades for mobile or smartphone specific network. Similarly other areas such as web content delivery also can be improved. We will look into this further in this research.

From a research point of view, this level of adaption of ubiquitous technologies which can give us such good insight provides us the means to achieve more in terms of understanding user's behaviour in his natural environment. Not only is this wide network of devices a source of limitless contextual information [Campbell et al. 2008] but also provides us with a way to help improve daily life of the user by considerably reducing the efforts one puts in operating the technology yet allowing him to enjoy the benefits of the technology in his day-to-day life.

However using smartphone as a research platform has its own demerits [Keshav et al. 2007; Falaki et al. 2011] from being able to provide only restricted access for development and device data to just the sheer diversity available in the smartphone market; making it hard for a single application to capture the data for the masses. This research we would try to overcome the barrier of developing a monitoring application for the Android platform with certain level of reusability for future research. It would be looking into application usage patterns for Android using an application developed called *WebSense* and try to find usage patterns which can help us further build a model to predict application usage among various other things. We would also be looking into a few possible uses to improve user's experience on the mobile phone using the prediction especially in the networking domain with the help of modern networking technologies such as *Software Defined Networking*.

2. DATA COLLECTION: CONTEXT FRAMEWORK, *WEBSENSE* AND DATA PROCESSING

Mobile platforms have various barriers when it comes to using them for research purposes. Each mobile platform comes with its own development framework, which makes it impossible to develop a piece of software which would flawlessly work on multiple platforms without any change. The platforms themselves have their own set of restriction on what contextual information can be monitored and what data that is being generated by the phone or the apps that are installed can be accessed. There are additional hassles such as ensuring compatibility of the application over the various devices that are available which work on that platform For example, iOS from Apple does not allow applications to monitor user application usage and web usage, both of which are very critical for this research. Android platform is considerably open and allows one to access various different kinds of contextual markers and app usage patterns. It also holds a majority among the masses in terms of mobile marketshare [Gartner 2013], making it a more suitable candidate to use for our research.

An essential part of this project is to develop and improve the process of gathering contextual information from real world users along with a way to monitor interactions on the mobile device on a day-to-day basis in an un-moderated environment. There are three essential aspects that need to be focused on to achieve this goal. Firstly, a framework which is used to gather contextual information from any Android based device. This framework was developed as a library for anyone to use in future research with minimal code to start monitoring contexts for their purposes. We will be discussing this further in section 2.1.

The second component is the *WebSense* mobile application, which is the front-end application that was mass-deployed on the *Android Play Store* allowing people to participate in the research. The app's primary goal was to allow collection of information while proving a front-end which would provide user useful information to provide value to the user. The third component is the web server responsible for storage and maintaing of the data along with providing methods to extract useful information for further research.

Although these components were developed to an extent in our previous research(see Appendix B) but various large scale changes were made in the way of their functioning and several performance improvements were made that allowed fixing of crashes and provided stability to the application; issues that were detected using ACRA crash logging system that was added into the application were also fixed during the research. In addition to this the web component which was just collecting data previously was developed further to do various other tasks that we would be discussing in section 2.3.

2.1. The Context Sensing Framework

Context is an essential part of any system that tries to understand user behaviour or monitor user interaction. Monitoring usage context is a vital component in our research as well as it provides us with the insight and the understanding of the user's condition when he performed an certain activity.

Context itself has been defined and revised by many researchers over the course of time partly because of the definition's need to take into consideration the advancements in technology and what can actually be achieved in the reality which was not possible when it was not known earlier. The most recent and probably the most apt definition of context is that "any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [Dey 2001]. This definition clearly outlines what we are trying to achieve with the framework.

The Framework was developed for Android platform after taking into consideration its openness in terms of data it is able to provide the overlaying application. Although there are several monitoring frameworks for Android [Lathia et al. 2013; Novak et al. 2013] however they all lack several vital components that are essential to our research such as application monitoring, web usage monitoring, network connections, WiFi access points among others.

Based on the platform restrictions, the sensors and the available APIs a set of contextual markers were selected to be collected, each type of context had its own process of extraction and reporting back.

— **Application Usage** - Application usage is the primary context that we are monitoring for this research. The framework provides information such as what app the user is using, for how long and meta-data about the app such as application name, package name, icon information etc. The process used to extract this information is by constantly scanning the device at fixed interval(4 seconds) and to look for changes. This information is propagated back to the app which is using the framework in the form of an *Intent Broadcast*.

— **Web Usage** - Web usage is derived from app usage, whenever the user is using an application which falls into the browser category and updates the data in the android browser database, the web browsing instance is observed. This is in the form of an URL and the duration it has been since the user moved to the page.

— **Location Monitoring** - Location context is monitored in a very effective manner (see Appendix B), proving a robust and fairly reliable process in providing location data to be used by the app. The polling duration and the provider can be customised based on the needs.

— **Power Consumption** - Power consumption is monitored in terms of battery power that is left, since polling the battery is not an effective way to do it, the framework constantly listens for battery update notification from the OS and then uses that data to calculate details and then sends over in the form of a broadcast. Several fixes were made to handle faulty power readings in the previous version.
— **WiFi Connectivity & Nearby Access Points** - WiFi connectivity of the device allows us to check if the device is ready to be connected to WiFi, if it is connected to an access point and if there are other access points nearby. The details of other access points along with their signal strength is also recorded. In the previous versions the scans were too frequent creating large scale record sets for WiFi and also repeaters for the same access-points were considered, this along with several minor issues were fixed.
— **Network Signals & Nearby Towers** - Network signals provide the connectivity status of the device with the GSM/CDM network. The strength of the connection along with nearby access-point details is also captured and provided back to the integrated application.
— **Bluetooth Status & Nearby Devices** - Bluetooth connectivity status of the device tells us if the device has bluetooth switched on or not. It also lets us know if the device is paired and what are the nearby devices, all of which can be used to predict co-location among other social networking patterns between users. The device is scanned and also a system broadcast provides updates to the framework at regular intervals.
— **Telephony Status** - The telephony status lets us know if the user is in a calling status or not.
— **User's Events (Real world activities)** - Real world activities that are recorded by the user on his calendar such as meetings are monitored. This allows us to look into the user behaviour based on his real world condition. The information is scanned at a certain interval, any updates are sent to the connected app in the form of notification.
— **Screen Status** - The status of the screen, if it is switched on or off is monitored and reported back. Off screen generally denotes inactivity of the user.
— **Data Connectivity Status** - connectivity status of the device to the internet using 3G, 2G, Edge or GPRS is monitored.
— **Mobile Device Settings** - Details about the devices various settings such as volume settings, brightness settings, manufacturing details, timezone is all monitored under the settings context.

The platform relies on a model of *Broadcast-Receiver*, any application using the framework would have to subscribe to the context framework's instance for receiving update about the particular context it wants to monitor along with the frequency of the updates it requires.

The framework's architecture and the broadcast receiver models are further explained in the appendix B.

2.2. *WebSense* - Front End Application

Although framework provides a robust groundwork to collect information, a font-end application is essential for the collection of the data for research purposes. The application that was developed for this purpose was *WebSense*. It was essential for the app to provide some valuable information as well so users would have an incentive to download it.

The application does two essential things, firstly it collects contextual information using the Contextual framework described in section 2.1. However this information on local device is not that useful so it has to be synced to the server where further processing of the data can be done, we will discuss this further in section 2.3.

The second important role of the application was to provide user some value in return of providing the data. Since this application was to be distributed on the App Store having a value was very essential. This requirement was completed by adding Application Trends screen which showed applications that were popular in the geographical vicinity and Web trends showing websites popular in the geographical vicinity. In addition to this it also allowed the user to monitor his app usage trends allowing him to identify the time spend on a particular app during the day/week/month.

The app was developed to a rudimentary level in the previous research, however it required several changes and improvements to actually function to its full potential and be deployed in the app store. Further the app was modified to accommodate the changes that were done to the sever and to the framework. You can read further about the internal functioning of the app and the development methodology in Appendix B.

2.3. Web Server Component

The web server component is responsible for storing all the information that the user's device provides in the required format. In the initial research a basic server was developed to provide API for storage and to accumulate data and provide statistics based listing for app usage trends.

The server constitutes of a *Node.js* backend component combined with *MongoDB* to store the information. The component store the required information coming from the app and various derived data into the various *collections*, which are NoSQL's version of tables.

— **User Information** - Information about the registered users along with their device information. This also includes information required to login and maintain multiple session on various devices. This also stores information about the user's *Office* and *Home* tags that we would discuss further in section 2.4.1.
— **Application Usage Information** - The application usage instances generated while being used by various users is stored after some processing, that includes cleansing of the data, updating the application listing record from Google Play Store. The information constitutes of application's package name, active time, launch time (time of the day), location information, Geohash tags and generated tags for location among others.
— **Web Usage Information** - The collection of all the instances of web usage by the user. This includes any record of web usage generated by the android system recognised browser (which writes data into android browser's database). The URLs are also scanned for meta data prior storing it into the collection.
— **Contextual Information** - Contextual information about the user gathered by the application is stored in a single place, i.e., all the various context markers are stored together but are tagged accordingly along with location tags for easy extraction.
— **Application Metadata** - Meta data collected from scarping application information on the store, primarily used for analysis of the kind of application to allow sorting of apps into categories. Other useful facts such as publisher, downloads, ratings etc are also captured. A web component and an API to perform this task was developed and can take any valid package name and fetch information accordingly.
— **Web Metadata** - Metadata about scrapped websites based on the user's browsing activities, includes information such as title, some content and images, this is used for generating the trends screen on the device.

2.4. Deployment, Collection & Processing of App Usage Data

The application after development and several cycles of revisions and fixes was deployed to the *Android Play Store* with the backend deployed on a secure University of Birmingham server. The deployed application was downloaded by people around the world and data was accumulated over the period of time. The detailed analysis of the data is discussed further in section 3.

The collected data that is stored after being processed at various different levels; For example the application usage records are cleansed by removing unwanted keys that are being passed, then each record is checked against existing application records in the meta data collection and in case it is not present the meta data is downloaded through the meta data API that was created for this purpose.

Location information which is available as coordinates makes it hard to work with so it was converted into a format which would work well with the custom processing algorithms. Additionally information such as *Geohash* is generated as well, use of which we will be discussing further in this section. Similar process is applied to the context information as well.

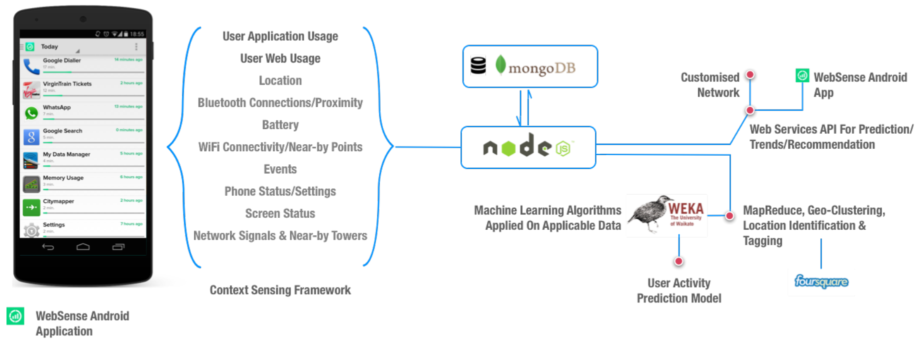

User Application Usage
User Web Usage
Location
Bluetooth Connections/Proximity
Battery
WiFi Connectivity/Near-by Points
Events
Phone Status/Settings
Screen Status
Network Signals & Near-by Towers
Context Sensing Framework
WebSense Android Application
mongoDB
node
Customised Network
WebSense Android App
Web Services API For Prediction/ Trends/Recommendation
Machine Learning Algorithms Applied On Applicable Data
WEKA
MapReduce, Geo-Clustering, Location Identification & Tagging
User Activity Prediction Model
foursquare

Fig. 1. Overall data processing and flow of information.

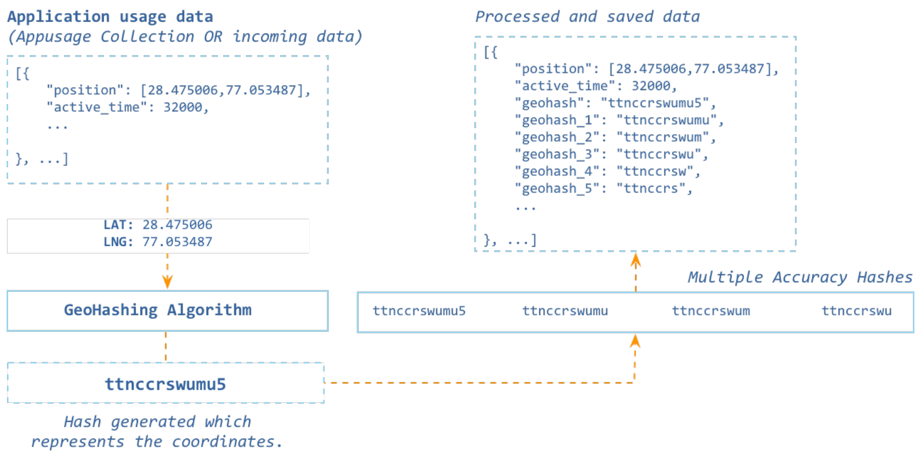
In previous work (refer to Appendix B for detailed information) we have used the basic data processing techniques to generate pattern with the help of standard *MapReduce* algorithm by reducing usage duration over application package-name and then geofencing the data over a particular radius (location information of the user). Although this method did provide some useful information on possible trends and served well for providing useful information back to *WebSense* app as a value add, but further analysis was done by utilising various other aspects of the data such as calculating trends information based on the time of the day the application was being used (local-time) and this was also combined with the location. This process was converted into API methods to be used in future systems that we would be discussing further in section 4.

Further we also looked into categories the application being used belonged to and associated it with the various times of the day and tried to extract useful information. The category of application being used is fetched from the app info collection that was generated whenever new applications are registered by the system, this allows to identify more meaningful patterns then just with a single applications.

Location being one of the most important contextual information was also analysed further to establish a relation if location has any significance on the apps being used by the user. However location itself might be irrelevant when we are considering multiple user's data once as geographical locations might be separated but still might have the same significance. For example a person who goes to office in London and a person who goes to office in Birmingham are both going to Office which in turn would have similar contextual implications. So inferring logical location instead of coordinate based location is important to analyse the result in the case of crowd-sensed data.

2.4.1. Geo-Clustering of Data & Location Tagging. The application usage data is tagged with geographical information (provided that the information was provided by the user's device and the location services were turned on by the user) however the location is in the form of latitude and longitude which in itself is a problematic type of information to process using traditional clustering algorithms. A technique was developed to process the data for easy use with MongoDB's inbuilt algorithms to provide clustered data that could be analysed further.

Geohash is a geocoding system for converting coordinate into a compact string format; it designed by Gustavo Niemeyer. This provides us a very effective way to handle coordinate data in the database in a format which is also unique and can be used for further analysis in place of using latitude and longitude directly. The has is generated by interleaving bits of the latitude and the longitude, the obtained bits are converted into a string [Balkić et al. 2012]. Each hash is a spatial bounding box which represents a certain geographical area; the longer the string is, more precise the location meaning the number of iterations of interleaving increases the accuracy, similarly if we remove the character (from the rear) the accuracy is reduced accordingly.



```
[{
    "position": [28.475006,77.053487],
    "active_time": 32000,
    ...

}, ...]
```

```
[{
    "position": [28.475006,77.053487],
    "active_time": 32000,
    "geohash": "ttnccrswumu5",
    "geohash_1": "ttnccrswumu",
    "geohash_2": "ttnccrswum",
    "geohash_3": "ttnccrswu",
    "geohash_4": "ttnccrsw",
    "geohash_5": "ttnccrs",
    ...

}, ...]
```

Fig. 2. Process of marking Geohash with multiple accuracy levels.

Since the location is in a string format and we can scale the accuracy (see figure 2) by increasing or decreasing the length of the string it provides us with an very effective way to aggregate the location data. It is also to be considered that this method is not just reducing the length of the characters of longitude and latitude as that would be providing a very low precision in terms of result as we keep reducing accuracy to identify bigger clusters. Each incoming app usage record is processed and Geohash for multiple accuracy levels are generated and stored within the record itself, this only happens if the location data is available. All the data processing algorithms that need to use location then use this to reduce the dataset further.

This proves very helpful when we need to process the data to identify critical location in user's life, for example home and work location of an user. As discussed identifying importance of location and tagging then would allow us to process the data for a group of user which can never be done with spatial coordinates. To identify the home and office location we analyse the app usage data, each app usage record consists of position/Geohash (coordinates and Geohash of multiple level), active_time (duration of the app for that instance), package_name(unique application identifier) and start_time_day (minute of the day at which the app got launched).

We cluster the application data by reducing the usage records over a period of time by accuracy level (hash with length 6, meaning a precision of .61km) using MapReduce algorithm, however this has be done based on time segments, so we consider two essential time segments, 0000 Hours to 0900 Hours and 0900 Hours to 1800 Hours of the day, these timings refer to the local time of the user and is not effected by the timezone he is in. These time windows would be the most probable time for people to be at home and work respectively (for most of the people, however exceptions might be there).

Based on the information we have, we mapReduce application usage on a Geohash of a particular accuracy for an individual to identify two prominent points in his daily life. Home location should be relatively clear however office location might be somewhat inconsistent provided there are holidays and weekends so further cleaning of the data based on our knowledge of home location can help us identify a viable candidate for office location. Once these locations are identified we tag the applicable app usage records as *Office* or *Home* based on geohash, this allows us to tag records which are created at home or office during the wrong time period, i.e. weekend records during afternoon will be most likely be tagged as Home even though the time period is wrong, similarly people working late at office would be tagged Office even though the time window assumes that they are at home.

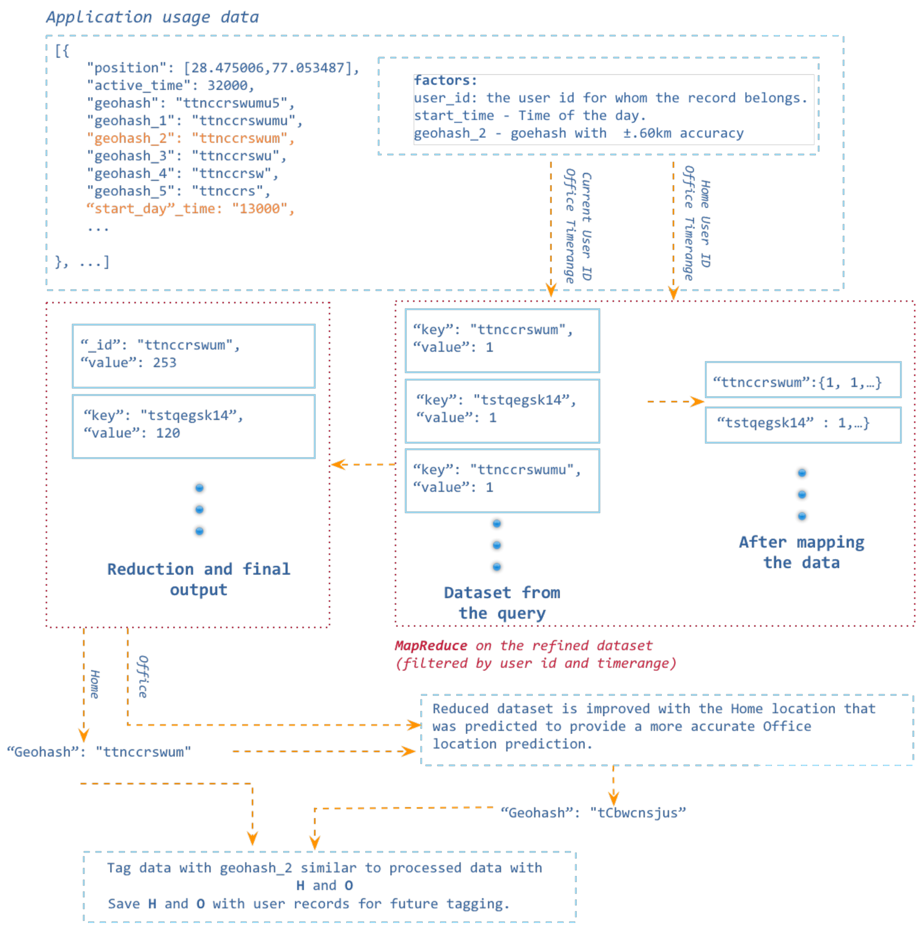
An API was built on top of this that allowed us to download the data in a format suitable for further processing using machine learning algorithms, this data would help us identify app usage patterns in office and homes and also will let us write prediction models which we will discuss further in section 3.

An interactive interface was created to analyse the extracted points further and not only to understand what the algorithm was really doing but also to analyse the results further visually (see figure 4).

However, the tagging of office and home constitutes of a very small segment of the available app usage records, there are lot of other places that the users travel and possibly use apps at those locations, there can be some statistical significance or even a correlation between the application being used, the time and the nature of the location. However, we have the same problem as before, i.e., spatial location may vary but the context that the location provides might be similar for example a restaurant in London may provide the same kind of contextual condition as a restaurant in Birmingham. Hence we need to identify the nature of the location to derive a relation between location and the application being used.

To process this further we MapReduce application usage over a Geohash key of high accuracy to extract a set of prominent recurring points (to filter out only statistically significant locations) for all the users (we are also taking into consideration common places that people visited). This gives us a list of locations which can be further explored. *Foursquare* provides various APIs[1] which allows us to geocode location data and extract categorical information about locations; For example, we can identify a collection of points at Waterloo Station in London as *"Tube Station"*. We process the set of locations that we discovered using the API to create a database of location and location category tags. This information is then used to tag application usage records with location tags.

[1]https://developer.foursquare.com/docs/ - Foursquare Developer API



```
[{
    "position": [28.475006,77.053487],
    "active_time": 32000,
    "geohash": "ttnccrswumu5",
    "geohash_1": "ttnccrswumu",
    "geohash_2": "ttnccrswum",
    "geohash_3": "ttnccrswu",
    "geohash_4": "ttnccrsw",
    "geohash_5": "ttnccrs",
    "start_day"_time: "13000",
    ...
}, ...]
```

```
factors:
user_id: the user id for whom the record belongs.
start_time - Time of the day.
geohash_2 - goehash with  ±.60km accuracy
```

```
"_id": "ttnccrswum",
"value": 253

"key": "tstqegsk14",
"value": 120
```

```
"key": "ttnccrswum",
"value": 1

"key": "tstqegsk14",
"value": 1

"key": "ttnccrswumu",
"value": 1
```

```
"ttnccrswum":{1, 1,…}

"tstqegsk14" : 1,…}
```

Reduced dataset is improved with the Home location that was predicted to provide a more accurate Office location prediction.

Tag data with geohash_2 similar to processed data with **H** and **O**
Save **H** and **O** with user records for future tagging.

Fig. 3. Process of identifying home and office locations for a user and tagging the app usage records.

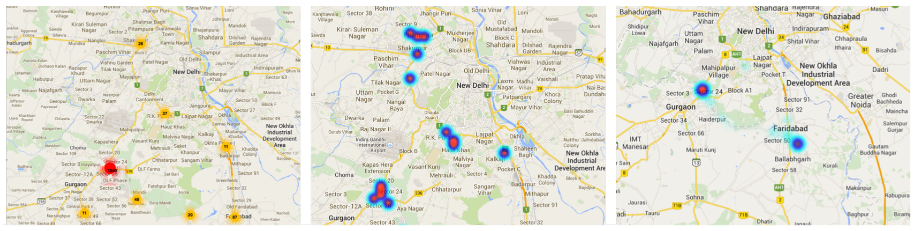
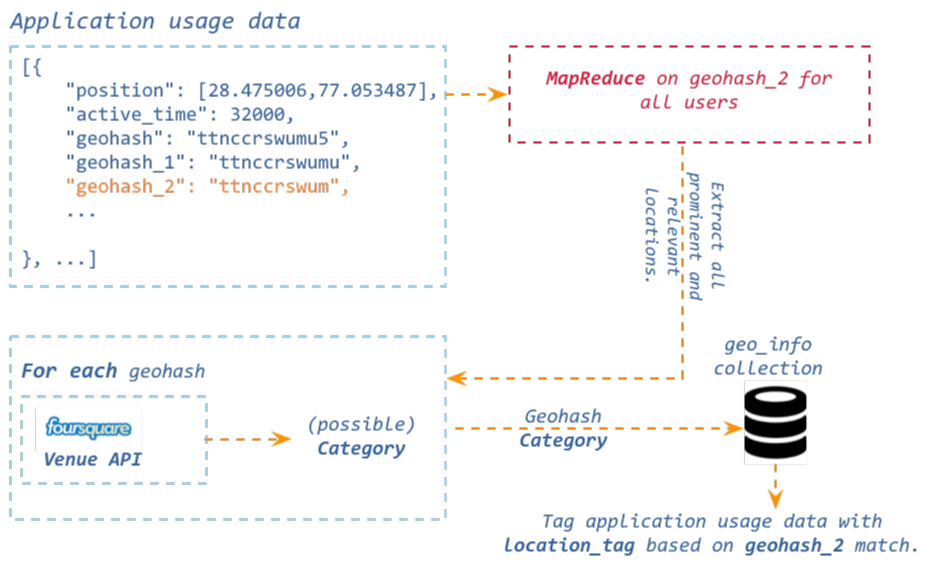


Fig. 4. Heat-map and clustering visualisation based on Geohash based clustering, (i) Initial basic clustering based on coordinates. (ii) Heat-map of the clustered data. (iii) Heat-map after the algorithm for home and office detected has been executed.


Application usage data
MapReduce on geohash_2 for all users
Extract all prominent and relevant Locations.
geo_info collection
For each geohash
foursquare
Venue API
(possible) Category
Geohash Category
Tag application usage data with location_tag based on geohash_2 match.

```
[{
    "position": [28.475006,77.053487],
    "active_time": 32000,
    "geohash": "ttnccrswumu5",
    "geohash_1": "ttnccrswumu",
    "geohash_2": "ttnccrswum",
    ...

}, ...]
```

Fig. 5. Tagging application usage records with location information.

An API was built over it to download the data in a format for machine learning algorithms with application information, active time, launch time and location category. The overall process and the flow of the information is depicted in figure 1.

Considering this is a small scale experiment with a limited set of users the diversity of the location among the group might not be optimal, but this process still provides a novel approach to tag specified location clustering of data which can be further improved with a larger dataset with more accurate location information.

3. ANALYSIS OF APP USAGE PATTERNS

The *WebSense* app once deployed on the *Android Play Store* was downloaded and used by users worldwide who were also contributing in the form of data for app usage and contextual data. In this section we analyse the collected data and try to understand and draw meaningful observations from it. We also use the data to generate prediction models from the data.

The app overall had 22 users in total over a duration of 7 weeks, however it is to be noted that users have joined and left (uninstalled) the research, meaning it is not necessary that all the users would have a consistent 7 week of records. The overall records collected overall 250,000 app interaction instances among the users and over 2500 hours of user interactions being recorded. The application diversity is also very interesting, over the course of time the framework has gathered information/meta data of 270 applications, allowing them to be used to mark app usage instances along with categories. During this duration 1950 URLs were browsed using the android recognised browsers.

The contextual information that was gathered over the period is much more; with over 700,000 among the users. The breakup of the contextual information is listed below in figure I.

From all the data that was gathered, we identified certain key components that were to be observed and with the help of various processes discussed in section 2 we filter the data.The following are some of the key components that were taken out for machine learning algorithms to process further.

Context Type	**Record Count**
SIGNAL_UPDATE_NOTIFY	31586
WIFI_UPDATE_NOTIFY	452844
BLUETOOTH_UPDATE_NOTIFY	47712
BATTERY_UPDATE_NOTIFY	39511
EVENT_UPDATE_NOTIFY	144088

Table I.

Contextual record subdivision among the collected in contextinfo collection.

— *package_name* - The name of the application package that is being used, this is a unique identifier for the app, several of the names have been been removed from set due to their association with the android system which makes them not a choice of the user. An example of this would be the android launcher.
— *category* - The category the application is assigned to by the developer. This information is scarped as a part of the meta-data collection when data is pushed to the server.
— *location_tag (personal places)* - The location tag is the tag which specifies if the location is either *Home* or *Office*.
— *location_tag (Public places)* - The location tag is the tag which specifies the outdoor location's significance based on the information fetched from the Foursquare API.
— *active_time* - The local start time of the application interaction commencement. Represented as minutes.
— *day_time* - the localised time of the day the user made that interacted in seconds, for example if the user interacts at 01:06 AM it would be recorded as 3601. This provides us an advantage, we can now compare data of users with the same time constraints wherever they are in the world.

3.1. User's Application Usage Models

Let us look into the Home and Office tagged users, out of the 22 users, 14 users had statistically significant data (others either did not have enough data or they did not provide location information making them irrelevant for this part of the study). Once the data is tagged we downloaded the data and ran it through a series of classifiers.

The goal of the models generated from this was the to predict the application usage based on time and location of the user, by location we mean the the local location_tags. We ran a series of classifiers for 3 different sets of individual's data. The classifiers used were *ZeroR*[2] as baseline, *NaiveBayes*[3], *Bayes Net*[4] and *J48*[5].

First set was the individual's set of *Office* records, for the set of records we ran classifiers based on *day_time* to predict *category*, the prediction highly depended on the data that was available for each of the user and also on the user's behaviour, from the various models being generated J48 usually had a very consistent prediction and was providing considerably better results then the baseline and other two classifiers (see Appendix A.1 for model's detailed summary). On an average J48 correctly classified an 80.26% of the records in contrast to the 62% by the baseline classifier, ZeroR.

[2]ZeroR - http://weka.sourceforge.net/doc.dev/weka/classifiers/rules/ZeroR.html
[3]NaiveBayes Classifier - http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/NaiveBayes.html
[4]BayesNet - http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/BayesNet.html
[5]J48 - http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

The second set of individual's *Home* tagged records, however not all the users had a detected home due to lack of significant records, around 11 people were statistically significant for this modelling session. The classifiers were trying to predict *category* based on *day_time*. The results (see Appendix A.2) as the previous one shows that J48 shows the most promising result of 71.11% accuracy, however the difference from the baseline is not as much as before.

The third set is a combined set of *Home* and *Office* records of the individual, here the modelling parameters change, we try to now predict *category* based on *day_time* and *location_tag*. Based on this we ran the classifiers, J48 provided a consistently good performance with an average of 70.35% with the baseline as 53%.

The deviation in prediction however is high enough for us to understand that predicating user's activity even after several stages of classification still is very complex, additionally, based on the experience with erratic behaviour of the classifiers with instances with low number of record we can also conclude that the prediction can be improved provided that we have a larger dataset.

We also analysed the data as a group, meaning we combined all personally tagged data of the users and tried to derive an efficient model from it. For this we have 2 variations we can try, first set would be the to analyse the relation between the category and the time of the day app is being used, we will call it *SET A* and the second set would be *SET B* which will be for a relation between category with location tags and time.

Data Set	**ZeroR**	**NaiveBayes**	**BayesNet**	**J48**
SET A	64.07%	39.48%	82.59%	83.95%
SET B	35.93%	42.89%	82.54%	84.12%
SET C	38.80%	38.38%	76.54%	78.11%

Table II.

With the collected data we ran multiple classifiers and built models, as you can see in Table II, on Set A we have a 84% predictability with the baseline at 64% and on Set B the results are even better with a 85.12% predictability with the baseline at 36%. Indicating a clear sense of predictability in this dataset, the dataset itself consists of around 60,000 records.

We also ran the series of classifiers on all the available data, regardless if they had location information or not to predict category based on time which we have marked as *SET C* in the result. As we can see from the percentages of correctly predicted instances the predictability is substantial however when we compare it to SET B, i.e. with the location tag, predictability is considerably higher, implying that location plays a significant role in the application being used.

3.2. Patterns in the Application Usage Data

With the data we can identify several factors that influence various decisions in mobile design and can be used to help improve user interaction. Let us first look into active time of an application. It indicates how long an application is active before the user switches to another application or closes the app or the mobile locked itself. From the collected data we calculated the average for active time is around 40 seconds, however there are cases of more then 2.5hour long interactions in case of multimedia and games applications. But the 40 seconds interaction windows actually provides us a glance into the overall attention span the user can spare per application at once, so applications should be developed to take into consideration that tasks should be

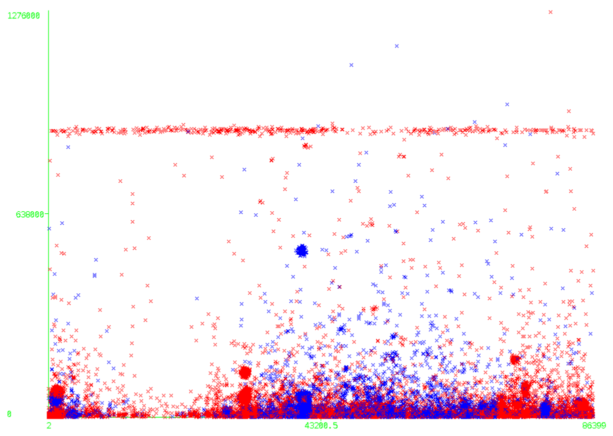


Fig. 6. Activity time throughout the day, red marks home instances and blue marks office instances.

complete-able within 40 second window hence not making the task requiring more attention then the user can already spare.

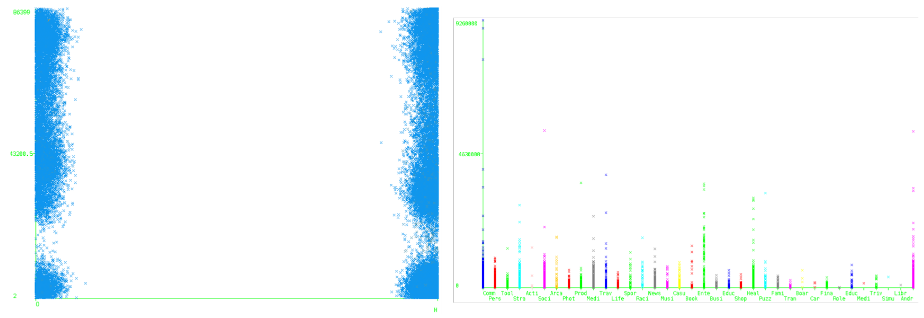


Fig. 7. Graphs showing different aspects of the data. (i) start time of the day with location tags, (ii) Various categories and active time of the app.

When analysing active time we also found that there is a time duration of around 15 minutes where we can see a clear pattern in figure 6 that throughout the day, applications tend to close here, The only possible cause of this can be concluded as screen turning off after 15 seconds. However, if that is the case, we can also see that there is a huge separation between user's app usage duration and the screen turn off time, we can hence reduce the screen switch off duration to around the same time as user's attention span in case of inactivity, the advantage being the battery being saved from turning off the screen. Using the available data we tried to sort app start time with location tag, we can see a pattern of people working late night at office and we can also see app usage pattern between office and home. We also can see the variation of active time between categories from which can understand that users spend more time on Entertainment, Communication, Social networking and Health and fitness application.

3.3. Model for Generic Location Based Prediction

From the available data we associated locations with categories as discussed in section 2.4.1 using Foursquare API. This gave us 70,000 app usage records tagged with

104 distinct location categories and 27 distinct application types. To make the prediction more accurate we have already removed common repetitive system apps such as *Launcher*, Android OS processes etc. We ran the the various classifiers that we used to predict category of application that one would use based on the location category and the time of the day. Furthermore we also tried to do the same prediction with just location type.

Clustering Algorithm	**Set A**	**Set B**	**Set C**	**Set D**
ZeroR	53.07%	53.07%	35.35%	24.26%
NaiveBayes	58.61%	63.12%	73.53%	47.67%
BayesNet	89.89%	63.12%	73.58%	88.00%
J48	92.61%	63.11%	73.62%	90.11%
J48Graft	**92.60%**	**63.11%**	**73.62%**	**90.09%**

Table III.

Set 1 represents, category prediction with location tag and time, Set 2 is category prediction with location alone, Set 3 predicts package name based on location tag and set 4 predicts category based on location tag and time. As we can see the performance of J48 and J48 Graft is exceptional in most cases when compared with baseline. Set A and Set B which are the most significant predictions are also the most important ones. The ability to predict the category based on where the person is and what time it is can allow us to understand a co-relation between location and the apps. It also helps us prove that there is a similarity in the application usage pattern between different people at different locations but with the same significance (for example, different tube stations in different parts of the country).

With this kind of prediction we can take this further by caching information based on what we predict, this along with various other API's that are available we can change the way networks interact with the mobile devices. We will be discussing this further in the next section.

4. USING THE DATA TO IMPROVE NETWORK DESIGN

Networking has evolved over the years, from a simple Ethernet connection between machines to transfer information with the evolving needs of the industries and the consumers. Organisations require a networked environment using modern technologies such as Virtual machines acting as desktop and servers on cloud-based networks, data-storage devices in the cloud and automatically managing of permissions. But with increasing demands networks are getting complex by the day, there are requirements for improvement in quality of service, constantly changing traffic and requirement of complicated security policies that the network needs to comply with the traditional networks are getting harder to manage.

These networking requirement are not only keep changing but also are very hard to comply with because of the way this has to be configured with networking devices such as routers and switches. Each manufacture would have their own way to configure it, making it hard for the administrators to follow thorough with the changes. And even if there is a way to do it on a device getting the change to propagate through the network on all the switches is nearly impossible because there is no standardisation among vendors and devices.

These problems make it nearly impossible for the networking to keep scaling at pace with the needs of organisations or network providers. Similarly is the case with mobile networks, from the analysis of the data in the previous sections we can clearly see the amount of interaction being performed by the mobile users on their respective devices is very significant. The amount of requests from the users is a constantly not only in the form of web pages but through the cloud based application that are available for these devices. This requires for a new form of networking that has to be more controllable and certainly more effective in providing content.

4.1. Software Defined Networking

The networking world has been experimenting with concepts which provide more control, flexibility in terms of dynamic rearrangement and configuration of the network, better moderation and easy management of policy groups with complex rules. One of the more explored way is with separation of control and data, this architecture of networks is known as Software Defined Networking or SDN. The concept relies on the ability to control the switches relatively easily. We separate the functionality of switches that support these architecture constitute of two layers or planes [Kirkpatrick 2013], a data control plane responsible for forwarding incoming traffic to the required destination. The decision of forwarding the traffic however is taken by the control plane, it maintains flow table to forward traffic that it encounters.

The control plane provides control over traffic routing is also responsible for propagating policies. The data plane uses low end protocols such as UDP, TCP to transfer data. On a larger scale we can define SDN as a three tier architecture, at the bottom is the southern API, responsible for providing commands to help communication of behaviour of the network's switches, discovering the topology. OpenFlow[6] is the most common example of this. On top of this is the controller (POX, NOX, OpenFloodlight, OpenDaylight etc), providing its own API for the applications on top of it to allow application to control and reconfigure networks structure. behalf of the applications [Paradis 2014]. Applications built on top of this are the ones who utilise the controllers methods to provide some useful functionality, prominent one being virtualisation ones [Alaettinoglu 2013].

4.2. SDN and Mobile Devices

SDN can create a huge impact on mobile devices by providing granular control over the network and its traffic. The challenge posed by the increasing requirements of the network can be solved by SDN networks with their dynamic management techniques to provide scalability and security in these networks which could not be traditionally achieved [Foundation. 2013].

With the prediction and contextual data available and the research that have been done before we can look into several possible improvements.

4.2.1. Predictive Caching with OpenCache & OpenFlow. The prediction API provides us the ability to predict application usage based on location and time. There are several API methods in the framework which can provide us with some kind of prediction for caching data.

— Application Prediction (Location, Time) - Prediction based on the location of access point and the time of the day. Can work with or without location or with or without time.
— Application Prediction (Location Tags, Time) - Prediction based on the type of location the access point is on and the time of the day. Can work with or without location or with or without time. This uses the prediction model generated using J48 classifier in the section 3.
— Web Prediction (Location, Time) - Prediction based on the trends of web usage based on location and time.

[6]http://archive.openflow.org/

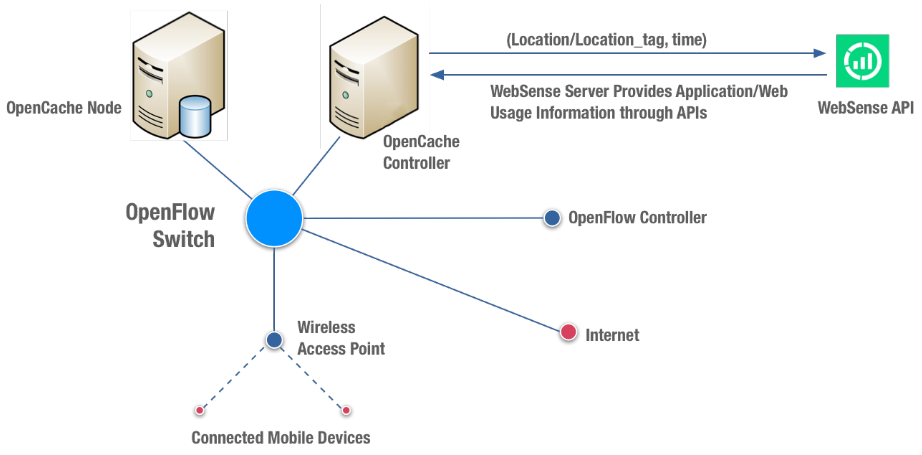


Fig. 8. Predictive Caching Architecture.

The application layer would implement a combination of OpenFlow and OpenCache [Broadbent and Race 2012]. The OpenCache's node would be fed data from the prediction API and based on the prediction it would cache the content. The caching of content in an SDN network has already been proven effective [Chanda and Westphal 2013] and the model itself has been proven in the section 3 to provide satisfiable predictions, hence a combination of both would succeeded. The figure 8 represents the mentioned architecture.

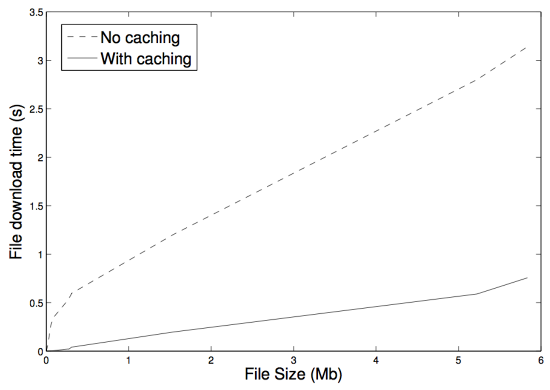


Fig. 9. Simple caching performance with SDN network [Chanda and Westphal 2013]

4.2.2. Predictive Load Balancing and Improvement to Quality of Service. The ability to monitor and control traffic in a network provides SDN with an opportunity to control how the network functions under increased load conditions or in case of network component failure. This allows us to improve the network's performance even in severe cases, [Foundation. 2013] provides a keen insight into how Mobile traffic can be handled with the help of SDN and OpenFlow, it also points out off-loading technology, policy monitoring and failure handling.

The loading balancing feature activates if there is an increase or decrease in user network activity which would either strain the network or make free bandwidth available. The best examples would be public places, for example, a train station would have an increase in commuters during peak hours, which would mean that the networks in the vicinity would receive much more load and would require more bandwidth. Decisions can be taken to provide a consistent user experience, can either change its network configuration to handle more people or in certain cases alter its services, for example switch from 3G to 4G to increase its available bandwidth if the bandwidth in use exceeds a certain threshold hence maintaining the quality of service for the end user. Once the traffic reduces it can re-allocate resources accordingly.

The data that we have gathered provides a new perspective to this concept. With our data we can identify the dense user activity points, we can then look into the various time-windows during which the density increases or decreases and create a predictive model which suggests based time-window and location the level of network load the access-point would have and automatically scale the network beforehand providing a seamless transition before any actual load is put on the server. We also have the data for the cell-tower and wifi connections in the contextual info collection. This data can even be used to identify the connected towers and improve the network's performance accordingly. The figure 10 represents the mentioned architecture.

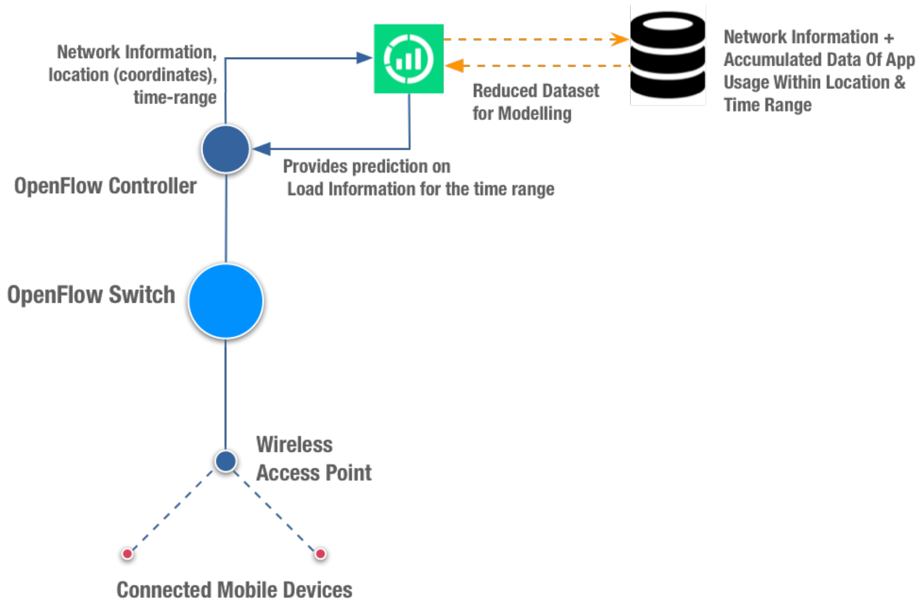

Network Information, location (coordinates), time-range
Reduced Dataset for Modelling
Network Information + Accumulated Data Of App Usage Within Location & Time Range
OpenFlow Controller
Provides prediction on Load Information for the time range
OpenFlow Switch
Wireless Access Point
Connected Mobile Devices

Fig. 10. Predictive Load Balancing & QoS Improvement Architecture.

Additionally, we can use the data that we have about the application. For example, if the users tend to use VoIP applications more at a particular place, at a particular time, we can optimise the network's QoS and QoE (quality of experience) for the users [Velrajan 2013]. This would however require some change in the way the applications are designed; we would require additional meta data, either inside the application or from the Play Store about the nature of the application, kind of services it requires and even the URLs that can be cached. An example of this would be, Skype would come tagged with "Service Type" as "VoIP", games with networking requirement would come tagged as "UDP" and so forth so we would know before hand what the network has to be optimised to, we call this application-aware network [Velrajan 2013]. This can in theory be extended to http requests, for example, if a news app such as BBC is being used it might be tagged with the API url, we can cache the data accordingly. Other caching details such as for how long would that content be valid.

If these changes are made available we can definitely improve the way networks are managed and change how they interact with mobile devices. Their ability to provide manage load and provide boosted performance to the apps/websites that are required or when there is an increase in traffic would improve the quality of experience people have on mobile

5. RELATED WORKS

The related work can be classified by various segments of the research. First segment that is the framework has had several efforts made on various platforms to develop a stable and reusable context sensing platform over the last few years. *EmotionSense* project developed initially for Symbian 60 and now android focuses on monitoring context for the purpose of behaviour and emotions of the users [Rachuri et al. 2010]. The framework provides various components such as adaptive sensing and broadcast based sensor information among other useful features. However, the android version being relatively new lacks several contextual markers that are readily available on the Android operating system.

Other platforms such as *funf* provides dynamic sensing along with customisable saving options. Mobile Sensing Framework works with Ambient Dynamics allows customisation of contextual scanning dynamically [Novak et al. 2013]. The framework also enables to mine and extract historical context that can be used in various ways.

The second segment is the monitoring of user application usage and other contextual markers to analyse and unearth important features in the data. A predictive launcher system called FALCON [Yan et al. 2012] provides predictive application launching services on Windows Mobile OS based on historical data of app usage that it gathers over the time along with other contextual markers. [Do and Gatica-Perez 2010] mines a large scale collected data to understand various patterns and statistically significant relations between various factors much like what we are trying to do here. Another research looks into the the various user activities and tries to derive implications based on the occurrence of various contextual factor and tries to build a user activity model with it.

[Oliver 2010] provides a similar research's finding on a totally different platform, *Blackberry*. With a dataset of over 17000 users with over a millennium of user interaction time, it provides a very clear explanation to various factor in mobile research but it still lacks to understand the relation of individual application interaction which we have tried to achieve in this research. *SystemSens* [Falaki et al. 2011] is also an interesting mobile architecture component built to analyse and research on mobile interaction, but it focuses on more system factors then user interaction such as power consumption and screen status, it does not give any focus to the user's specific interaction with the apps and webpages. Several other researches [Ferreira et al. 2011; Alawnah and Sagahyroon 2013] have focused on understanding the power consumption aspect, enabling us to understand and improve the situation of power consumption on mobile. The networking aspect of the research has not been explored much probably due to the fact that SDN is still an emerging technology and is not widely deployed yet. However there are a few initial ideas that are trying to reinvent networking design using modern technologies like SDN and OpenFlow switches. [Chanda and Westphal 2013] provides an architecture with practical implementation of an caching engine base on OpenFlow and SDN with proof for how caching can be improved with the help of SDN. Several white-papers have touched the subject of mobile based caching and load balancing in an Software defined network [Foundation. 2013; Velrajan 2013]. This is just theoretical work and does a practical implementation. meSDN takes the research one step further with a practical implementation of SDN for mobile network with an application-aware network which helps improve the quality of service [Lee et al. 2014], however, this does not consider the ability to predict in the caching or the predictive load balancing.

6. LIMITATIONS & FUTURE WORKS

There are several shortcomings in this research that we know of, however, each of them can be dealt with if provided enough time and resources. Firstly, the data collected is very limited, both in terms of the number of users and also in terms of the duration it was collected for. Additionally, the user group is relatively less diverse, i.e., it constitutes mostly of people in the range of 20-30 years and either work or study at universities, this creates a lot of discrepancies in the data such has no clear distinction in work and office location, less diverse applications being used, odd work hours etc. Additionally, the application had a few users who uninstalled application in between the study that led to several smaller user datasets which proved statistically insignificant in several cases.

The application was developed for gathering of the data with location as one of the aspects, however since location become a prime aspect of the study, a more accurate and frequently scanned location information would have been very helpful in deriving conclusions. This especially comes in handy when dealing with Foursquare APIs. Also, expanding the application's support to more devices and platforms would certainly help increase the dataset and the user-base. The foursquare tagging can be improved even further with better location data and by combining categories, at present it has diverse data which can be combined, for example, restaurants currently are tagged by types, i.e., indian, italian, persian etc. instead of being tagged as just restaurant.

The SDN model is another aspect that lacked certain features. Although it was well described in the sections above a fully working model with evaluation would certainly have proved vital to the research. Additionally having the meta-data about applications, which at present unfortunately is not available anywhere, would be helpful in providing increased quality of service to the users.

The SDN implementation of the caching engine with detailed analysis of the results and improvements it would be providing will certainly help us prove the usefulness of the architecture. A detailed model that uses application data along with network information would also help in providing the required data for the load-balancing architecture.

7. DISCUSSION

Through this research we have tried to look into the various aspects of mobile computing; the activities that we participate in daily, which can be improved for the betterment of the overall mobile usage experience and reduce the strain technology puts on the user's daily life.

The *WebSense* application along with its architecture provides an innovative platform that can be used further to not only analyse application usage but to understand various other co-relations between various contextual markers. Similarly, the Context Sensing framework can act as the groundwork for future contextual research on mobile by driving the efforts of the researchers towards the real problem away from the technological barrier. However, this framework can certainly be improved and can adopt certain features from existing framework such as EmotionSense [Rachuri et al. 2010] while retain its simplicity of usage.

We also looked into the gathered data and processed it with various in-built and custom algorithms to extract meaningful patterns in the data and also to model data based on location, location tags and time to predict application and category of the application that a person would use with a considerably high success rates using various machine learning algorithms. The duration or active_time provided us with an insight into human attention span towards application and how it can be used in development of a less strenuous application. The predication results when used with the emerging networking technology, Software Defined Networking, in the proposed novel architectures certainly seems to be promising as shown by the caching experiments and the prediction accuracy of the model, however a real world implementation would be required before we can further speculate its usability. QoS and load balancing technologies, which are already in use alongside SDN can also certainly benefit from the dataset we have about application usage, migration patterns and network connectivity data.

We believe that mobile certainly holds the potential to change the people's lives and alter the way they interact with the world. With more and more increase in mobile usage we would certainly have to turn towards emerging technologies to meet our ever growing needs and to develop architectures which would not only make the user's experience on the mobile devices seamless but also introduce a simplicity in the way we interact with mobile, helping us focus on the real-life rather then be entangled with technologies.

REFERENCES

Cengiz Alaettinoglu. 2013. Software Defined Networking. (2013).

Sameer Alawnah and Assim Sagahyroon. 2013. *Modeling smartphones power*. IEEE.

Asymco. 2014. Smartphone penetration reaches 70% in the US. http://www.gsmarena.com/asymco_pricing_doesnt_affect_smartphone_adoption_in_the_us-news-8982.php. (2014). [Online; accessed 09-September-2014].

Zoran Balkić, Damir Šoštarić, and Goran Horvat. 2012. GeoHash and UUID identifier for multi-agent systems. In *Agent and Multi-Agent Systems. Technologies and Applications*. Springer, 290–298.

Matthew Broadbent and Nicholas Race. 2012. OpenCache: exploring efficient and transparent content delivery mechanisms for video-on-demand. In *Proceedings of the 2012 ACM conference on CoNEXT student workshop*. ACM, 15–16.

Andrew T Campbell, Shane B Eisenman, Nicholas D Lane, Emiliano Miluzzo, Ronald A Peterson, Hong Lu, Xiao Zheng, Mirco Musolesi, Kristóf Fodor, and Gahng-Seop Ahn. 2008. The rise of people-centric sensing. *Internet Computing, IEEE* 12, 4 (2008), 12–21.

Abhishek Chanda and Cedric Westphal. 2013. A content management layer for software-defined information centric networks. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 47–48.

Anind K Dey. 2001. Understanding and using context. *Personal and ubiquitous computing* 5, 1 (2001), 4–7.

Trinh-Minh-Tri Do and Daniel Gatica-Perez. 2010. By their apps you shall understand them: mining large-scale patterns of mobile phone usage. In *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia*. ACM, 27.

Hossein Falaki, Ratul Mahajan, and Deborah Estrin. 2011. SystemSens: a tool for monitoring usage in smartphone research deployments. In *Proceedings of the sixth international workshop on MobiArch*. ACM, 25–30.

Denzil Ferreira, Anind K Dey, and Vassilis Kostakos. 2011. Understanding human-smartphone concerns: a study of battery life. In *Pervasive Computing*. Springer, 19–33.

Open Networking Foundation. 2013. OpenFlow-Enabled Mobile and Wireless Networks. https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-wireless-mobile.pdf. (2013). [Online; accessed 10-September-2014].

Gartner. 2013. Sales of Mobile Phones in All Regions Except Asia/Pacific Declined in the First Quarter of 2013. http://www.gartner.com/newsroom/id/2482816. (2013). [Online; accessed 09-September-2014].

Srinivasan Keshav, Yatin Chawathe, Mike Chen, Yongguang Zhang, and Alec Wolman. 2007. Cell phones as a research platform.. In *MobiSys*, Vol. 7. 2–2.

Keith Kirkpatrick. 2013. Software-defined networking. *Commun. ACM* 56, 9 (2013), 16–19.

Neal Lathia, Veljko Pejovic, Kiran K Rachuri, Cecilia Mascolo, Mirco Musolesi, and Peter J Rentfrow. 2013. Smartphones for large-scale behaviour change interventions. *IEEE Pervasive Computing (May 2013)* 1 (2013), 3–9.

Jeongkeun Lee, Mostafa Uddin, Jean Tourrilhes, Souvik Sen, Sujata Banerjee, Manfred Arndt, Kyu-Han Kim, and Tamer Nadeem. 2014. meSDN: Mobile Extension of SDN. (2014).

Gabor Novak, Darren Carlson, and Stan Jarzabek. 2013. An Extensible Mobile Sensing Platform for mHealth AND Telemedicine Applications. In *Proceeding of Conference on Mobile and Information Technologies in Medicine (MobileMed 2013), At Prague, Czech Republic*.

Earl Oliver. 2010. The challenges in large-scale smartphone user studies. In *Proceedings of the 2nd ACM International Workshop on Hot Topics in Planet-scale Measurement*. ACM, 5.

Antti Oulasvirta, Tye Rattenbury, Lingyi Ma, and Eeva Raita. 2012. Habits make smartphone use more pervasive. *Personal and Ubiquitous Computing* 16, 1 (2012), 105–114.

Thomas Paradis. 2014. Software-Defined Networking. (2014).

Kiran K Rachuri, Mirco Musolesi, Cecilia Mascolo, Peter J Rentfrow, Chris Longworth, and Andrius Aucinas. 2010. EmotionSense: a mobile phones based adaptive platform for experimental social psychology research. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*. ACM, 281–290.

Philip E Ross. 2011. Top 11 technologies of the decade. *IEEE Spectrum* 48, 1 (2011), 27–63.

Aaron Smith. 2012. 46% of American adults are smartphone owners. *Pew Internet & American Life Project* (2012).

Saro Velrajan. 2013. Application-Aware routing in software-defined networks. (2013).

Mark Weiser. 1991. The computer for the 21st century. *Scientific american* 265, 3 (1991), 94–104.

Tingxin Yan, David Chu, Deepak Ganesan, Aman Kansal, and Jie Liu. 2012. Fast app launching for mobile devices using predictive user context. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 113–126.

Online Appendix to: Understanding User Behaviour by Mining Smartphone Usage Patterns & Exploring Them to Improve User Experience

Karthikeya Udupa Kuppar Manjunath, University of Birmingham
Mirco Musolesi, (Supervisor) University of Birmingham
Veljko Pejovic, (Supervisor) University of Birmingham

A. INDIVIDUAL PREDICTION MODEL

A.1. Office tagged model's correctly predicted instances %

User	ZeroR	NaiveBayes	BayesNet	J48
User 1	41.67%	41.38%	56.60%	73.76%
User 2	57.81%	52.71%	99.99%	100.00%
User 3	64.86%	70.27%	64.86%	70.27%
User 4	87.27%	83.64%	87.27%	90%
User 5	64.86%	70.27%	64.86%	86.49%
User 6	87.27%	83.64%	87.27%	90%
User 7	32.43%	58.11%	96.40%	100%
User 8	43.10%	45.61%	43.10%	53.14%
User 9	62.09%	63.51%	62.09%	69.67%
User 10	41.67%	42.11%	64.53%	86.23%
User 11	84%	74%	84%	84%
User 12	43.48%	43.48%	43.48%	50.84%
User 13	85.42%	85.42%	85.42%	85.83%
User 14	83.89%	82.46%	83.41%	83.41%
Average	*62.84%*	*64.04%*	*73.09%*	**80.26%**

A.2. Home model's tagged model's correctly predicted instances %

User	ZeroR	NaiveBayes	BayesNet	J48
User 1	37.79%	37.79%	37.79%	42.77%
User 2	49.70%	99.99%	43.02%	99.99%
User 3	69.16%	68.92%	66.27%	72.53%
User 5	73.33%	100%	100%	100%
User 6	38.75%	42.19%	38.54%	47.16%
User 7	69.96%	70.08%	69.24%	75.51%
User 8	52.69%	52.69%	52.41%	54.67%
User 9	41.67%	56.60%	41.38%	73.76%
User 10	70.93%	74.41%	68.37%	82.82%
User 11	49.33%	50.67%	49.48%	49.63%
User 13	84.92%	84.42%	81.41%	83.42%
Average	*58.02%*	*67.07%*	*58.90%*	**71.11%**

A.3. Office and Home tagged model's combined correctly predicted instances %

User	**ZeroR**	**NaiveBayes**	**BayesNet**	**J48**
User 1	38.27%	38.27%	39.21%	43.15%
User 2	39.65%	54.29%	99.99%	99.99%
User 3	68.81%	69.03%	65.71%	72.57%
User 5	32.43%	92.79%	58.11%	100%
User 6	39.38%	38.66%	41.72%	46.34%
User 7	69.85%	69.15%	72.76%	76.83%
User 8	54.85%	53.76%	54.74%	54.96%
User 9	43.95%	43.99%	52.05%	66.09%
User 10	72.03%	71.36%	74.54%	80.07%
User 11	47.53%	48.15%	47.74%	49.59%
User 13	85.19%	84.97%	85.19%	84.28%
Average	*53.81%*	*60.40%*	*62.89%*	***70.35%***

B. SECTION 2

The following sections are extracted from a previous work of ours submitted for the Mini research work at University of Birmingham. This explains the architecture of the software and the backend used in this project which was developed to an extend in the previous works.

Chapter 3

Android Context Sensing Framework

Context is a very primal component in all of the anticipatory systems. As one can conclude from figure 5.1, the contextual information that can be gathered using modern smartphones is a lot and more and more sensors are being added onto the devices with every new release. However, there are various mobile platforms and each consisting of its own development environment, tools and deployment stores making it very difficult to develop a framework which can function on multiple platforms. For the purpose of this project selected the Android development environment for various reasons. The platform this provides a much more flexible and open framework to extract contextual information. It also the most widely used mobile operating system and consists of devices of all price ranges hence providing a much more diverse target audience.

Although there have been many contextual frameworks in the past some of which are for Android as well; providing a relatively easy methodology to extract meaningful information (Novak, Carlson, and Jarzabek 2013; Rachuri, Musolesi, Mascolo, Rentfrow, Longworth, and Aucinas 2010), however the present requirement of the project is much simpler and there are some components that the frameworks presently do not monitor, for example, (Novak, Carlson, and Jarzabek 2013) does not monitor network, WiFi access points, (Rachuri, Musolesi, Mascolo, Rentfrow, Longworth, and Aucinas 2010) does not monitor user's schedule and phone profile. Also a framework at this point would provide more flexibility and access to the more recent API's which older frameworks do not cover. Although The context sensing framework designed for the purpose of this project but it is designed in a way to interface and connect to any application for monitoring purpose.

3.1 Contextual information types

There are various sensors that are available for the purpose of monitoring in the modern mobile devices and supported by the Android SDK. The following are the contextual information which were considered to be supported and available for subscription in the first version of the framework.

- Location (GPS or Network)
- Battery/Power source information

- Network connectivity status
- Network signals (Cell tower connectivity)
- WiFi access points and connectivity status
- User events
- Bluetooth status and nearby devices
- Telephony events
- Screen activity context
- User phone profile (Screen brightness, Volume settings & Vibration)

We will further discuss a few of the the above mentioned context's in details and how a scanning methodology was customised for it be both effective and efficient.

3.2 Framework's working architecture

In order to understand how the framework would actually function once integrated we divide the system's architecture into two components, first is the external end points of the framework which any application can interact with the second component is the actual internal component which handles subscriptions, monitoring, polling and content delivery.

3.2.1 End-points and external handling

As we described earlier the purpose of creating a new framework is to make sure information is provided in the simplest possible way and should be extensible. Also the framework has to be plug and play, there should be very minimal code that would have to be written on order to integrate it with an existing application and to perform what is required. The functionality of any existing application should not turn away from its original goal to focus on the integration of the framework and procuring data from it. The framework should just be a component integrated into the application providing a stream of required information and which can be controlled by the application.

The framework's architecture can be described as a subscription based model, wherein each of the components can be subscribed externally by the connecting application. In addition to that each subscription can be provided with additional parameters in order to tailor the subscription based on the needs of the application.

To begin monitoring any particular context one just has to call the *monitorContext* method from *ContextManager* object. The call then initiates a monitoring service for the context based the parameters that is being passed along with it.

As we can see in the function call, there are several parameters, let us look into them for more detailed understanding of the function's actual functioning.

```
public void monitorContext (
            ContextManagerServices mService,
            long minimumUpdateTime,
            long pollingTime,
            ArrayList<Integer> flags
    );
```

Snippet 1: Function definition of the monitor context.

- ***ContextManagerServices* Parameter -** *ContextManagerServices* is an enumeration defining various types of contexts that can be monitored by the framework, this specifies which context the application wants to monitor using the framework.
- ***minimumUpdateTime* Parameter -** This specifies the minimum time the application should wait before sending a forced broadcast about the context regardless if there is any change in the context or not. This is useful for application which want information about context at regular intervals regardless of the magnitude of the change.
- ***pollingTime* Parameter -** This parameter specifies the time interval after which the framework should poll the sensors or the operating system for information about the context. This is specifically for information which do not have system provided callbacks (Broadcast/Intent architecture) and require polling or in cases where power consumption for constant monitoring is too steep and is not feasible (e.g. GPS based location monitoring).
- ***flags* Parameter -** This is an extensible parameter which is essentially an array of flags which can be passed to the framework. This also provides a backward compatible way to extend the framework without changing the original function call definition. An example for the use of this parameter is for passing information about the source to use for location tracking to use while performing the polling task.

With the help of this single call, one could start monitoring any contextual information that is made available in the present version of the framework. In case of missing parameter related to the timing information, the framework uses a constant value for monitoring tasks which is customised for differently for each type of context.

To notify the application about contextual information the Android platform provides a very effective methodology to transfer information between the framework with the help of *BroadcastManager* which sends a message called *Broadcast* to all the *BroadcastReceiver*. The working of the methodology is very simple yet very effective. As figure 3.1 explains we can ask for an object to register itself to the *LocalBroadcastManager* to be notified in case it receives a broadcast with a particular name or tag which is referred to as *action* in Android.

The framework uses the same action string for all the notifications with more detailed information about the content being supplied inside the *Intent Bundle*.

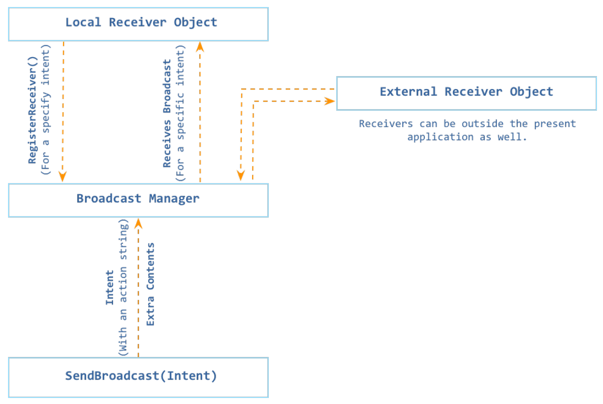


Figure 3.1: Architecture for the information broadcasting model.

The application using the framework would have to register a broadcast receiver just once to the*LocalBroadcastManager* for any and all contexts the framework is monitoring. When the framework has an update it would send a broadcast consisting of the following components:

- **Action -** All broadcasts from the application are associated with one action string '***CONTEXT_CHANGE_NOTIFY***' regardless what type of context it is.
- **Additional Information -** Other then the action string, broadcast also contains additional information in the form of a collection of key/value pairs. The framework sends various other values along with each of the broadcasts.
  - ***CONTEXT_TYPE* -** The type of context the broadcast contains.
  - ***Context Information Flag* -** This information is passed with the key named after the context type that is passed, hence varies depending on the content it contains. The value is a JSON (*JavaScript Object Notation*) formatted string containing information passed by the framework.

Finally, the application should be able to stop monitoring a context as easily as it begin monitoring the context. The context can be stopped from being monitored and hence the application would receive no further updates by simply calling

3.2.2 Internal working

The internal working of the framework constitutes on multiple components of the framework working in parallel and interacting with each other constantly. As described in section 3.2.1,

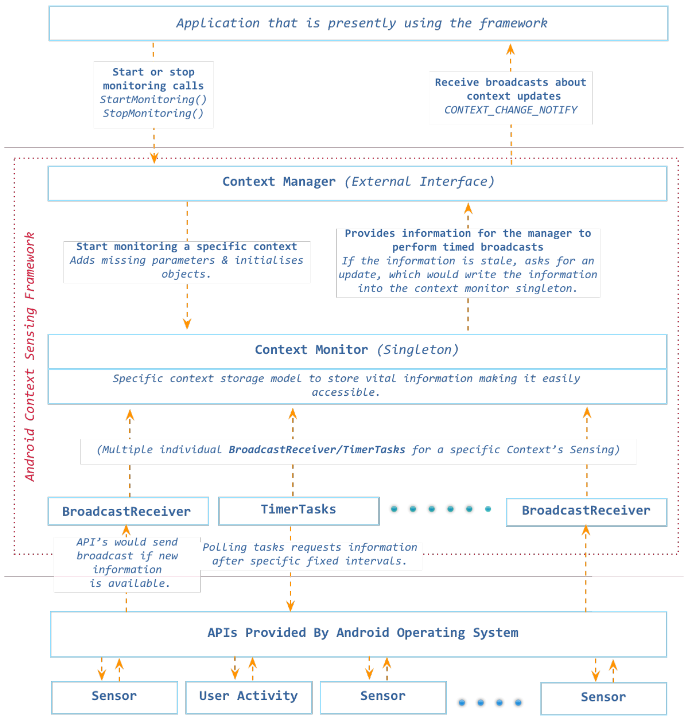

Application that is presently using the framework
Start or stop monitoring calls
StartMonitoring()
StopMonitoring()
Receive broadcasts about context updates
CONTEXT_CHANGE_NOTIFY
Android Context Sensing Framework
Context Manager (External Interface)
Start monitoring a specific context
Adds missing parameters & initialises objects.
Provides information for the manager to perform timed broadcasts
If the information is stale, asks for an update, which would write the information into the context monitor singleton.
Context Monitor (Singleton)
Specific context storage model to store vital information making it easily accessible.
(Multiple individual BroadcastReceiver/TimerTasks for a specific Context's Sensing)
BroadcastReceiver
TimerTasks
BroadcastReceiver
API's would send broadcast if new information is available.
Polling tasks requests information after specific fixed intervals.
APIs Provided By Android Operating System
Sensor
User Activity
Sensor
Sensor

Figure 3.2: Architecture of the Context Sensing Framework.

```
public void stopMonitoringContext(
            ContextManagerServices mService
        );
```

Snippet 2: Function definition to stop monitor context.

ContextManager is the external interface with which applications trying to use the framework would have to interact with. The class is also responsible for delivering broadcast at specified *minimumUpdateTime* interval. However, the framework is build on a on a set of classes which ensure that latest information is always made available to *ContextManager* and is done is an efficient way.

A singleton class, *ContextMonitor* is primarily responsible for performing all the monitoring tasks at the specified time intervals and updating the values for the *ContextManager* to access and pass on further to the application. There are two kinds of context extraction process that the class has to deal with.

- **Monitoring Tasks -** Certain contexts have a broadcast and intent model provided by the operating system, in which case, a broadcast receiver for that specific action string is created in the singleton to receive information and update its variables accordingly and in case of a relevant change a broadcast is sent to the receivers. The subscription to receive the update is added and removed whenever the applications requests the framework to.

- **Polling Tasks -** Certain contexts need to be accessed using Android system API. These calls would have to be made at the specified time intervals to keep the information updated. For this purpose we use a timer task in *ContextMonitor* and is triggered when monitoring is requested. In case there is a relevant update in the information a broadcast is sent to all listeners about it.

  There are several advantages of using this development model, primarily this provides a single point of access to all the information that is required by the framework which provides easy of use and extensibility to the framework. Additionally, a single instance of the monitoring task ensures there are no multiple tasks being run by the framework for the same type of context which can be accidentally triggered in case the application using the framework starts to monitor same context at multiple places if the singleton was not in place. Although the framework provides a very resource effect method to gather information (see 6 for performance benchmark).

3.3 Understanding individual context monitoring

Each of the context's being monitored by framework have their own unique methodology in which they work. The uniqueness helps them to perform monitoring in a very effective and power efficient way. Let us look in details a two of the important context's working.

3.3.1 Location

Location is one of the most important contextual information that can be extracted by the framework. With modern devices which people carry everywhere it has become all the more important. The major concern with location monitoring is the power consumption factor when using GPS (Ben Abdesslem, Phillips, and Henderson 2009), so continuously monitoring of GPS is never recommended on mobile devices. Also, time it requires to switch on the GPS sensor.

Getting a triangulation from the satellites can also be a time consuming task sometimes. Modern devices provide network based location services which require both network connectivity and internet to provide location. This process is battery efficient but accuracy is usually questionable.

The framework provides the application option to opt for either of the technique in addition to the ability to specify the polling interval.

Android operating system stores the last known position[1] which is updated when any of the application tries to fetch location using GPS or network provider. So once every few minutes (defined by the provided polling time) the device first looks for location that is stored using the *getLastKnownLocation* method. Then the framework analyses if the location that is being provided by the system is a better location then the one it already has and if it meets the criteria (see figure 3.3), if the criteria is met this location is stored and broadcasted. If not, the system would trigger the specified location provider (GPS or Network) to provide it with a new location, the fetched location is then broadcasted and stored.

3.3.2 Network information

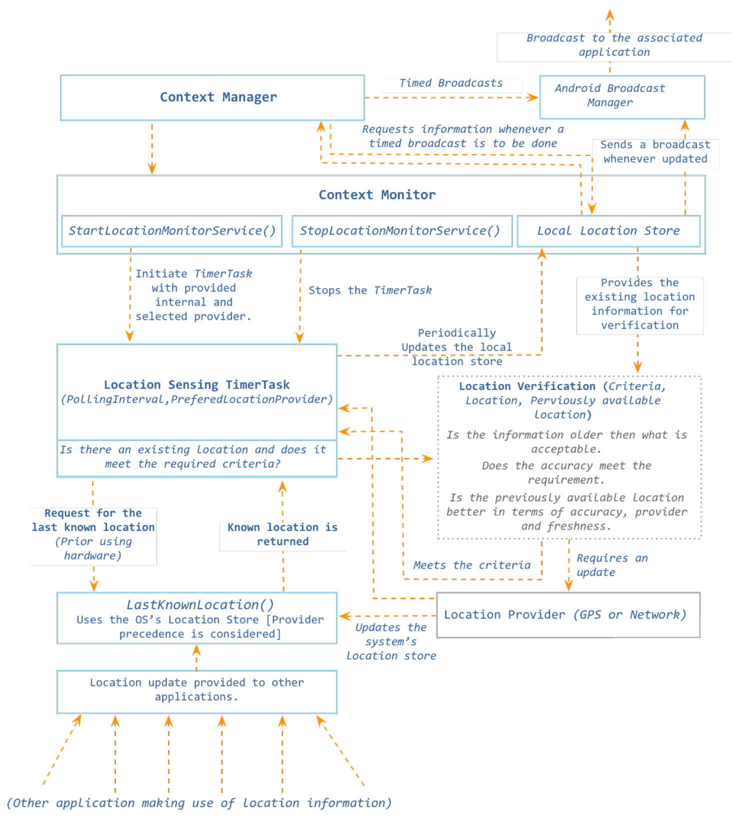
Mobile phones are almost always connected to a cellular network, when the user moves the network connection also changes. The systems decides which of the towers providing signal to connect to out of the various ones that are available. This information can be very useful in terms of context. For example with this information one can map the network density and signal strength of the area. There are several components that constitute of network information and there is not a single method to monitor this.

- **Signal Strength -** Signal Strength constantly changes and can be monitored by a system provided broadcast receiver. However the frequency of the change is very high, hence broadcast is done only when there is a significant change in the signal levels.
- **Cell Towers Nearby -** Although the device usually connects to a single cell tower but there are various other towers which broadcast their presence and the device picks it up along with other information such as their signal strength. This information is also provided through a broadcast receiver by the system.
- **Data Connectivity -** The networks may or may not be providing data connectivity to the device or the user might have turned off the data connectivity. This information needs to be polled constantly as the broadcasts are not reliable and do not provide internet connectivity information. The framework frequently checks for updates about connectivity information and notifies if there is any change.

3.4 Shortcomings and possible improvements

The framework provides a solid foundation with features that make it extensible and might provide useful to various kinds of application and research work on the Android platforms. There are several areas which should can be optimised and improved further in the future versions.

[1]http://developer.android.com/reference/android/location/LocationManager.html#getLastKnownLocation



Broadcast to the associated application
Context Manager
Timed Broadcasts
Android Broadcast Manager
Requests information whenever a timed broadcast is to be done
Sends a broadcast whenever updated
Context Monitor
StartLocationMonitorService()
StopLocationMonitorService()
Local Location Store
Initiate TimerTask with provided internal and selected provider.
Stops the TimerTask
Provides the existing location information for verification
Periodically Updates the local location store
Location Sensing TimerTask (PollingInterval,PreferedLocationProvider)
Is there an existing location and does it meet the required criteria?
Location Verification (Criteria, Location, Perviously available Location)
Is the information older then what is acceptable.
Does the accuracy meet the requirement.
Is the previously available location better in terms of accuracy, provider and freshness.
Request for the last known location (Prior using hardware)
Known location is returned
Meets the criteria
Requires an update
LastKnownLocation() Uses the OS's Location Store [Provider precedence is considered]
Location Provider (GPS or Network)
Updates the system's location store
Location update provided to other applications.
(Other application making use of location information)

Figure 3.3: Process for efficient monitoring of location.

Firstly, the framework can be made more robust and reliable. Currently it has been designed for devices running Android version 4.0 and 4.4, backward compatibility would certainly make the framework useable on a wide range of devices providing more diversity to future studies.

Although android is at present one of the most popular operating system in the mobile market, there are a considerable number of the mobile users who use other popular operating systems like iOS and Windows platform, an extension of the sensing framework with similar functionality (within the bounds of the operating system) on other platforms would be certainly helpful in conducting studies with a broader group of audience then just people using android.

Additional context's like accelerometer, proximity, ambient light and application monitoring (currently implemented in an external build over the framework) are presently missing in the framework and would be valuable additions to it.

The sensor information gathering technique is very basic in nature and lacks adaptive qualities like in more complex libraries EmotionSense (Rachuri, Musolesi, Mascolo, Rentfrow, Longworth, and Aucinas 2010), which allows the sensing to be done in an adaptive manner instead on a fixed interval. For example movement of the user should be tracked based on the movement he has done during the previous time interval, if the location change between two interval is considerably less or none at all, the next sensing should take place an increased time interval then usual or if the change is considerable the sensing time interval should be reduced. Additionally, it should take into consideration the battery usage of the sensor and the current state of the device, i.e. sensing intervals should adapt itself based on the battery life of the device to improve overall performance of the framework.

Given enough time the framework can be expanded into a much more robust, functionally rich and highly customisable component which can be used in various kinds of studies and real world application.

Chapter 4

User's smartphone usage data collection application

The most essential goal of this project is to devise a technique to gather information about how a real world user interacts with his mobile device on a day to day basis in his natural and unmoderated environment. The context sensing framework provides a very robust foundation using which we can gather information about the context in which the user is using the mobile device. But at present the framework does not consist of any possible way to monitor and gather information about mobile phone interaction of the user both in terms of web and application usage.

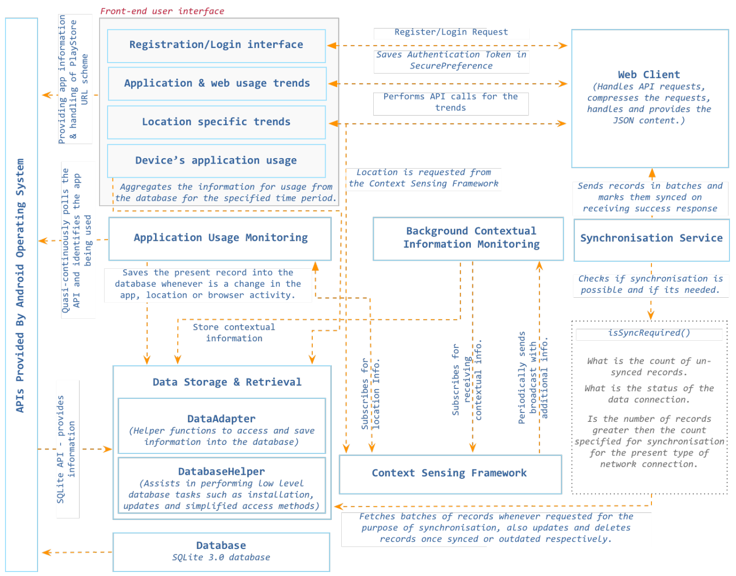
WebSense is an application for the Android platform developed for the very purpose of collection of user application usage data collection along with the related contextual information. The application is integrated with the context sensing framework described in chapter 3 which it uses to extract contextual information of the user.

There were two aspects of the application which were taken into consideration while designing the architecture of the application. First was the actual development of the monitoring functionality to provide an efficient way that was to be used to monitor how the user interacts with the device in a ubiquitous way and eventually send it to a central data repository on the web for collective analysis. Second aspect was the utility value it should provide to the user which is essential for crowd-sensing application as apps in the crowd-sensing domain should be lucrative enough for people to install even if they are not interested in the research value of the application (see section 7.1 for deployment challenges). In the following sections we look into each of the aspects and the various components which collectively provide a skeletal over which the application functions.

4.1 Application usage demographics collection

Android is a very open platform in terms of providing access to data sources when we compare it with the other most popular operating system for mobile platforms such as Apple's iOS (Apple Inc. 2013). This allows us to develop application which can monitor user interaction in detail.

The most essential criteria of any monitoring application is to maintain a near 100% uptime, which means that the app should always be running in the background and performing the necessary operations. Implementing a front end interface provides user an option to terminate the process very easily by accident or intent hence denying the change of gather information. However a background service on the other hand is more complicated to be terminated and also ensures that no hinderance is caused to the user in performing his day to day mobile phone usage. Hence the architecture of the app consists of several of background services each with their own set of responsibilities. This, along with a design to ensure that the monitoring service is switched on in case of device restart, accidental termination by the operating system or in case of an internal malfunction causing the termination of the application provides the system to operate at a near 100% uptime (see chapter 6 for detailed evaluation).



Front-end user interface
APIs Provided By Android Operating System
Providing app information & handling of PlayStore URL scheme
Registration/Login interface
Application & web usage trends
Location specific trends
Device’s application usage
Aggregates the information for usage from the database for the specified time period.
Register/Login Request
Saves Authentication Token in SecurePreference
Performs API calls for the trends
Web Client
(Handles API requests, compresses the requests, handles and provides the JSON content.)
Location is requested from the Context Sensing Framework
Sends records in batches and marks them synced on receiving success response
Quasi-continuously polls the API and identifies the app being used
Application Usage Monitoring
Background Contextual Information Monitoring
Synchronisation Service
Saves the present record into the database whenever is a change in the app, location or browser activity.
Checks if synchronisation is possible and if its needed.
Store contextual information
isSyncRequired()
What is the count of un-synced records.
What is the status of the data connection.
Is the number of records greater then the count specified for synchronisation for the present type of network connection.
Data Storage & Retrieval
DataAdapter
(Helper functions to access and save information into the database)
DatabaseHelper
(Assists in performing low level database tasks such as installation, updates and simplified access methods)
SQLite API - provides information
Subscribes for location Info.
Subscribes for receiving contextual info.
Periodically sends broadcast with additional info.
Context Sensing Framework
Fetches batches of records whenever requested for the purpose of synchronisation, also updates and deletes records once synced or outdated respectively.
Database
SQLite 3.0 database

Figure 4.1: Architecture of *WebSense* Android Application.

Let us look into the various background services along with their role in the overall architecture of the system.

4.1.1 Application usage monitor

The primal responsibility of the app is to monitor the user’s application usage continuously and with certain level of dependability. This task is performed by the service *AppUsageMonitor*. The service is a self-reviving service so in case it is terminated either by the operating system or due to a malfunction it revives itself within a matter of seconds, ensuring an almost 100% uptime.

Since Android does not provide a broadcast or any sort of delegation methodology to know when the application is switched, the only possible approach to do this at present is a quasi-continuous monitoring of the device’s application stack. An interval of 4-second was chosen between each scan since research has indicated that user’s attention span switches between the device and the environment somewhere within this duration (Oulasvirta, Tamminen, Roto, and Kuorelahti 2005), hence we can assume that the user would switch applications as well at around the same frequency. Android provides access to the list of currently active processes through the *ActivityManager* class. The class not only provides information about the running tasks but also provides a mechanism to access the order of the application’s screen presence. With this information we can identify which application is currently running and being interacted with on the device by the user.

Android also equips its developers with intents which one can listen to for detecting user’s action of turning the device’s screen on and off. This allows us to stop monitoring user’s activity tasks when the phone is turned off hence saving crucial resources.

The application uses SQLite database[1] to store the monitoring information. This includes package name of the application, start time, end time (Both in UTC timestamp format), duration of the run, location it was used at (subscribes to the framework, listens and stores location information constantly) along with the day of the hour the application was used at. Additionally, a flag marking the synchronisation state of the record is also maintained.

Whenever an application switch is performed the data is written into the database hence minimising the chances of data lose incase of a malfunction or termination. The information is also saved in the scenario when location update is received while an application is running, hence splitting a continuous session of the application based on the location. This is particularly useful in case of navigational applications as the user would keep it open over an extended period of time while his location changes constantly.

If the application is detected to be the default browser of the device, the app also gathers information about the page user is presently viewing. Although, there is no direct way to perform this, android provides an API to access browser history[2], the most recent record in history would be the page the user is viewing currently. Whenever user changes the URL a new record is created and hence the time spent on a particular URL is also recorded.

4.1.2 Contextual information monitoring

ContextBackgroundMonitor class is also a background service which is responsible for registering and monitoring the contextual information. Presently it monitors all the contextual data provided by the framework (see section 3.1). It is responsible is to make sure that as soon as the

[1]self-contained, transactional SQL database engine https://sqlite.org/
[2]Browser Data Provider - http://developer.android.com/reference/android/provider/Browser.html

service is initiated it subscribes to all the available contextual information provider and whenever there is any new information made available by the framework it saves it in the database. Since we require information about location for mapping the information geographically and also have to provide user with information in relation to his location, all the context information is stored along with the latest location information attached to it.

The class is initiated as soon as the monitoring begins and when the class is destroyed it releases all the receivers and initiates the stop procedure in the framework (see section 3.2.1) hence stopping monitoring process altogether.

4.1.3 Web synchronisation service

The context sensing and the application usage monitoring generates a considerable amount content in terms of memory. This can directly be used for the purposes of local mining (see section 4.2.2) but this being a crowd-sensing application, we need the information at a central location to further analyse information of the user-base as a collective. For this purpose, the information that is in the local datastore has to be constantly sent to the remote server on the web.

The class *SyncManager* performs all of the synchronisation of the device with the web by uploading the information in the most efficient way possible. After a certain interval of time the class triggers a method to check for a possibility for a need of uploading of the information. Records which have not been synced in the database is the first criteria the method checks for. To avoid repetitive reads to count un-synced records the monitoring services automatically increments the count whenever a new record is added and stores it on a preference file, this file is read by the synchronisation process. If the criteria for data upload is met the information is uploaded in chunks of fixed number of records in the form of a JSON string and once the data is successfully uploaded the records that were sent are marked as synced. The synchronising process uses the end points provided by the web application and sends the data after compressing it using GZIP [3]. The records are sent with an authentication key to both validate and associate the records, this key is obtained during the login process (see section 4.2.1 and 4.3.1).

The service is also responsible for periodically clearing old data which has been synced already and are beyond the time-period which is relevant to the application (30 days in this case).

4.2 User interface & Providing utility value to the user

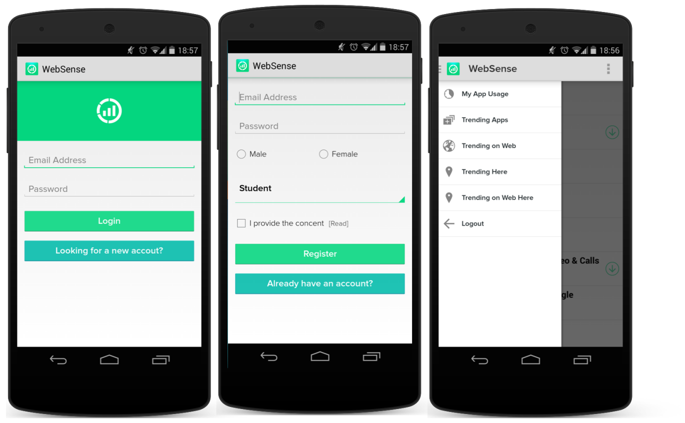
The monitoring and reporting aspect of the project could have been done without actually a front end with everything entirely handled by background services but the primary test subjects of this application are not a selective group of people, like any crowd-sensing application the future research based on this application tends to make this available to as many users as possible, not only the people who are interested in the research aspect but common public with Android devices. As we will see later on in chapter 7 one of main issue when dealing with crowd-sensing application is that it is bottle-necked by the fact that the entire research, application's widespread usage and the amount of data gathered depends solely on the decision of the user to install and keep the application on his phone. So to ensure widespread participation and retention of the

[3]GZIP File format specification -http://tools.ietf.org/html/rfc1952

application we need to provide the user utility value through the frond end of the application, so regardless if the user is concerned with the study or not he would still have a reason to install the application.

The user interface of the application which runs atop the data gathering services provides Information and feedback to the user and also helps in gathering certain user's personal information. The application is also integrated with a crash reporting framework called ACRA [4] which provides the crashes and issues faced by the users to the web server, helping us to improve the application's performance and making it more stable.

4.2.1 Login/Registration interface



(i) Login interface (i) Registration interface (i) Navigation drawer interface

Figure 4.2: Initial screens of *WebSense* app related to authentication and basic navigation

The login/registration interface is the initial screen of the application where in the user provides the below mentioned information if he is registering or just logs into the application if he has already registered.

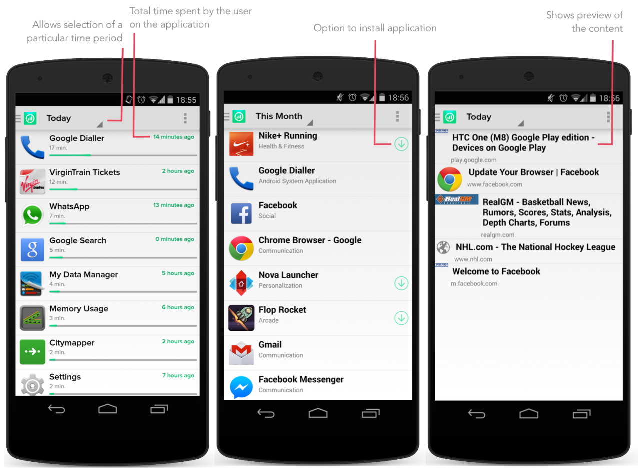
- Email Address - Allows us to have the means to contact the user in the future.
- Password - For authentication purposes in case the user logs out or wants to login on another device.

[4]Application Crash report for Android - https://github.com/ACRA/acra/

- Gender - Can be used as a metric to perform future studies.
- Employment Status - Additional parameter which can be used in future studies.
- Consent to allow monitoring - Along with a link to the End user's license agreement.

The application allows multiple sessions using the same credentials on multiple devices. An option to log out of the application is also provided inside the application, in case the user wants to do something privately or wants to stop monitoring completely.

4.2.2 Application usage information



(i) User's application usage (ii) Trending application (iii) Trending webpages

Figure 4.3: Content display screens of *WebSense* app providing trends and usage information.

Application usage of the user is being constantly monitored by the application in the background, this interface provides a view by accumulating the total usage over a certain period of time. The interface provides information about the total usage in a very visual way allowing users to compare the various application. An option to filter the information to a particular duration (day, week and month) is also available.

4.2.3 Application and web usage trends

This provides similar as the previous screen but instead of providing information for user's application usage, it provides aggregated information about the trends throughout the user-base of the application. It shows the trends of application usage based on the data accumulated over the period of time on the web server over a period of time. The web API has an end point which provides the information for the required duration of time (day, week or month).

The application's information is also provided along side, hence icon and the name of the application is shown along with an option to install the application if it is not already installed. This acts as a recommendation system for the user to try new application based on popularity.

Along with the application usage trends the app also shows information about the web usage trends, showing information along with some content and images from the websites popular among the masses in the given duration.

4.2.4 Localised web and app usage trends

The information about the app and web is can also be restricted to the present geographical area. The app has the capability to present the user information about what is popular in the user's current area. This can be very helpful for example, if the user is opens the application at a train station the app would provide information about the apps and websites people use there, which are most likely to be the ones providing information about the timings of the train.

The geographical radius is decided on the server the location is provided by the context sensing framework which then is passed along with the web request. The process of handling this request is discussed further in detail in section 4.3.2.

4.3 Web API and feature extraction

As we have seen the application provides a robust and lucrative way for the users to provide their application usage information, however in order to achieve this at a larger scale and not just for a single user we need a web server to handle storing of information, extract features and finally process it to provide meaningful information for the application to display and even perform predictive tasks at a later stage. For this purpose a web application was build and deployed on the internet. The application is build using Node.js [5] which provides a very reliable framework to build high performance web API's, dealing with an increasing number of web requests (Tilkov and Vinoski 2010) which is bound to scale over time. The datastore for the application is a MongoDB[6] database which provides various functionalities to process and extract information efficiently. It also provides lot of features to improve handling of high request rates and take care of the needs for scalable systems.

The web app is essentially a collection of RESTful API end points with JSON data format to store, process and provide information, all of whom incorporate GZIP compression in their

[5]Even-driven JavaScript backend http://nodejs.org/
[6]Non-relational, Document based database https://www.mongodb.org/

requests and responses to decrease bandwidth being used. It also constitutes of a analytics dashboard which showcases various metrics from the server. Let us briefly go through the available API endpoints and their internal working.

4.3.1 Storing information

The application stores various types of information. The first possible request an app can make is to register or authenticate the user. The information about the user is stored in the *user* collection. The user document is designed to handle multiple devices and multiple sessions running simultaneously. The document model also stores other information that is sent by the app when the user registered which can be used as a research parameter for future studies.

When authenticated the service checks for the user record along with the device information and either creating or updating the device information that is provided with the login information and returning a unique authentication key which can be used for all further requests.

Storing of web and app usage information is more complex, when the service is called the first thing the server checks is if it is a valid service in terms of the authentication key that is being supplied. If it is, then it starts processing the information that is provided. The user is almost instantly provided an OK status message while the processing is going on.

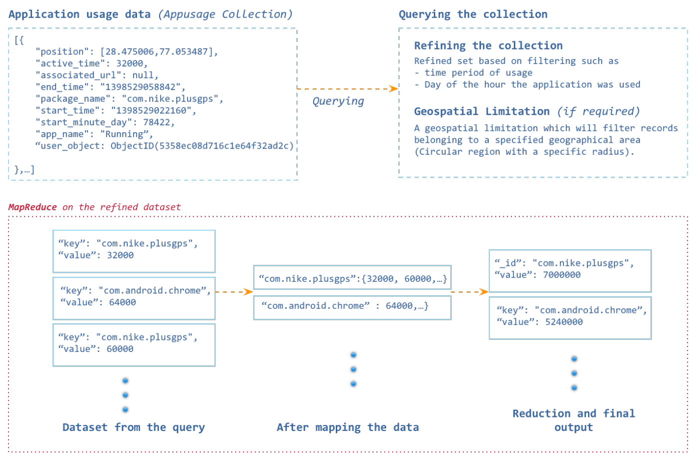
It collects all the package names that were sent in the request and checks the existing collection of application information for their records, the ones which are new are then sent to a different service call which asynchronously checks each of the package name and fetches information about the application from the app store by scrapping the web page for information. The location information is also corrected and stored in the required format to perform geospatial queries later on.

The process for storing context information is also identical, except there is not much processing at present and once the information is cleaned and converted into required format it is directly pushed into the *context* collection.

4.3.2 Extracting meaningful information and patterns

The web app is also responsible to identify and provide patterns identified from the stored information to the application on the device about application and web usage. The information is extracted from the database using the MapReduce technique which MongoDb supports. The details of the technique are explained in the Figure 4.4.

The process for extraction of information usually involves aggregating the app usage for each package, however,this can also accommodate filtering based on parameters such as like duration and location information. Location filtering uses the geospatial functions that is provided by mongoDB to reduce the records to a particular area only. The radius along with things such as packages to ignore can all be configured in the config file for the server allowing dynamic changes to the server. The diagram above shows in depth the working of the functionality.


Application usage data *(Appusage Collection)*
Querying
Querying the collection
MapReduce on the refined dataset
Dataset from the query
After mapping the data
Reduction and final output

```
[{
    "position": [28.475006,77.053487],
    "active_time": 32000,
    "associated_url": null,
    "end_time": "1398529058842",
    "package_name": "com.nike.plusgps",
    "start_time": "1398529022160",
    "start_minute_day": 78422,
    "app_name": "Running",
    "user_object: ObjectID(5358ec08d716c1e64f32ad2c)

},…]
```

Refining the collection
Refined set based on filtering such as
- time period of usage
- Day of the hour the application was used

Geospatial Limitation *(if required)*
A geospatial limitation which will filter records belonging to a specified geographical area (Circular region with a specific radius).

```
"key": "com.nike.plusgps",
"value": 32000

"key": "com.android.chrome",
"value": 64000

"key": "com.nike.plusgps",
"value": 60000
```

```
"com.nike.plusgps":{32000, 60000,…}

"com.android.chrome" : 64000,…}
```

```
"_id": "com.nike.plusgps",
"value": 7000000

"key": "com.android.chrome",
"value": 5240000
```

Figure 4.4: Applying MapReduce on the app usage data.

4.4 Shortcomings and possible improvements

The application and the framework were both developed over a period of 1.5 months. The purpose of this project was to lay an initial foundation for a robust data gathering application along with a framework which would be re-useable component for context extraction both of which can be re-shaped and used for studies beyond the scope of the present project. This being said, there are various improvements that can be done in the existing application and its web counterpart.

4.4.1 Android app

The android application has a few shortcomings in this first release which can be improved in the future releases along with addition of new features.

- Firstly, any improvements that were mentioned in relation to the framework (see section 3.4) would effect the overall performance of the app, both in terms of resource consumption and the amount of data that can be gathered using the application.

- The application at present fails in monitoring web usage if they not performed on the default browser which is accessible by the Browser data provider (*Chrome* on Android 4.0 and above) in the Android SDK. This can effect the overall demographics of the web usage data being collected as interest of the users using other applications to browse internet would not be registered by the application.

- The application also fails at monitoring tasks which can perform their functionality in the background while user is either using some other application or has switched off the device's screen for example applications such as music players and radios can play audio for the user while running in the background without having the front-end open.
- The application presently provides analytics and trends information which is very generic which can be customised and converted to a personalised recommendation system among various other possible improvements that can be done to provide more value to the user as an utility application (see section 5.2).

4.4.2 Web component

The web component can be improved considerably by adding more features in addition to improving overall performance and scalability of the system.

- The web component at present does not deal much with the context information, it simply makes use of app usage and location information to provide various results. Future features can make use of all the available information both for research purposes and also for implementing in the application.
- The web component's security can be improved by using *Bcrypt*[7] to store and encrypt private information. Additionally, more precautions can be taken by disassociating the contextual data from the user (see section 7.3).
- The database connection at present picks a connection from the connection pool of the driver, which is a reusable pool of already active connection helping in reducing frequency of opening and closing of connections (MongoLab 2013). There are several things that can be done to improve the speed network transactions.
  - Sharding[8] maybe applied and to the MongoDB instance to improve the performance of the application and handle increase in the data needs.
  - At present whenever there is an API call the web service triggers the MongoDB driver along with its customised query which reads data from the database, process it and returns it back to the service to be delivered further to the device. Considering the number of records being created by the devices, once this application has become widespread this process would start taking more and more time to process the request. A caching layer which would store the information in memory instead of reading raw information from the database and processing it again can improve the performance of the queries. Technologies which are meant to handle rapidly changing content in the memory such as Redis[9] can be used for this purpose in the future.

[7]Bcrypt: Bowlfish file encryption http://bcrypt.sourceforge.net/
[8]MongoDB Sharding - http://docs.mongodb.org/manual/sharding/
[9]Redis: An Open source key-value datastore - http://redis.io/

  - Use of MapReduce technique to perform various complex functions in the application benefits the overall performance, but at present each time the service is called the MapReduce process is performed. This can be avoided by using incremental MapReduce[10] which automatically reduces the information when new information is added incrementally hence reducing the load on the server by a considerable amount. Future implementations can also look into a Hadoop based MapReduce implementation on the existing MongoDB instance as it has shown promising results (Dede, Govindaraju, Gunter, Canon, and Ramakrishnan 2013).

[10]MongoDB: Incremental MapReduce - http://docs.mongodb.org/manual/tutorial/perform-incremental-map-reduce/