Journal of Theoretical and Applied Information Technology
15th June 2018. Vol.96. No 11
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A WEB APPLICATION FOR TRAFFIC STATUS UPDATE USING CROWD-SOURCED DATA ACQUISITION AND REAL-TIME MODIFICATION

[1]SHUVASHISH PAUL, [2]PINKU DEB NATH, [3]NASEEF M. ABDUS SATTAR, [4]HASAN U. ZAMAN

[1,2,3]Student, Dept. of Electrical and Computer Engineering, North South University, Bangladesh

[4]Professor, Dept. of Electrical and Computer Engineering, North South University, Bangladesh

E-mail: [1]shuvashish.paul@northsouth.edu, [2]pinku.nath@northsouth.edu, [3]naseef.mohammed@northsouth.edu, [4]hasan.zaman@northsouth.edu

ABSTRACT

Traffic jams are one of the most frustrating inconveniences experienced in all the major cities of the world. In the case of Dhaka, the capital of Bangladesh, factors such as high population density and increasing usage of private transport result in horrendous traffic jams on a daily basis and loss of valuable working hours. This paper introduces rTraffic -- a smartphone based web application that aims at making this issue a little more bearable by combining a crowd-sourced data acquisition model and real-time notifications system to provide a visual representation of the current traffic conditions in Dhaka and send notifications pertaining to the major intersections in the city. The paper also talks about the basic methodology, low-level implementation details and scaling factor considerations for real world deployment and performance benchmarks.

Keywords: *Real-time Traffic Notifications, Crowd-sourced Data Model, RESTful Application, Android, Traffic Congestion*

1. INTRODUCTION

Bangladesh is one of the developing countries and has been seeing continuous growth in its gross domestic product (GDP) per capita for the past few years [1]. As the GDP per capita increases, so does the purchasing power an individual possesses. Many industries are thus currently experiencing a "boom" in the local Bangladeshi market as citizens are able to afford more and more products. Research has shown that the automotive industry is one of the heavyweights among them [2].

However, as the amount of cars people own increases - the roads that they are to drive on seldom see any improvements. They are over-congested, and over capacity. As a result, traffic congestion has become a part of "Bangladeshi life." The fact that one has to brave traffic congestion if they are out on the streets is unfortunately now a certain phenomenon [3]. It is not uncommon for one to have to brave traffic jams that last from thirty minutes to as long as three hours on the streets of Dhaka, time that could otherwise have been put to more productive use [4]. This issue has been growing for the past 15 years bit by bit, and is starting to reach critical mass [5]. There are infrastructural problems that aggravate the traffic conditions since there are no fast and dedicated mas rapid transits (MRTs) such as trains etc. In addition, most the governmental and commercial centers and service providers are located in the capital city, including universities, banks etc. These lead massive traffic load at all hours of the day [4]. A feasible solution to these problems will require long term implementations of policies and constructions.

Nonetheless, faced with this fact of life, we, the team behind rTraffic (Realtime Traffic) sought to provide a logistical crowd-sourced solution to the problem by building an application that makes the situation a little more bearable. It collects real-time traffic information from users who are currently on the roads around key intersection by applying a crowd-sourced data model, aggregates this information and makes it available to all users of the application. So to say, one can find out about the status of roads one has to pass through to go to their destination at the planning phase of their trip. They may then make adjustments to their travel plan, or wait until the congestion clears up. The application also features alerts for traffic anomalies (that may or may not be causing congestion) - which are all too common in Bangladesh.

The initial work of this proposed solution presented in this paper was presented in [1].

2. EXISTING SOLUTIONS

Much work has been done in this area that aims to make commuters aware of traffic conditions before they set out on their trip globally. It is common to find municipal governments provide websites or APIs that make this data available for use. This data is then leveraged by 3rd party service providers like Google Traffic to provide real-time traffic status updates [7]. Unfortunately though, there are no such feeds that contain data for Bangladesh.

Researchers from MIT have also attempted to use cell-phone base station (BTS) data to triangulate location information from it and then run analysis atop it to track congested roads [8], however this approach requires a partnership with telecom infrastructure providers, which renders it inaccessible for us. Researchers from IBM have devised a way to predict traffic congestion in the city of Dublin using Semantic Web Technologies relying on integration with numerous sensors such as weather data, road work information, incident or event trackers [8], however there is no way to get access to such data in Bangladesh in a streamlined way even if one wishes to. Researchers from University of Pennsylvania have tried to tackle the issue of traffic congestion using what they refer to as "Congestion Prediction Networks," which try to dictate the traffic flow based on real time feedback from the streets in an algorithmic way [9] that aims to maximize road utilization, however this approach also requires a feedback mechanism which we do not have.

However, there have been some efforts in the local Bangladeshi market to tackle this problem as well. We have attempted to review the most prominent ones to get an understanding of their methodology.

2.1 GO! Traffic BD

This is a somewhat well-known application that has attempted to tackle this issue we are with limited success in the past. When our team tried to use the application just to get a feel for the application / environment, we ran into a few issues, all of which are documented and explained below.

a) Slow / Unresponsive UI

We got stuck several times in the log-in screen that required us to reset the entire app.

b) Data Availability

Due to reliance on a Facebook group for data that then has to be manually parsed and entered into the application, data availability remains a concern.

c) Data Quality

Contributing data is further complicated by having to find a Facebook group and posting there, this lowers the quality and frequency of the average submission.

2.2 FACEBOOK GROUPS

The other solution to the issue is usually volunteer run Facebook groups where people post updates manually as they travel. Unfortunately, it has the same problems as GO! Traffic due to methodology overlap. In addition, traffic updates of only the most frequent locations are posted and the users have to query in the groups to know about the traffic conditions in a specific region and wait for others to give the updates. Another problem is that the information is text-based, that is, the users have to read the posts to get an idea of the traffic conditions and there is a lack of a simple and intuitive visualization.

2.3 PROPOSED SOLUTION

Our proposed solution, rTraffic, aims to be free of all these issues. We have spent a considerable amount of time to ensure a lag-free experience throughout the application as long as there is Internet connectivity. Our data entry methods perform automatic validation and are available for indexing immediately by users to deal with data quality concerns. We are also implementing several "quality of life" features that make it as painless as possible to automatically contribute data since contribution complexity plays a big role in the willingness of users to submit data.

3. METHODOLOGIES

At the core of the application is the Haversine distance calculation formula, which is used to calculate the "great-circle distance" between two points on a sphere from their longitude and latitude data. The formula gets its name from the term "versed sine," which refers to a trigonometric function which defines that the versine of an angle equals one minus that angle's cosine. Haversine literally refers to "half a versine." The formula is given in its entirely below [10]:

$$hav\left(\frac{d}{r}\right) = hav(y_2 - y_1) + \cos(y_1)\cos(y_2)hav(x_2 - x_1) \quad (1)$$

where $hav(x)$ refers to the haversine function, and is defined as:

$$hav(x) = sin^2\left(\frac{x}{2}\right) = \frac{1-\cos x}{2} \quad (2)$$

Here, d is the variable in (1) that we solve for. It refers to the great circle distance between two points. r refers to the radius that we are searching within (defined as 5 kilometers for our application) and x / y variables refer to the latitude of point (1,2) and longitude of point (1,2) respectively. Once expanded further mathematically, we can solve for d.

$$d = rhav^{-1}(h) = 2r\arcsin(\sqrt{h}) \quad (3)$$

Here h refers to the earlier hav (d/r). Once fully expanded, the final formula is:

$$d = 2r\arcsin\sqrt{z} \quad (4)$$

where

$$z = sin^2\left(\frac{y_2-y_1}{2}\right) + \cos(y_1)\cos(y_2)\,sin^2\left(\frac{x_2-yx_1}{2}\right) \quad (5)$$

It uses this formula to calculate locations using longitude / latitude data that is a certain radius (5 kilometers) away from where the user is. This formula is implemented using Structured Query Language (SQL) by the application and is used to lookup surrounding points of interest every time a request comes in real time.

4. TECHNOLOGIES USED

The system is made up of two components - an Application Programming Interface (API) that implements the Representational State Transfer (RESTful) design paradigm, and an API consumer application that runs on the users' smartphones. The server providing the API also provides access to a database which is streamlined for fast storage and query of Global Positioning System (GPS) data. Several well-established web technologies are being used to build the components, a non-exhaustive list is shared in the next section.

5. SYSTEM DESIGN

As mentioned earlier, the system is in fact two separate components working together in harmony to implement a common goal. Their implementation details as well as the components that make up each application are shared in the subsequent subsections. The sections discuss the API and the Application separately for clarity. For the API server, we used the Laravel framework developed in PHP, which provides some of the frequently features such as views, routes etc. integrated with the framework which speeds up the development and testing phases of the system development [11].

5.1 SYSTEM COMPONENTS

5.1.1 API COMPONENTS

- Core Language – PHP 7
- Virtual Machine – Facebook HipHop
- Database Server – MariaDB 10.1 with Galera Cluster
- Framework – Laravel 5.3
- Cache Layer – Redis 3.2
- Web Server – Engine-X (“nginx”) mainline
- CDN – CloudFlare
- Push Notifications – Firebase Cloud Messaging
- E-mail Services – Mailgun
- Web Host – Amazon Web Services (AWS)

These components are tied together to build a RESTful Application Programming Interface that serves as a common gateway for information exchange between all API consumer applications on many platforms. We are using HTTP with TLSv1.2 as transport and Javascript Object Notation (JSON) as the “language” our API accepts and speaks. JSON was chosen due to ease of parsing (almost all programming languages offer libraries – some even native), and widespread use in API applications [12]. In addition, we can easily upgrade the various sub-components of our servers and client applications with the latest community-driven and enterprise packages because the mode of data transfer of all the popular packages is JSON.

5.1.2 ANDROID APPLICATION

- Core Language - Java with Android NDK
- Virtual Machine - Android Runtime (ART)
- Local Database - SQLite
- REST Client - Retrofit
- JSON Handler - Moshi (GSON)
- GPS Handler - Google Play Services
- Notifications - Firebase Cloud Messaging
- Object Relational Mapper (ORM) - ActiveAndroid

These components are tied together to implement a native, lag-free Android application that communicates with our API to act as an API consumer. The application is available in the form of an apk installation in Google Play Store.

5.2 SYSTEM ARCHITECTURE

In this section, we discuss a general overview of the system architecture of the server for handling API requests and the Android application which provides an interface to the users.

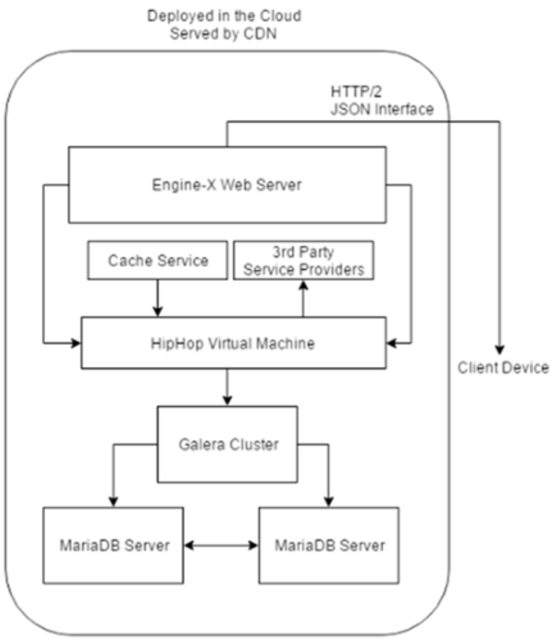


Figure 1: High-level Solution Layout for the API

5.2.1 APPLICATION PROGRAMMING INTERFACE

Figure 1 illustrates the inner workings of the individual API components. At the heart of the application is the HipHop Virtual Machine (HHVM), this is where our code is executed based on user requests and the results are then sent back. However, to do so, HHVM requires data and services from many other components.

For our data-store, we are using the MariaDB Database server. It was chosen instead of other SQL implementations mainly due to two reasons: i) ease of clustering for redundancy and scaling factors, ii) native support for geo-spatial data (which longitude and latitude points are). We make use of the ARIA Database Engine (implemented by MariaDB) to manipulate spatial data effectively [13]. However, one database server is a single point of failure for the application and ends up becoming a barrier to effective scaling and clustering.

To resolve this, we are deploying MariaDB alongside Galera - an application that makes it easier to cluster SQL database servers. It sets up a master-slave relationship where requests are load-balanced between slave based on policies. This gets us high performance alongside considerable reliability.

For the cache system, we are utilizing two different kinds of caching. Firstly, we rely on Just In Time (JIT) compilation features from HHVM [14] to run our code effectively - this forms the first layer of caching. We have a 2nd layer of caching for the data-store itself to cope with extremely high loads in the form of sharded redis servers. Any time an expensive query that repeats multiple time is run, the results are cached for a period of time. If the same query is made before the cache timer expires, we serve that request directly from the cache and do not have to re-do any processing.

There are some 3rd party services that the API also integrates with. Twilio is used to send SMS alerts for many operational as well as traffic status updates. The same updates are also available over Android Notifications (through Firebase) and E-mail (through Mailgun). All of these services are tied together and used to expose one common interface into the public internet through the use of the Engine-X ("nginx") web server. We had two options for web servers, one was the Apache HTTP server and the other one was NGINX server. We chose NGINX because it provides us the features to concurrently handle multiple requests and built-in support for load-balancing in the case of excess HTTP requests [15]. These features will help to upgrade our system when the number of users of our system will increase in the future.

The entire system runs on highly resilient infrastructure provided by Amazon Web Services (AWS) in Singapore [16]. We used an EC2 instance for hosting our server and a S2 instance to store the coordinates and user updates. We are also using a content delivery network (CDN) for performance acceleration and security purposes. We used Cloudflare as our CDN because it provides the feature to cache some the frequent queries online for a stated period of time so that the same requests are fetched from Cloudflare instead of making a query to our servers. The main advantage is that the frequent requests can be serviced for a brief period of time even if our servers go offline.

5.2.2 API CONSUMER - ANDROID APPLICATION

Figure 2 illustrates the inner workings of the client android application. We have chosen to design the application in a way that aims to lessen the load on the users' personal devices as much as possible. This is done for a myriad of performance as well as reliability reasons - namely to prevent our application from consuming too much resources (which is then reflected on the device's battery life), but also to keep the user experience as snazzy as possible.

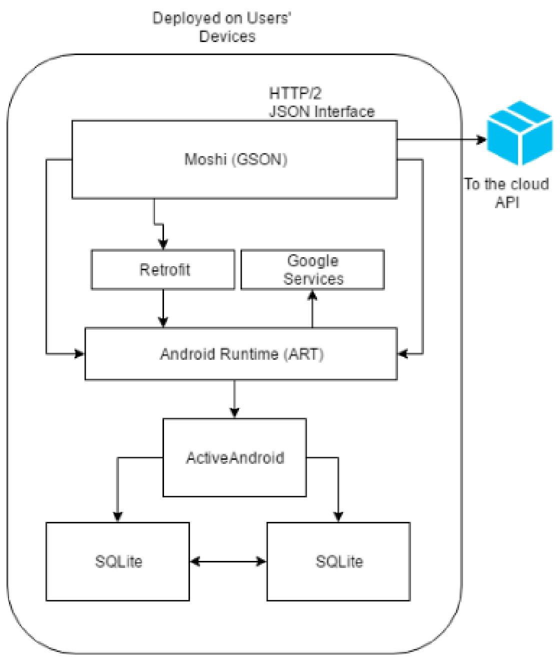


Figure 2: High-level Solution Layout for the Android Application

The application delegates authentication management and data aggregation / storage duties to the API. Since our API exclusively communicates using JSON, we first needed an way to be able to speak JSON from Java. This is where Google's GSON library comes in, it provides us with JSON Serialization / De-serialization services.

We also needed a HTTP client that could understand the RESTful design paradigm. For this, we chose Retrofit - a common and popular library for RESTful communications. By using this, we did not have to implement things like HTTP verb parsing, or header parsing or building interfaces for compression/decompression.

Local data storage on the user's device is in the form of SQLite, however the traditional ways of implementing SQLite in Java (via JDBC) can be quite cumbersome. Since we are using an object relational mapper (ORM) in the API itself, we also chose to use one in Android as well - namely ActiveAndroid. This allows us to manipulate data in the SQL datastore without having to write any SQL statements manually, we can simply access data in a object-oriented way instead. In addition, we opted to use SQL, instead of NoSQL databases, because we can make structured queries easily.

As our application makes extensive use of global positioning data (GPS), we also needed a robust way of reliably accessing the GPS subsystem in Android. This is where Google Play Services come in, they have built a nice abstraction layer that takes care of GPS initialization and triangulation for you - we can simply ask the library for the GPS coordinates. Google Cloud Messaging (Firebase) is used any time the API has to push a notification to the user's device. Firebase is a form of online database that provides fast real time access to data and is ideal for push notifications [17]. Google Maps is used to make sense of the surroundings around users as well as to show real time updates via overlaying. All of these components are tied together by the Android Runtime (ART) to implement one application.

6. API OPERATIONAL WORKFLOW

This section aims to explain how the various API methods actually work under-the-hood. Full implementation level details are given. A high level request flow diagram can be seen in figure 3. Each subsection explains a HTTP Verb (Request Type), and Endpoint pair. Variables enclosed in [] are required, variables enclosed in [?] are optional. HTTP Verb can be one of GET, POST, PUT or DELETE. A special verb called CONSOLE will be used to indicate that this action is not triggered by a user, but automatically by the application itself.

6.1 GET /api/v1/points/[latitude?]/[longitude?]

By default, this endpoint returns a list of all registered intersections/points in the system. Users however also have an option of filtering the results by providing the latitude and longitude parameters. If the parameters are present, then the system tries to locate relevant points of interest (from an user's perspective) which are typically major traffic intersections within a 5 kilometer radius. It is expected that the longitude and latitude values given are from the user's current location.

First, the data is validated to ensure that the given parameters are in fact valid longitude and latitude coordinates by checking if they fall within acceptable ranges. We then check if we have cached results for this (longitude, latitude) pair in our cache service. If we do, we directly move to the JSON serialization step. If we don't, the long/lat values are converted into radians and plugged into the Haversine distance calculation formula which is run on all location entries in the database. The database returns a set of rows that match the distance requirements. This data is then serialized into JSON and sent off to the requester over an encrypted HTTP session. We also insert a copy of the data into our cache servers so we can serve this exact query by another client without having to calculate the Haversine distance between eligible points in our database. Since traffic architecture is fairly static, we feel that it works well.

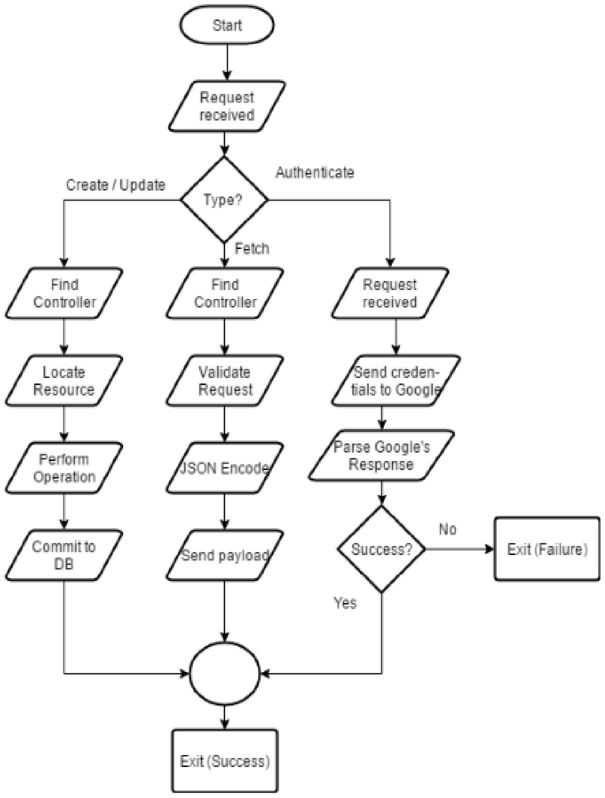
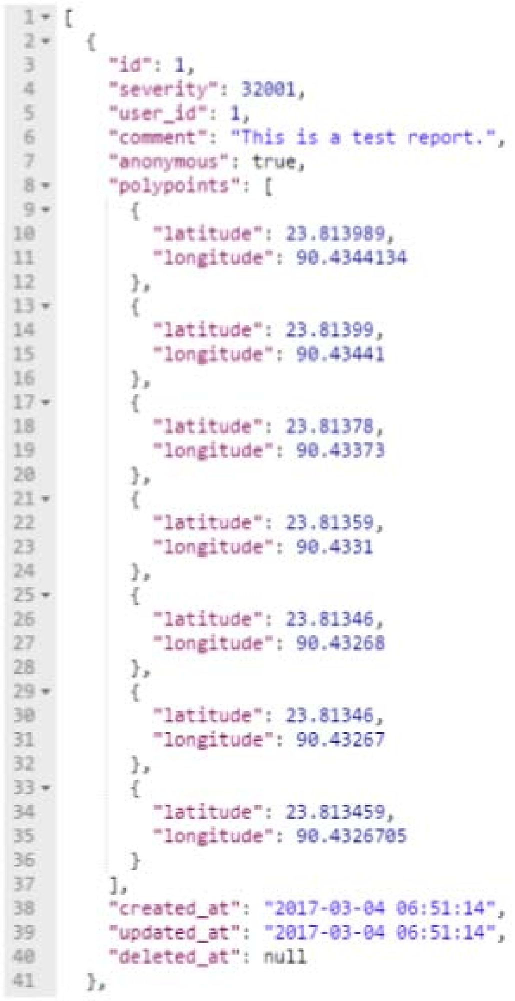


Figure 3: API Request Flow Depending on Request Type

6.2 GET /api/v1/reports/[id?]/[history?]

This endpoint by default returns all traffic status update reports submitted in the last hour. Users however have an option of narrowing down the results that concern a specific point only by specifying the optional ID parameter. If the history flag is set to an acceptable value (between 1 hour and 24 hours), then historical reports will also be visible on the generated output. Figure 4 shows a snapshot of the json response provided by the server in the response body when a request is sent to this endpoint.



```
[
  {
    "id": 1,
    "severity": 32001,
    "user_id": 1,
    "comment": "This is a test report.",
    "anonymous": true,
    "polypoints": [
      {
        "latitude": 23.813989,
        "longitude": 90.4344134
      },
      {
        "latitude": 23.81399,
        "longitude": 90.43441
      },
      {
        "latitude": 23.81378,
        "longitude": 90.43373
      },
      {
        "latitude": 23.81359,
        "longitude": 90.4331
      },
      {
        "latitude": 23.81346,
        "longitude": 90.43268
      },
      {
        "latitude": 23.81346,
        "longitude": 90.43267
      },
      {
        "latitude": 23.813459,
        "longitude": 90.4326705
      }
    ],
    "created_at": "2017-03-04 06:51:14",
    "updated_at": "2017-03-04 06:51:14",
    "deleted_at": null
  },
```

Figure 4: API Request Flow Depending on Request Type

6.3 GET /api/v1/reports/point/[id?]/[history?]

This endpoint is meant to return a list of all traffic status update reports that were contributed in the last hour by default about a specific point. If the history flag is set to an acceptable value (between 1 hour and 24 hours), then historical reports will also be visible on the generated output.

The very first thing that we check for is that whether id is numeric and positive - if it is, we consider it valid and pass it onto the database engine to look up results. The results are cached but with a 1 minute interval (since the nature of the data is so dynamic). If a cache-miss occurs, we go back and fetch the data from the database. If not, it is served directly from the cache. Once we have the data, we serialize it in JSON and send it to the client for further parsing.

6.4 GET /api/v1/reports/geom/[long]/[lat]/[history?]

This method does the exact same thing as "GET /api/v1/reports/point/[id]/[history?]," except the user does not need to know the ID associated with a location. They can simply look it up using longitude and latitude. This lookup process includes a "tolerance" of 100 meters, as in: we classify the request to be about a specific point if the longitude/latitude pair given is within 100 meters of that point. This is done because pinpoint accuracy is not possible to be achieved using consumer GPS implementations.

6.5 POST /api/v1/reports

This method deals with the creation of reports (both notification based and manual). In the HTTP POSTFIELDS, a JSON encoded request body is expected. The first thing we thus check for is whether the request body is properly populated and it contains valid JSON data that we can deserialize. Once we have valid information, we then check the report parameters for validity and if it passes this check, it is finally committed into the database. The validity check is meant to ensure that a few constraints are applied on all committed data. We check that a report does not include too many intersections (currently, reports are rejected if they contain more than 2 intersections to prevent spurious or bogus reporting). We also check that all required parameters are present.

Submitter reputation may be checked before the final step however, if one has been found to be submitting bogus reports - the submission may be rejected. There are different policies that can be tested to check reputation such as frequency of posts and automatic detection of spam in the user updates. Once everything is done, a success message is sent back to the user thanking him for his contribution.

6.6 PUT /api/v1/reports/[id]

This method allows the user to edit a previously submitted report (within the same hour). Report ownership is checked before the edited reports are committed into the database.

6.7 DELETE /api/v1/reports/[id]

This method allows the user to delete a previously submitted report (within the same day). Report ownership is checked before the deletion may proceed.

6.8 GET /api/v1/oauth?[code?][error?][provider?]

This endpoint handles users logging into the application using credentials belonging to 3rd party service providers like Google or Facebook. Non OAuth users are currently not supported. In our system, we implemented the three legged oauth verification. The client application makes requests to this endpoint specifying the OAuth service provider and the OAuth authorization code. The API reaches out to that provider using that authorization code and asks them to validate it. If everything goes well, the user is granted a session and logged in. Once this is done, an unique access token is generated and sent back to the user. The API expects all further requests from that user to include this API token for them to be considered valid. If not, an error message is sent back to the user.

6.9 GET /api/v1/excluded-regions

This endpoint returns a JSON encoded list of all the excluded regions that an user has registered. We first check if the request contains a valid authentication token and then look up the user's details via this token. The excluded regions are then queried from the database using this user reference. If it is found that the user has indeed registered excluded regions before, the results are JSON encoded and returned. If not, an empty array is returned.

However, if API authentication fails, an error message is returned instead.

6.10 POST /api/v1/excluded-regions

This endpoint deals with the creation of a new excluded region. A latitude/longitude pair and a canonical name are expected to be within the JSON serialized request body. If all parameters are found to be present and API token is valid, the new entry is added to the database and a success message containing the ID of the newly created region is returned to the user.

In case of any error, the user is notified of such error by the API.

Journal of Theoretical and Applied Information Technology
15th June 2018. Vol.96. No 11
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

6.11 DELETE /api/v1/excluded-regions/[id]

This endpoint handles the deletion of an excluded region from an user's account. If a numeric and positive ID parameter is given, we first validate whether the resource referenced by the ID actually exists in the database. If it does, we then check that the API token provided alongside the request actually has authorization to perform this deletion request.

If authorization is confirmed, we remove the record from the database and return a success message in the form of a JSON encoded payload to the user.

6.12 GET /api/v1/poi

This endpoint returns a JSON encoded list of all the points that the user has registered to receive real-time notifications about. We first check if the request contains a valid authentication token and then look up the user's details via this token. The points of interest are then queried from the database using this user reference. If it is found that the user has indeed registered points of interest before, the results are JSON encoded and returned. If not, an empty array is returned.

However, if API authentication fails, an error message is returned instead which is handled appropriately and the user is notified about the request failure.

6.13 POST /api/v1/poi

This endpoint deals with the creation of a new point of interest. A valid ID belonging to any of the points currently in the database is expected to be within the request body. If all parameters are found to be present and API token is valid, the new entry is added to the database and a success message is returned to the user. In case of any error, the user is notified of such error by the API.

6.14 DELETE /api/v1/poi/[id]

This endpoint handles the deletion of an excluded region from an user's account. If a numeric and positive ID parameter is given, we first validate whether the resource referenced by the ID actually exists in the database and that the user has actually designated it as a point of interest. If authorization is confirmed, we remove the record from the database and return a success message in the form of a JSON encoded payload to the user.

6.15 CONSOLE /api/v1/automate

This is a pseudo endpoint that is used for all the automation. Any notifications of any type to be sent to users is generated by this endpoint (behind the scenes). This is where traffic congestion notifications are generated based on user preferences and sent out to users via Firebase post aggregation. Any pre-aggregation or preemptive caching tasks are also handled by this endpoint since it is automatically invoked every minute. When this is done, any task that might have been queued earlier for background processing (to make API responses faster) are also gradually de-queued and processed.

7. OPERATIONAL WORKFLOW

The application is built using components of Google's material design and tries to follow all design requirements strictly. This means that rTraffic is visually similar to many native Android applications by Google or other publishers and users can fully expected "gestures" that they are used to function just as effectively in our application. One example of such gestures could be the presence of a navigation drawer that pops out anytime the application's home button is clicked, or a toolbar-based UI.

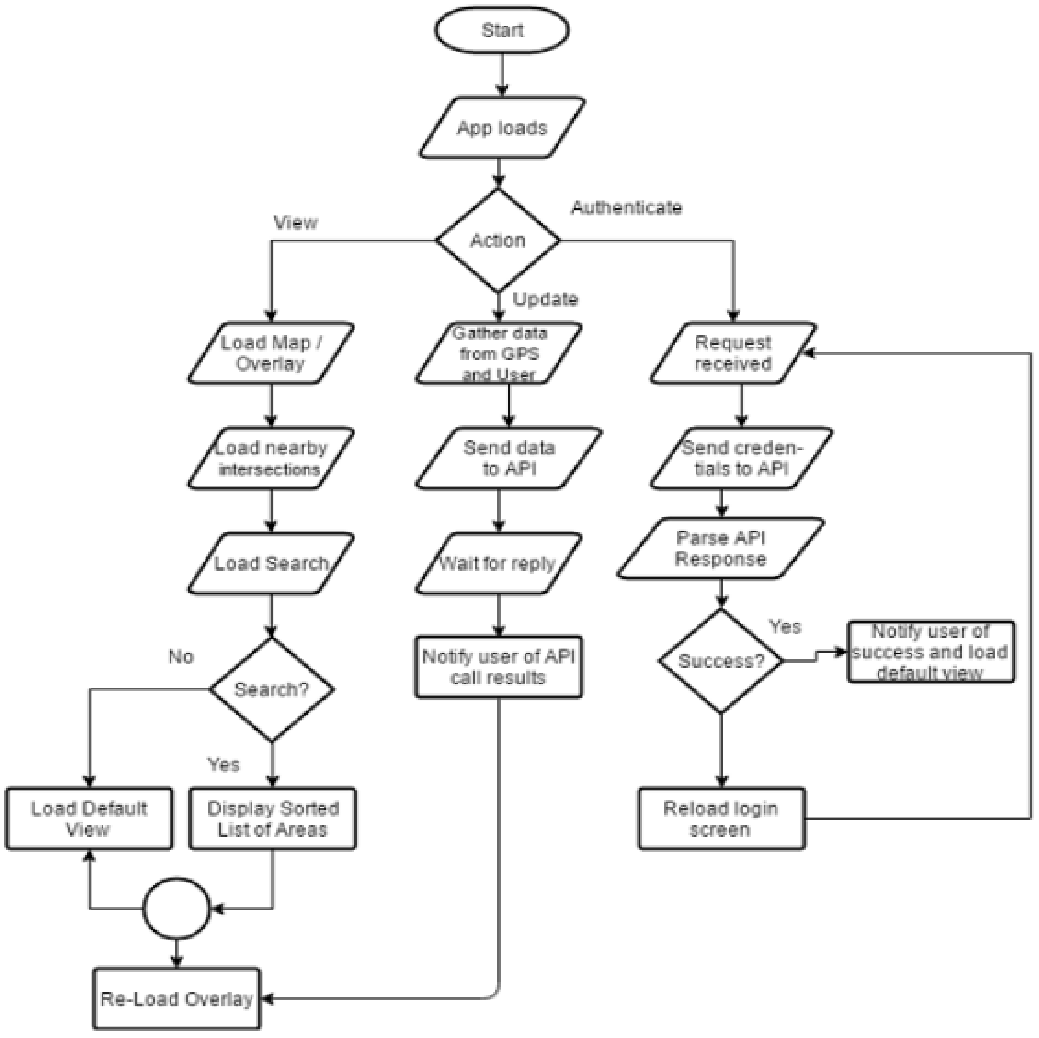


Fig. 5. App Request Flow Depending on Action Type

Latter subsections aim to explain how every implemented feature on the application actually works behind the scenes. Figure 5 aims to break this down in a graphical way.

7.1 Android Application Structure

In android architecture, a specific page of an application is referred to as an "activity." Specific use cases or application scenarios are implemented as activities that are paired with an XML layout defining the design of that page.

The activity itself is defined as a java class file which controls the behavior of the elements defined in the XML file. The control flow of the activities and the activity to take control at application startup is defined in a special XML file called AndroidManifest.xml. When an activity takes control of the application, it passes through different stages of its life-cycle determined by the user's input and may pause or terminate its life-cycle and pass control to other activities.

7.2 Use Cases And Features Of The Application

The following lists the generalized use cases of the many features of the client application described on a per activity basis. In android development terminologies, an activity class is dedicated to handle the logic involved in each of these user activities.

7.2.1 Welcome

The welcome activity is run only once after the user installs rTraffic on their phone. Its goal is to inform the user of the application's features and many use cases. Essentially, it is a list of 5 presentation slides that the user may "swipe" through to learn more about the application's various functionalities. Once the user is finished going through the slides, execution control is handed over to the authentication flow described in the next subsection.

7.2.2 Authentication

Authentication is the first "activity" called once the application has successfully started up. As we are authenticating against Google (instead of a local user database exclusive to our application), the first thing we check for is whether Google's authentication services API is available and reachable from the client application. If this check succeeds, we simply instruct the user to choose from a list of Google accounts that are currently active on the Android device. Users may choose to add a new Google account to the system from this window as well if they do not have any accounts already registered to the device, or if they would like to use a separate account for rTraffic.

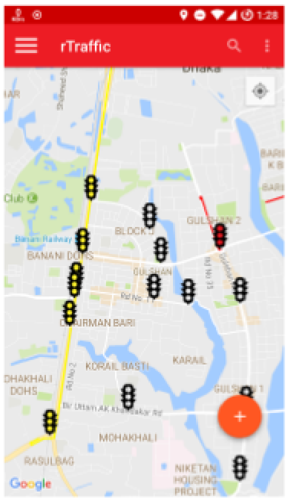
Once the user chooses the account that they would like to use, a request is formed and sent to Google to validate the user's authenticity. If Google responds positively to this API request, the user's authentication state is internally updated and they are then redirected to the "Main Activity". The API returns an authentication token at the end of this process and all subsequent requests are expected to be signed using this token. If a request is sent without this token, the API refuses to service said request. This token has an expiry date and the server requests to the Google's authentication services if the token expires during the user's usage period of the system.

However, if Google denies the authentication request, the user is notified that their log-in attempt has failed and given the option to try again. It is worth noting that we import the user's name, e-mail address and current avatar from their Google account. Any changes made on the account will also show up in our application.

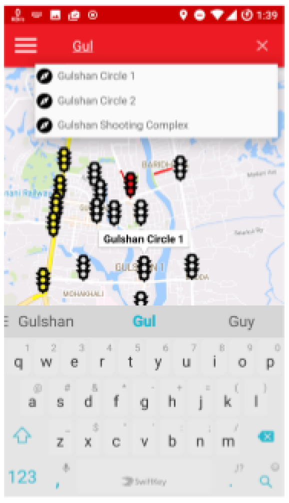
7.2.3 Main Activity

The main activity features a full-screen map where traffic conditions are represented using various colorful markers and drawings across the roads. Key intersections are marked with 3 vertical dots that can change color to indicate the traffic condition of that intersection. We support 3 states - namely, Un-congested, Congested and Slow but Moving. Based on the current state of a location, the 3 vertical dots also change color to Green, Red and Yellow (respectively). A special color (white) is used to indicate the lack of data about a point. It is expected that most users wishing to consume the data provided by rTraffic will be using this activity the most.

Individual users' reports are visualized on the map as "polylines" that go along roads or highways. The term polyline refers to "polygonal lines." This is what Google calls lines that follow a road or a specific area. These are also color-coded in the same way as the intersection status indicators. The intersections' status is derived from the status of the polylines passing through them, in fact. So to say, if a polyline with the status set to "congested" passes through an intersection, that intersection's status will also be set to "congested," and it will change its color to red to reflect this change. Users may also click on intersection markers to read comments contributed by users when submitting reports that go through the intersections. Users may also utilize the search module to find an intersection quickly. Figure 6(a) shows the Main Activity and figure 6(b) shows the Search Module that is accessible from the toolbar.



(a) Main View



(b) Search View

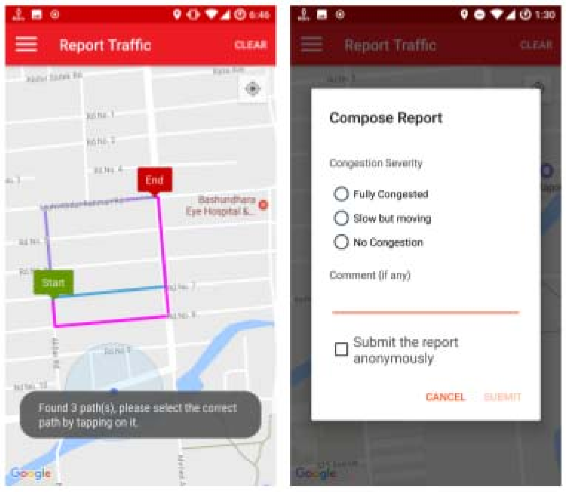
Figure 6: Main Activity

7.2.4 Traffic Reporting

This activity incorporates a full-screen map much like the main activity. Once loaded, it tells the user to long-tap at the starting point of the location in the map that they intend to submit a report for. Once they do so, the app prompts them to similarly tap at the ending point of their reported location.

Once this information is provided, the application consults the Google Maps directions API to find all possible paths between these two points that go through publicly accessible roads. The user is then instructed to choose one of them and confirm their intent of submitting a report. The user may choose to start over by pressing the "Clear" button in case of malformed input being accidentally provided.

Once they confirm that everything is alright, a new window pops up that asks the user to select the severity of the traffic condition in the region. They may also provide an optional comment at this stage. The option to anonymously submit a report is also provided here. Once all data is provided, the application consults the back-end API to submit the report and thanks the user. All of this data is encapsulated into a "report" and also added to the main map in the Main Activity. Figure 7(a) and figure 7(b) show the visual representation of this activity.



Found 3 path(s), please select the correct path by tapping on it.

(a) Path Chooser (b) Report Confirmation

Figure 7: Traffic Report Activity

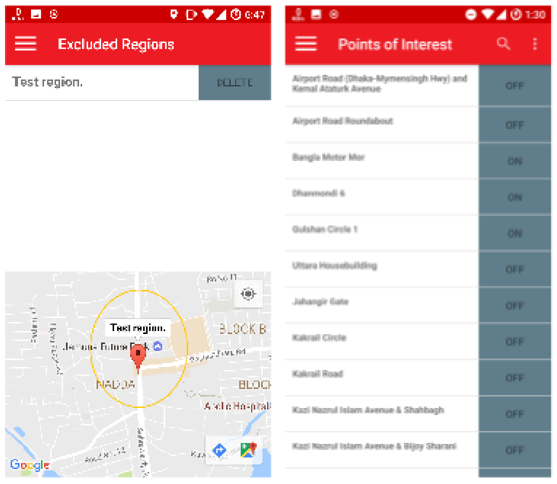
7.2.5 Account Management

Users may invoke this activity from the navigation drawer. Its purpose is to allow an user to sign out and change the Google account that they may be using rTraffic with at present. Users also have the option of revoking all previously granted permissions to rTraffic on their Google account via a special "Disconnect" button. Both actions are performed by making API requests to Google with previously saved access tokens and updating the user's authentication state based on response from Google.

7.2.6 Excluded Regions

Users may register arbitrary locations as "excluded." What this means is that the application's background data-collection services as well as real-time notification services will all be suppressed if it detects that the user is very close to or inside of an area that they had designated as excluded. The primary motivation behind implementing this feature was to give users more control to the application's behaviors. For example: an user might live next to a highway and thus receive requests to contribute data on a regular basis. The user may not appreciate such requests, and this feature allows the user to suppress the application from ever bothering him inside his own home.

Users may mark a region as excluded by long-tapping that location on the full-screen map of this activity. Once done, they will be asked to provide a name for this region and it will be committed to the API server as well as the local, internal database. Users may also remove any previously registered excluded regions from this activity by pressing the "Delete" button associated with that region. A visual representation of this activity may be seen at figure 8(a).



Test region.	DELETE

Airport Road (Dhaka-Mymensingh Hwy) and Kemal Ataturk Avenue	OFF
Airport Road Roundabout	OFF
Bangla Motor Mor	ON
Dhanmondi 6	ON
Gulshan Circle 1	ON
Uttara Housebuilding	OFF
Jahangir Gate	OFF
Kakrail Circle	OFF
Kakrail Road	OFF
Kazi Nazrul Islam Avenue & Shahbagh	OFF
Kazi Nazrul Islam Avenue & Bijoy Sharani	OFF

(a) Excluded Regions (b) Points of Interest

Figure 8: Excluded Regions and Points of Interest Activities

7.2.7 Point Of Interest

This activity allows the user to register for real-time updates about specific intersections. The user interface consists of a list of all intersections that the application knows about, each row of the list contains a button that may be toggled to express interest. When the toggle button is clicked, a request is made to the API back-end server to keep track of the users' interest or disinterest. The API will send any and all reports concerning these intersections as Firebase push notifications to this user's device. We chose to require users to register to receive push notifications because otherwise the amount of notifications that we have to push might end up overwhelming the user.

A search functionality is also available to allow users to quickly find the intersections that they are interested in. A visual representation of this activity may be seen at figure 8(b). Here, the labels 'ON' and 'OFF" at the right corner of each intersection indicates whether the system should send updates regarding the specific intersection.

7.2.8 Settings

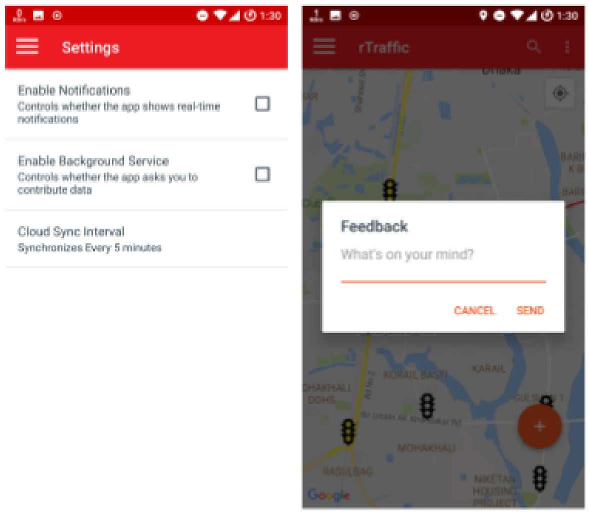
This activity has some crucial application settings that allow the user to customize the application's behavior to their liking. Notably, the user may toggle the background service that prompts them for data collection on and off here. Users may also centrally disable all notifications here and choose an agreeable synchronization period for the API (defaults to every 5 minutes). A visual representation of this activity may be seen at figure 9(a).

7.2.9 Help

This activity features a list of "Frequently Asked Questions (FAQ)" that the user may go through to learn more about the application. We have also tried to put justifications behind the various device permissions that we require from the user to access facilities such as GPS/Internet here. The help contents are not stored alongside the application in the user's device, but rather retrieved dynamically from the back-end system. For this reason, it is possible for us to update the help content without updating the application.

7.2.10 Feedback

Users may choose the feedback option from the navigation drawer anytime they choose to send feedback to us. We are using the Doorbell feedback management system which allows to get a description of which activity the user was using as well as what they were trying to do when they ran into issues and chose to submit feedback. We can then reply back to the feedback as needed. A visual representation of this activity may be seen at figure 9(b).


Settings

Enable Notifications
Controls whether the app shows real-time notifications

Enable Background Service
Controls whether the app asks you to contribute data

Cloud Sync Interval
Synchronizes Every 5 minutes

Feedback
What's on your mind?

(a) App Settings (b) Feedback UI

Figure 9: Application Settings and Feedback Activities

7.3 Background Services

The application also runs a number of background services that run on the user's' devices indefinitely and are designed to handle events when the application is dormant or perhaps not running at all. These services are designed to be as lightweight as possible which means that their memory and power usage do not hamper the user experience. This section is meant for describing them.

7.3.1 Real-Time Notification Services

This service consumes the Firebase notifications sent to the application and updates the internal database(s) accordingly. For example: If the service is notified that a new report has been received by the API, it reaches out to the API and downloads that report. It then draws that report atop the main map of the application. Once this is all done, it checks (to double confirm) that this report concerns a location that the user had previously "shown interest" in (via the Points of Interest UI). If that check comes back as positive, it generates a notification to alert the user on the traffic status change.

This service also handles informational notifications that we can send to the user announcing changes to the app or any generic information broadcast.

7.3.2 Traffic Status Services

This service is invoked every n minutes (n can be altered by the user in the settings activity) and is meant to determine whether a user should be prompted to contribute traffic status data to the application or not. It first determines whether internet access and GPS access is available to us. If yes, it then locates the user's longitude/latitude via the GPS and reaches out to the Google Geocoding API to convert that to a human readable address. It then checks the address to determine whether the user is currently on a road or not. If positive, user's current movement speed is profiled (to determine if a possible traffic jam is in play). If the movement speed is determined to be less than 10 kilometers an hour while on a road, a notification is generated asking the user to confirm whether they are stuck in a traffic jam.

The user has the option of dismissing the notification or agreeing to contribute data. If they respond positively, the user is taken directly to the "Traffic Reporting" activity with the GPS focus on their current location.

8. RESULTS AND DISCUSSIONS

We have implemented an API back-end application as well as a native Android application following the system design that we described in the previous sections. The application is presently available on the Google Play Store (Android marketplace). Any user with a suitable Android device can simply search for 'rTraffic' in Google Play Store to download it.

The general user interface of the application is shown in figure 6 - 9. We have taken care to preserve precious system resources as much as possible, the application is designed to be fast and lightweight. This section includes some performance benchmark results from the solution that we built.

8.1 Benchmarking Api Performance

APIs (and web properties in general) are benchmarked in terms of how much time it takes them to service a single request and how many requests the system is capable of servicing in a second. We have benchmarked the rTraffic web API accordingly, the results can be seen in the table below. The tests were performed using the industry standard ApacheBench tool. It is worth mentioning that the system where the API was deployed at the time had the following hardware specifications: Intel Core i7 3770k CPU, 32GB DDR3 RAM and a 512GB SSD.

The test only included those endpoints that would potentially receive the most load. The first column defines the endpoint and the HTTP request verb. The second column is the amount of requests that the API answered within a second while the third column is the (average) time taken to service a single request. It should be noted that network latency is ignored in this benchmark as the testing was performed from the same machine where the API service was running. The results in table 1 are from a single non-load balanced instance of the API server, it is possible to increase performance exponentially by switching to a load balanced design as needed.

Table 1: API Benchmark Results

API Endpoint	Requests Per Second	Time Per Request
points(GET)	852	1.175 ms
reports(GET)	799	1.251 ms
reports(POST)	621	1.457 ms
excluded-regions (GET)	904	1.107 ms
excluded-regions (POST)	653	1.394 ms
poi (GET)	820	1.210 ms
poi(POST)	580	1.642 ms

8.2 Android Application - Resource Usage

Unfortunately, due to how fragmented the Android ecosystem is, benchmarking applications yield subjective rather than objective results. The following results were obtained from a Samsung Galaxy Note 3 device through the use of the Android Device Monitor. It should be noted that resource usage and performance can vary wildly from device to device due to how vendors can customize the systems.

Table 2: Resource Benchmark Results

Metric	Measurement
CPU Usage (active)	30%
CPU Usage (background)	less than 5%
RAM Usage	46MB
Network I/O (init)	5MB
Battery Impact (active)	10% per hour
Battery Impact (background)	determined by config

9. FUTURE WORK

The objective of our endeavor is to mitigate the traffic congestion in a major city and help users

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

plan their schedules and travel paths in order to save time and money. The current system works solely based on user contributed data. We have plans to implement pattern recognition capable of actually predicting traffic conditions using machine learning or artificial intelligence algorithms in a future version.

10. CONCLUSION

In this paper, we have demonstrated the methodology to develop an application that collects traffic updates from the users and present the data in the form of relevant notifications and structured visualization on the map of Dhaka city. We have experimented the usage of the application on a daily basis and satisfied with the application's consumption of mobile resources and the speed of receiving real time updates. While we believe our method is technically superior to all existing implementations within the country at the time of writing, success will only be possible if a significant amount of adopters use the app and contribute data towards the service.

REFRENCES:

[1] "Per capita income rises to $1466", *The Daily Star*, 2016. [Online]. Available: http://www.thedailystar.net/frontpage/capita-income-rises-1466-1204930. [Accessed: 28- Nov- 2016].

[2] "2015 Auto Industry Trends In Emerging Markets - Carmudi BD", *2015 Auto Industry Trends In Emerging Markets - Carmudi BD.*, 2015. [Online]. Available: http://www.carmudi.com.bd/research. [Accessed: 28- Nov- 2016].

[3] J. Rosen, "The Bangladeshi Traffic Jam That Never Ends", *The New York Times*, 2016. [Online]. Available: http://www.nytimes.com/2016/09/23/t-magazine/travel/dhaka-bangladesh-traffic.html. [Accessed: 28- Nov- 2016].

[4] K. Mahmud, K. Gope, S. Mustafizur and S. Chowdhury, "Possible Causes & Solutions of Traffic Jam in Dhaka City", *Journal of Management and Sustainability*, vol. 2, no. 2, 2012.

[5] K. Nasrin, R. Jonathan, "Urban development and livelihoods of the poor in Dhaka", 2005.

[6] S. Paul, P. D. Nath, N. M. A. Sattar and H. U. Zaman, "rTraffic - a realtime web application for traffic status update in the streets of Bangladesh", *2017 International Conference on Research and Innovation in Information Systems (ICRIIS)*, Langkawi Island, Malaysia, 16-17 July, 2017.

[7] "Stuck in traffic?", *Official Google Blog* , 2007. [Online]. Available: https://googleblog.blogspot.com/2007/02/stuck-in-traffic.html. [Accessed: 28- Nov- 2016].

[8] C. L. David, "Traffic lights: There's a better way", *MIT News*. [Online]. Available: https://news.mit.edu/2014/traffic-lights-theres-a-better-way-0707. [Accessed: 28- Nov- 2016].

[9] M. Rahul, L. Insup, and S. Oleg, "Real-Time Traffic Congestion Prediction", *NSF-NCO/NITRD National Workshop on High Confidence Transportation Cyber-Physical Systems*, November 2008.

[10] Van Brummelen, Glen Robert, "*Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry*", 2013, Princeton University Press. ISBN 9780691148922. 0691148929. Retrieved 2015-11-10.

[11] "Laravel", laravel.com [online]. Available: https://laravel.com

[12] C. Douglas, "The application/json Media Type for JavaScript Object Notation (JSON)", *JSON.org*, July, 2006.

[13] "SPATIAL INDEX", *MariaDB KnowledgeBase*.[Online]. Available: https://mariadb.com/kb/en/mariadb/spatial-index. [Accessed: 28- Nov- 2016].

[14] "Faster and Cheaper: The Evolution of the hhvm JIT", *HHVM*. [Online]. Available: http://hhvm.com/blog/2027/faster-and-cheaper-the-evolution-of-the-hhvm-jit.[Accessed: 28- Nov- 2016].

[15] "nginx", *www.nginx.com* [online]. Available: https://www.nginx.com

[16] "Cloud Products", *aws.amazon.com* [online]. Available: https://aws.amazon.com/products

[17] "Firebase", *firebase.google.com* [online]. Available: https://firebase.google.com