Vehicular network for incremental route planning and avoiding traffic congestion

Manish Kumar,
Y9227319
Department: CSE

Vempati Anurag Sai
Y9227645
Department: EE

Supervisor: Dr. R.K Ghosh

*Abstract:*We intend to develop an efficient navigation system through the busy traffic. Google maps provide us with an up to date traffic update. But, many a times there are unforeseen situations (such as, rallies, maintenance work, landslides etc.) that alter the traffic instantaneously. We try to incorporate such situations into our mechanism and reroute the traffic through an alternate route. We heavily rely on traffic data provided by google and on top of that, we maintain a server which contains critical information about various situations mentioned above. A vehicular network is created by which each mobile user can send prior information regarding these situations to the server.

INTRODUCTION

In the modern age it is equally important that we reach correct destination and we get there right on time. Google has radically changed an individual's life with their innovative maps. They provide direction, traffic, estimated travel time and what not. Our project is all about making it more productive for the user. We develop a vehicular network which helps in providing personalized time estimates, inter-user communication through a centralized server and incremental route planning. We made use of Google Maps API in Android v4.2.

MECHANISM

I. *Getting Route Data:*

The user is first queried regarding the source and destination points. Google maps API is then used to fetch route data in the form of a .JSON file. This file is a hierarchy of various collections of location coordinates. The top most in this hierarchy is, set of 'routes' between the source and destination. Each route then consists of set of 'legs' that make this particular route. Each leg in turn consists of various 'steps' that make this particular leg. It also contains travel time to cover that leg at an average (universal) driving speed "just" based on distance (this estimate doesn't consider current traffic). The .JSON file also returns a 'polyline' for each leg which essentially is a dense set of points covering that leg. This polyline is used in drawing the path.

So, basically a .JSON file looks something like this:

```
Route1
    Leg1
            Time, start, end, polylines
            Step1
                    <Set of latitudes and longitudes>
            Step2
            :
    Leg2
            Time, start, end, polylines
            Step1
                    <Set of latitudes and longitudes>
            Step2
            :
```

    :
    :
Route2
:

An example .JSON file:

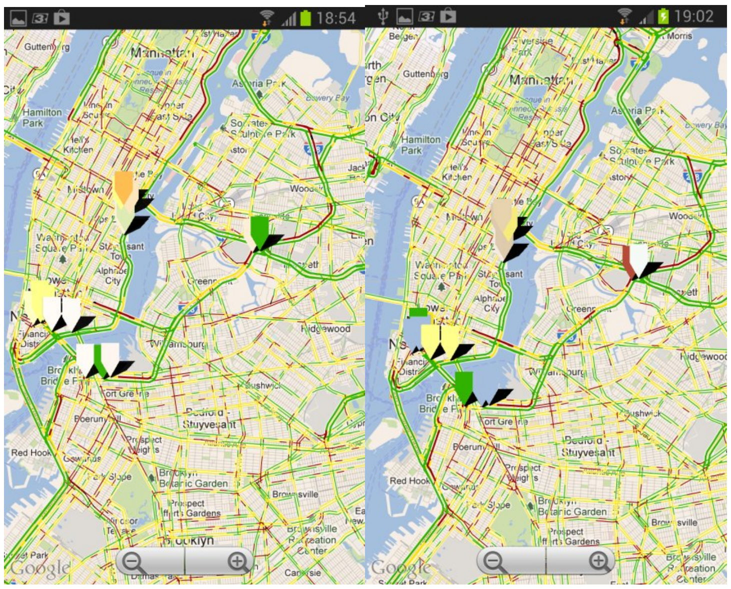
```
{
   "routes" : [
      {
         "bounds" : {
            "northeast" : {
               "lat" : 31.668980,
               "lng" : 74.89889000000001
            },
            "southwest" : {
               "lat" : 31.629340,
               "lng" : 74.87768000000001
            }
         },
         "copyrights" : "Map data ©2013 Google",
         "legs" : [
            {
               "distance" : {
                  "text" : "5.6 km",
                  "value" : 5570
               },
               "duration" : {
                  "text" : "13 mins",
                  "value" : 801
               },
               "end_address" : "Majitha Road, Amritsar, Punjab 143001, India",
               "end_location" : {
                  "lat" : 31.66830000000001,
                  "lng" : 74.898870
               },
               "start_address" : "Hall Bazar, Amritsar, Punjab, India",
               "start_location" : {
                  "lat" : 31.629340,
                  "lng" : 74.87768000000001
               },
               "steps" : [
                  {
                     "distance" : {
                        "text" : "0.1 km",
                        "value" : 104
                     },
                     "duration" : {
                        "text" : "1 min",
                        "value" : 11
                     },
                     "end_location" : {
                        "lat" : 31.630210,
                        "lng" : 74.87807000000001
                     },
                     "html_instructions" : "Head \u003cb\u003enortheast\u003c/b\u003e",
                     "polyline" : {
                        "points" : "kr``Eop_hMg@UEAmAa@ICg@O"
                     },
                     "start_location" : {
                        "lat" : 31.629340,
                        "lng" : 74.87768000000001
                     },
                     "travel_mode" : "DRIVING"
                  },
               ],
               "via_waypoint" : []
            }
         ],
         "overview_polyline" : {
            "points" :
"kr``Eop_hMm@WwAe@g@OMh@}@]ADAFCFEDGASMKY?_@i@EmESaDMqOq@GEAK_@iCk@iCOiBg@uAWa@a@c@OIiA{@aAu@q
@e@gAk@oBaAeCo@yDi@_BUmI}AuGkAgHuAkCm@{KgCoFkAcFiA}NeDqMsCcE_AuA[KMeFkAuBa@y@UqFoAgH_B_MqC_@Om
AUcCk@mAY_GcB_Cq@m@YsHaD]OBIfBeEn@}Ah@aBj@cBNmBBe@z@AJADEAyB@eIAeAG?aEB"
```

```
         },
         "summary" : "Majitha Rd",
         "warnings" : [],
         "waypoint_order" : []
      }
   ],
   "status" : "OK"
}
```

This .JSON file is then parsed and the relevant data like latitudes, longitudes, time and polylines are stored into the database.

II. *Getting Traffic Information:*

Since, Google doesn't directly provide traffic data, we have to read the colour from the path to get the traffic estimate. For each of the routes in database, we read the colour value at each and every location specified by the set of latitudes and longitudes comprising the routes. The colour that's read is very sensitive to variation. i.e., if the latitude/longitude shifts a little off the path, it reads completely different colour. So, we sampled at large number of points nearby and considered only the ones that match to one of the traffic colour codes (Red, Yellow, Green etc.).



Marker colour indicates the Traffic colour at that location at different times

Depending on the colour, we estimated time between every two consecutive coordinates (a 'step') as follows:

Time = Distance between the coordinates/ Avg. speed

Now, the average speed depends on the colour value at that location. Google gives following values for each of the colours:

- Green: more than 50 mph or 80 kmph
- Yellow: 25 - 50 mph or 40 - 80 kmph
- Red: less than 25 mph or 40 kmph
- Red/Black: very slow, stop-and-go traffic
- Gray: no data currently available

These values are rough estimates and vary with location. So, we tried to come up with a personalized speed values. Since we track the location of the user, we can calculate his average speed in each of the above traffic categories. With every new input, the means are updated as follows:

```
For each step in a route {
        If colour == Green{
                $Mean_G$= ($Mean_G$*$N_G$ + current_speed) / ($N_G$+1);
                $N_G$++;
        }
        else, if colour == Yellow{
                $Mean_Y$  =  ($Mean_Y$*$N_Y$ + current_speed) / ($N_Y$ +1);
                $N_Y$ ++;
        }
        else, if colour == Red{
                $Mean_R$= ($Mean_R$*$N_Y$ + current_speed) / ($N_Y$ +1);
                $N_Y$ ++;
        }
        else {
                $Mean_{RB}$= ($Mean_{RB}$*$N_{RG}$ + current_speed) / ($N_{RG}$ +1);
                $N_{RG}$ ++;
        }
}
```

Where, N_G is number of steps with colour Green, $Mean_G$ is average speed of the user in Green Traffic areas and so on.
Once we have time for each step, time for a 'leg' is calculated by adding up all the individual times for its steps.

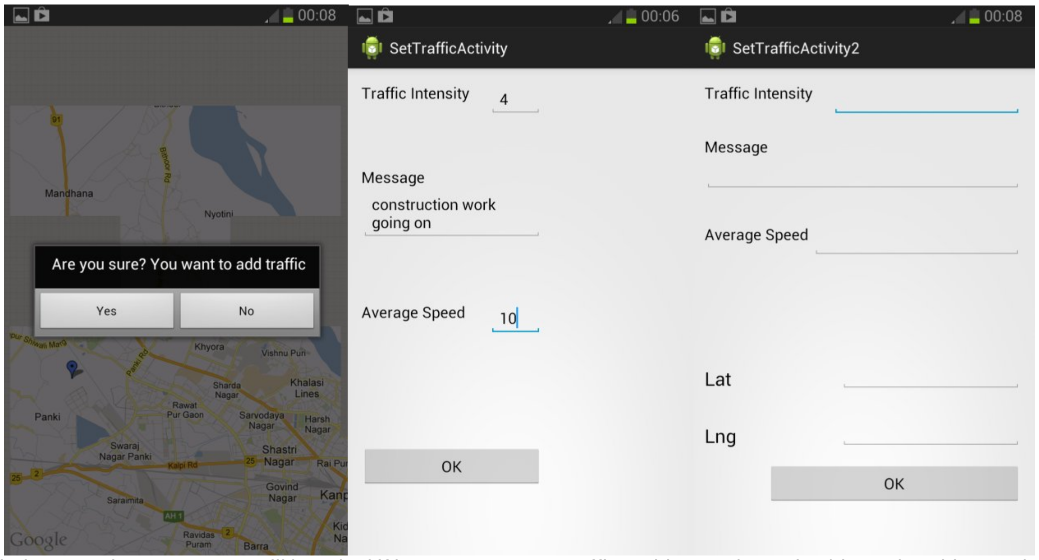
III. *Central Server:*

A MySQL server is created to store critical information regarding unforeseen circumstances. Each entry consists of 5 fields :- *<Latitude, Longitude, Message, Intensity, Average Speed* >.
Latitude & Longitude specify the problem location. *Message* describes the problem in textual form. *Average Speed& Intensity* are used in re-evaluating travel time estimates.

Now for each route between the source and destination, additional time will be added if a problem location lies on that route. We assume that a problem at some point will affect the entire leg.

Additional time = *Intensity* * leg_size / *Average Speed*

This additional time is added to previously calculated time of travel for the leg.
We also provided the option to add entry to the server.



Are you sure? You want to add traffic

Yes | No

SetTrafficActivity

Traffic Intensity 4

Message construction work going on

Average Speed 10

OK

SetTrafficActivity2

Traffic Intensity

Message

Average Speed

Lat

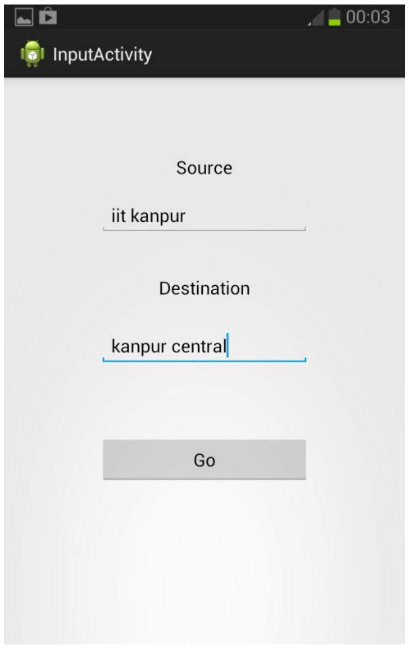
Lng

OK

Window1: On long press user will be asked if he wants to enter traffic problem at that point (shown by a blue marker)
Window2: Enter the relevant values
Window3: Alternate option to add entries by latitude and longitude of the point

IV. *Incremental Routing:*

The user is asked to provide the source and destination. Routes are obtained and the time of travel for each of them is calculated. The one that takes least time is shown in BLUE and the rest in RED. The GPS location of the user is continually taken. Once the user completes his first leg in the journey, the entire process is repeated and best route from there is shown. So, if some problem pops up during the travel, the re-routing takes place at the end of the current leg and the problem is averted. A message is displayed at the problem location describing the situation.



InputActivity

Source

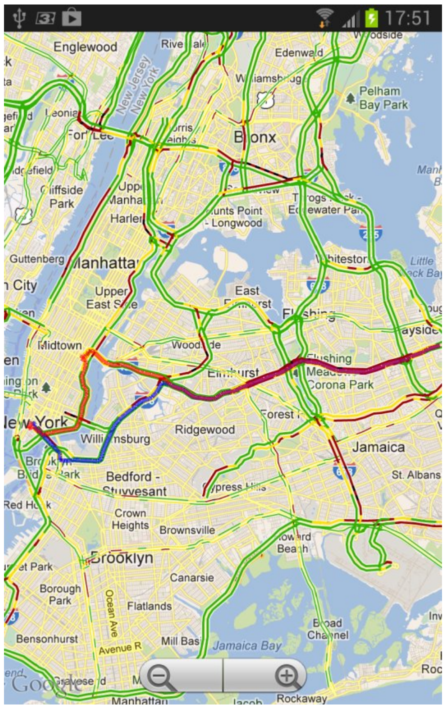
iit kanpur

Destination

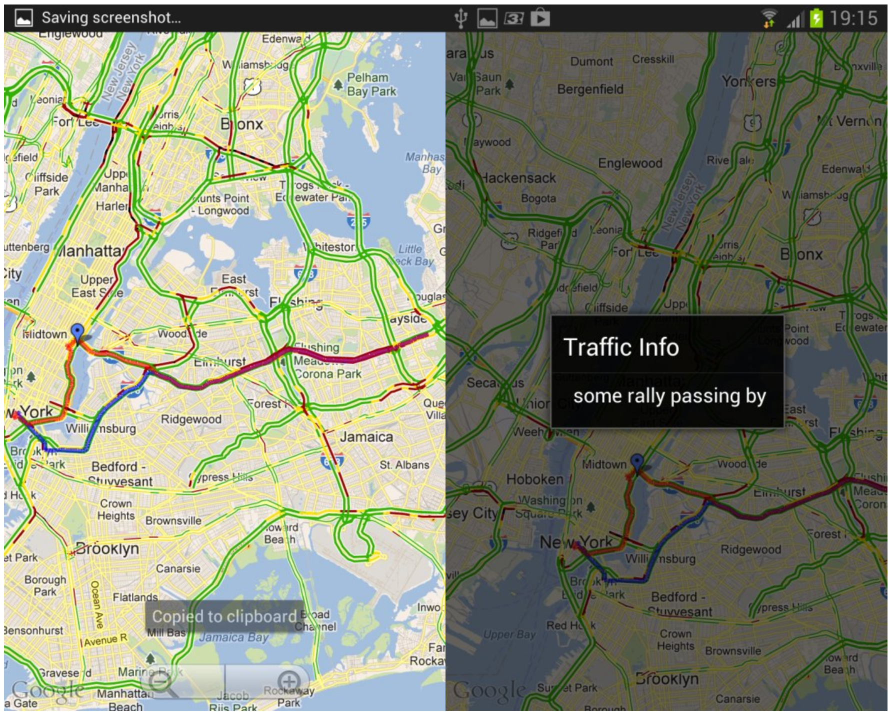
kanpur central

Go

User has to enter his source and destination from option menu



Two possible paths from Source to Destination. BLUE takes less time and RED takes longer



Traffic Info

some rally passing by

Marker indicating a problematic situation at that location. On clicking, you can read the message.

TECHNICAL DESCRIPTION

Tested on Platform: Android v4.2
Tested on Device: Galaxy One Note
Map Data: Google Maps API v1.1
External Libraries used:
JDBC connector - to connect to database server for fetching and Updating traffic data on our servers.
Java json library – for parsing json file.

Database used – MySQL. Version 5.6

We are using MapActivity to display google map and many other activities to take input from user. Using ItemizedOverlay we displayed traffic markers on map.
Using overlay class and

DIFFICULTIES FACED

The main difficulties we faced were parsing json file in which we had google map/route data and extracting traffic data from map.

Parsing of json file was not much difficult as we used json library to parse it.

As google does not provide traffic data we had to manually extract it from the traffic map displayed on screen. For each geo-coordinate which we received in json file we fetched its corresponding pixel location of android device using google map reflection api.

Then took screenshot of android device and fetched the colour of each pixel location. Each colour value is related to traffic condition at that point as mentioned above. Using pixel colour value we noted traffic value at that point.

ACKNOLEDGEMENTS

We would like to thank Dr. R.K Ghosh for his valuable suggestions in framing the project. We are thankful to CSE department for providing us with a device to test our project.