HiReach
innovative mobility solutions to cope with transport poverty

Documentation

Congestion API

AUTHOR

CODRUT ALEXANDRU PANEA

SEPTEMBER 2020

www.hireach-project.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 769819.

TABLE OF CONTENTS

1.1 Introduction **1**
1.2 Congestion API – How it works **1**
1.3 Machine Learning Prediction API Module **3**
1.3.1 Challenges when building the ML module *3*
1.3.2 Solution 1: Predict the whole system state *3*
1.3.3 Solution 2: Predict individually the state of small portions of the system *3*
1.4 Congestion API Gateway – How it works **6**

LIST OF FIGURES

Figure 1. High Level Architecture Diagram of the Congestion API 1
Figure 2. Visual representation of how the API is calculating the congestion in a bus stop 2
Figure 3. Visual representation of the history of location data in the bus stop Piata Titan 4
Figure 4. Location data history from the bus stop Piata Titan represented as time series data 4
Figure 5. Generated time series data 5
Figure 6. Predicted values using Prophet 6
Figure 7. High-level architecture diagram of the API Gateway 7

1.1 Introduction

This is a REST API built to track congestion spots and crowded areas using real-time location data from mobile devices.

Use case: A mobility mobile app is available, with a large user base. The mobile app, installed on a user device, will send location data (latitude, longitude) to this API. Using this data, the API calculates live congestion in different requested spots. The API can also make future predictions based on history data by using a Machine Learning algorithm.

By using this API, you can:

- show crowded areas on a map
- plan a route avoiding crowded bus stations
- notify your users when approaching crowded areas
- predict future congestion to enhance route planning

The Congestion API can be used with the following two APIs to extend its feature set:

- **Congestion API Gateway** that is a REST API gateway built to aggregate data from multiple Congestion APIs. The congestion API is more accurate when the user base is larger. This API Gateway is built to make it possible for multiple companies to share their congestion data while keeping it in their own private database. The more companies contribute with location data, the more accurate the congestion data will be.
- **Prediction API** that is a REST API built on top of **Facebook Prophet** [1] for forecasting time-series data. It is used by the Congestion API to predict future congestion.

1.2 Congestion API – How it works

The high-level architecture of this API is presented in Figure 1.

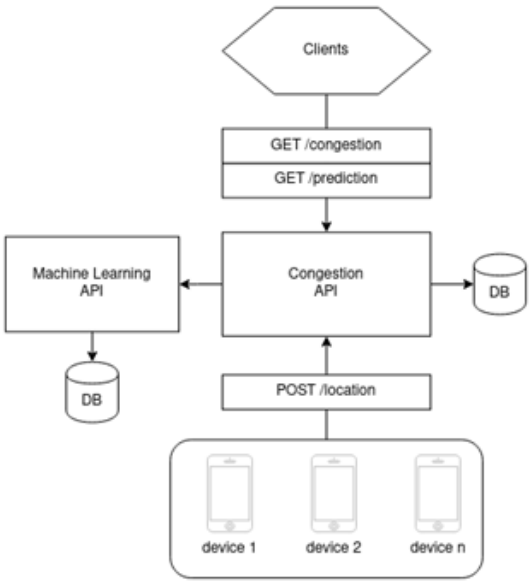

Figure 1. High Level Architecture Diagram of the Congestion API

The Congestion API is storing location data (latitude, longitude), received from mobile devices, in a PostgreSQL database having the PostGIS extension.

PostGIS is a spatial database extender for a PostgreSQL object-relational database. It adds support for geographic objects allowing location queries to be run in SQL [2].

Using the PostGIS extension built-in functions we can calculate distances between points to find out how many devices are in a certain location.

In the image below we have a visual representation of how the API is calculating the congestion. The red dots represent the coordinates (latitude, longitude) of the devices. The black circles represent the perimeter to be considered when calculating congestion.

If we want to know the congestion at the bus stop Piata Titan, which is the centre of the two black circles, we need to provide the API the polar coordinates of the bus stop and a value for the radius of the circle. The small circle has a radius of 10 meters and the big circle 20 meters. Using the API, we will find out that there are two devices when providing a radius of 10 meters and 3 devices when providing a radius of 20 meters.

Behind the scenes the API is calculating the distances from the centre of the circle (the bus stop) to every other device on the map. If the distance is lower than the radius then the device is counted.

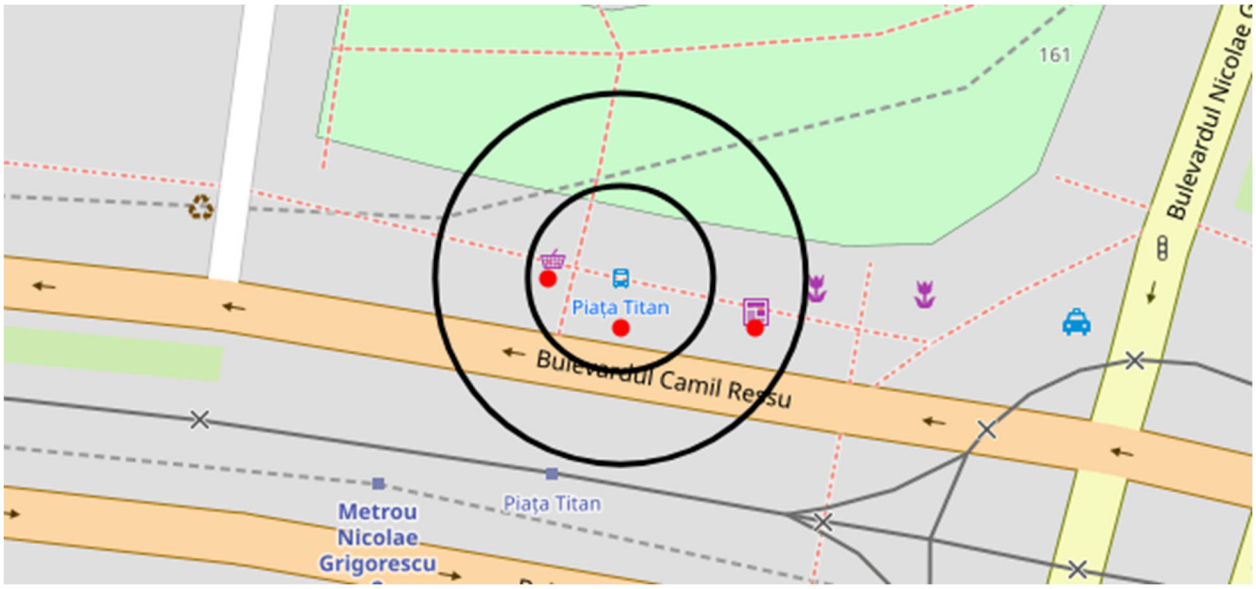

Figure 2. Visual representation of how the API is calculating the congestion in a bus stop

The distance is calculated using the haversine formula which determines the great-circle distance between two points on a sphere given their longitudes and latitudes [3].

We first tried to calculate the distance using Vicenty's formula [4] which considers the Earth to be a spheroid (a mathematical approximation of the Earth's surface taking the flattening at the poles into consideration). This method is the most accurate method of calculating distances between polar coordinates, but it is also the slowest. For coordinates that are close together, or close to the Equator, the loss in accuracy when considering a sphere over

a spheroid would be negligible. So, we decided to consider the Earth to be a sphere because it is mathematically much simpler than the spheroid and therefore much faster to calculate, making the overall speed of the API greater.

1.3 Machine Learning Prediction API Module

The Congestion API alone can only tell the congestion grade happening now or in the past. Making future predictions of the congestion allows to plan a route and exclude congested stops from it. The Machine Learning Prediction Module adds this feature to the Congestion API.

1.3.1 Challenges when building the ML module

When querying for congestion we can set a variable value for the radius of the circle that forms the perimeter of the location we want to get the congestion grade from. The location data coming from the devices is saved into the database with a timestamp, but it is also spread across the surface of the Earth. Having time series data and the ability to set a variable value for the radius of the circle with the centre in any point on the Earth's map makes the problem challenging.

1.3.2 Solution 1: Predict the whole system state

Every device that sends its location data to the API is a mobile device used by a real person. One potential solution is to predict the next move of every person on the map, up to the point of the wanted future prediction date, using the history of past location data. Doing this we have a future state of the whole system and we can apply the same query as the one getting the current congestion without any modifications.

Predicting the next move of a person would require a long history of past locations at regular time intervals. For the prediction to be accurate we would need to train the next movements of every person individually because every person has different patterns.

We gave up on this idea because it would require lots of computational resources and would be hard to build an accurate model.

1.3.3 Solution 2: Predict individually the state of small portions of the system

Instead of predicting the state of the whole system we can predict the state of a small area defined by the location point and the radius requested by the user of the API.

This approach is totally different from the first one because we do not predict the next move of individuals on a map. Instead we transform our data into pure time series data and use a time series forecasting model. It is a faster approach and a lot less computationally intensive.

How it works

Let us consider that a user made a request to predict the congestion in the bus stop Piata Titan (44.4133671, 26.1630280) with a radius of 10 meters at a certain future date. The next steps will follow:

1. *Transform location data into time series data*

First step is to get the history of devices in that location and transform it into time series data. From Figure 3 we can see that counting the devices (red dots) inside the perimeter of the black circle at regular time intervals gives us time series data.

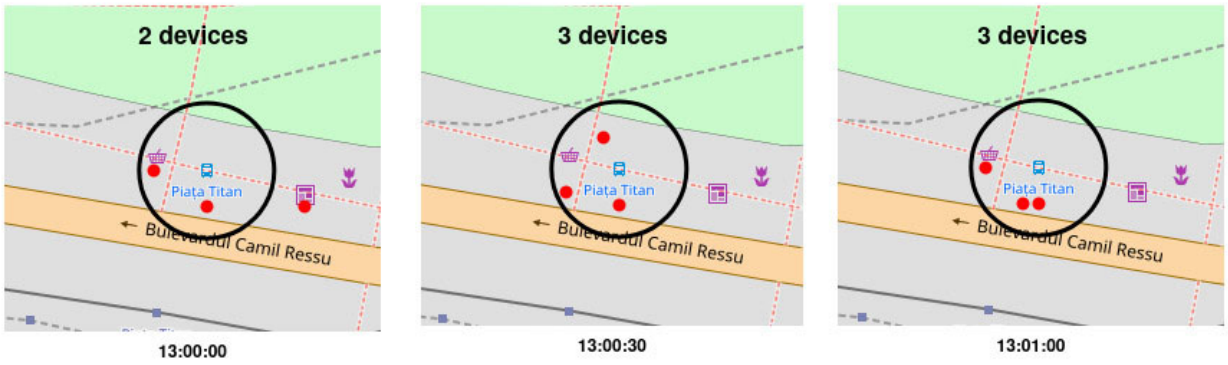

Figure 3. Visual representation of the history of location data in the bus stop Piata Titan

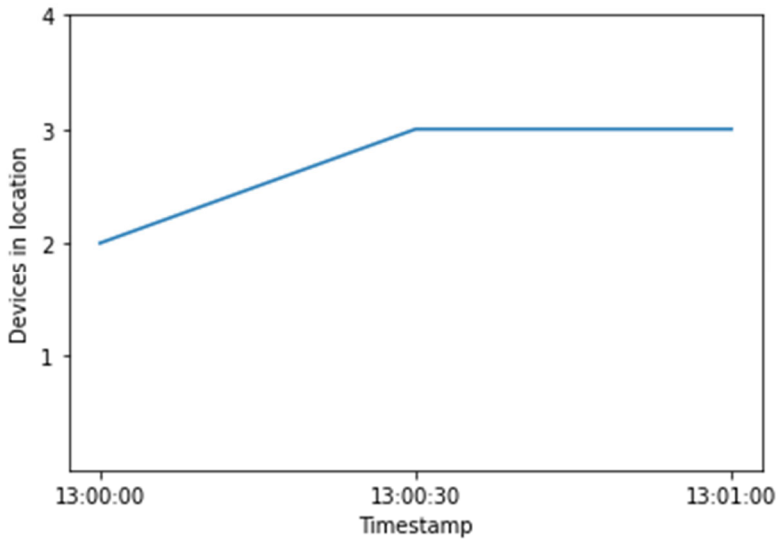

Figure 4. Location data history from the bus stop Piata Titan represented as time series data

Because the interval at which a certain device sends its location data cannot be perfectly regular (one possible cause can be network latency), we ceil/floor the timestamp. E.g. *13:00:14* becomes *13:00:00*.

2. *Fit the data using a machine learning model*

Next step is to fit the data using a Machine Learning model for forecasting time series and use this model to predict the congestion for the requested future date.

Because we do not have real data, we need to simulate it so we can test our model. We generated data trying to simulate the typical daily congestion in a bus stop. The data has the following properties:

- Between 7:00-9:00 or 16:00-20:00 generate random values from 16 to 20 devices.
- Between 9:00-16:00 generate random values from 5 to 8 devices.
- Between 20:00-23:00 generate random values from 0 to 4 devices.
- Between 23:00-7:00 generate random values from 0 to 2 devices.

Following the rules above a value for the number of devices is generated every 30 seconds over a period of two weeks.

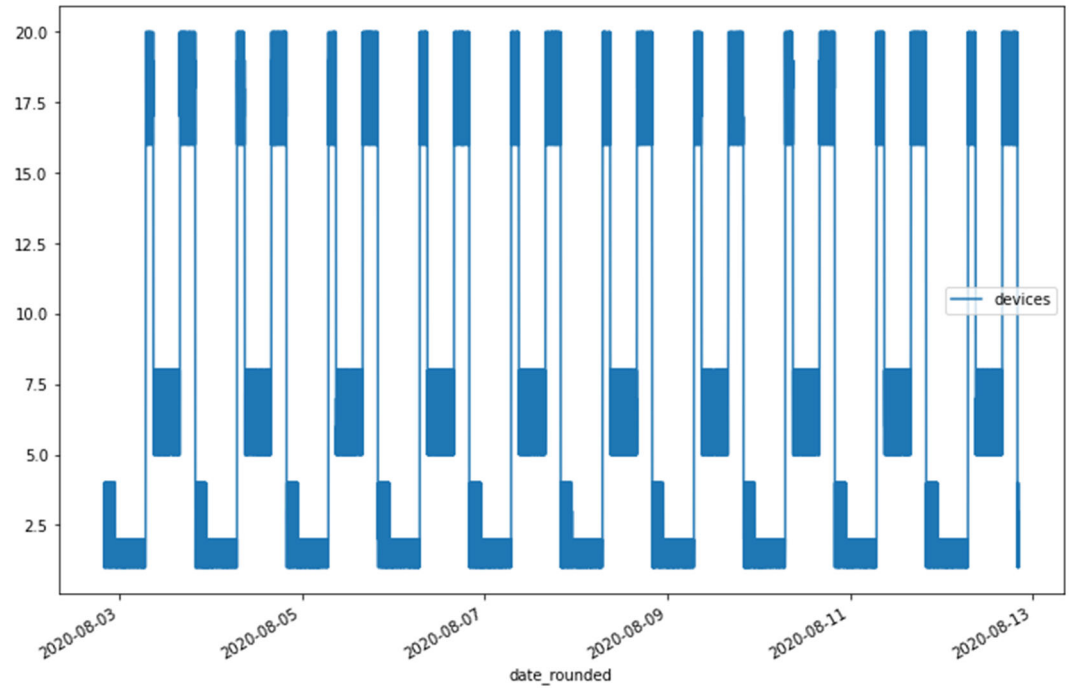

Figure 5. Generated time series data

When choosing the right model for forecasting we want to choose one that is fast and accurate.

We first tested the Autoregressive Integrated Moving Average (ARIMA) model [5]. Because we have daily seasonality for our data, the ARIMA model needs to do a high number of iterations over the data. For this reason, we could not successfully fit the model on a machine with 24 GB of RAM. Because this model requires lots of resources, we started to look for other, less demanding, models.

Second model we tested is called Prophet. Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data [1].

Fitting and predicting values for our generated data using Prophet takes roughly three seconds which is fast. We can see the results in Figure 6.

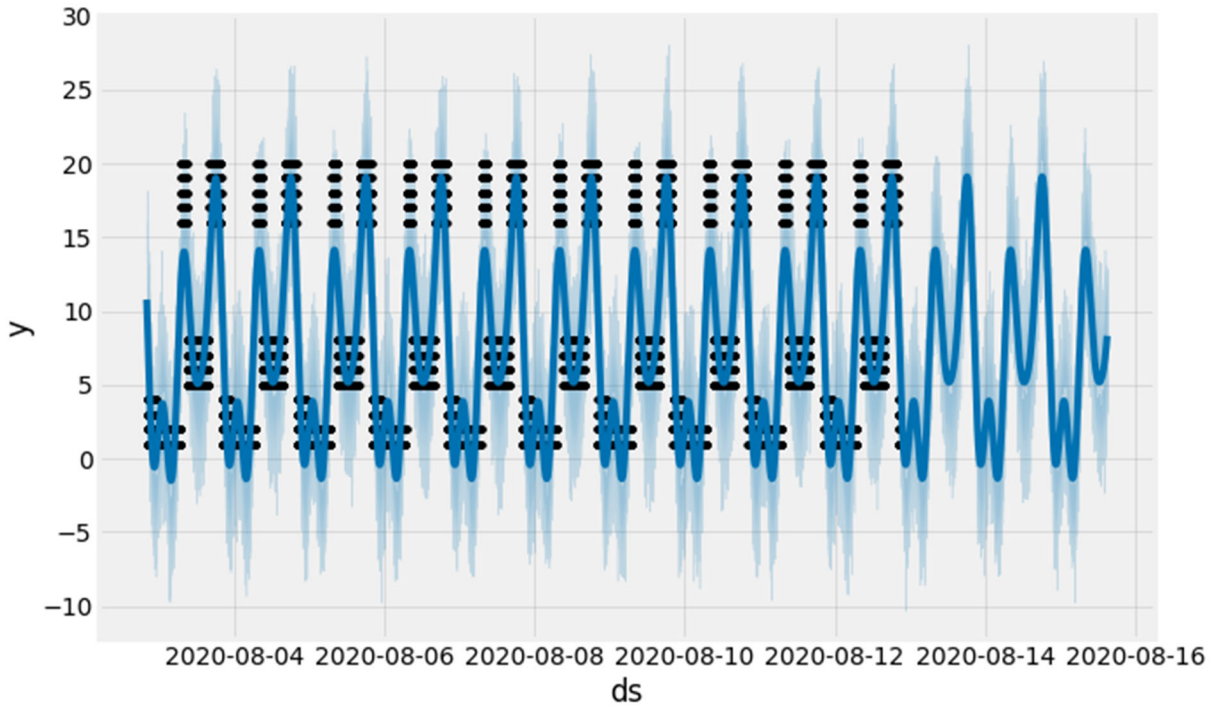

Figure 6. Predicted values using Prophet

The black dots represent the generated values and the blue line represent the predicted values by our model. This is the result using the default settings for the model without any tuning. The results are not perfect and can be improved by fine tuning the settings. We decided to not do any tuning of the settings and use the defaults because our data is not real. If we tune the model for this data, it might not work when used with real data.

We decided to stick with this model because it is fast, easy to tune and gives good results even without any tuning.

3. *Cache the fitted model*

After the prediction has been made, we cache the fitted model for the requested area to use it for subsequent requests. By doing this the following prediction requests will receive a response almost instantly (roughly 100 milliseconds) even when changing the prediction date.

1.4 Congestion API Gateway – How it works

Multiple companies are using the Congestion API, but they want to keep their user's location data in their own private database and only share congestion data. Users who have access to the API Gateway can request data about congestion in a certain point defined by latitude and longitude. The gateway then calls every instance of the Congestion API and aggregates the data from all companies thus returning a more accurate response. The Gateway also provides the option to get all instances URLs and call them individually. You

can store the Congestion API instances URLs and call them directly in case the API Gateway is down.

The high-level architecture of this API is presented in Figure 7.

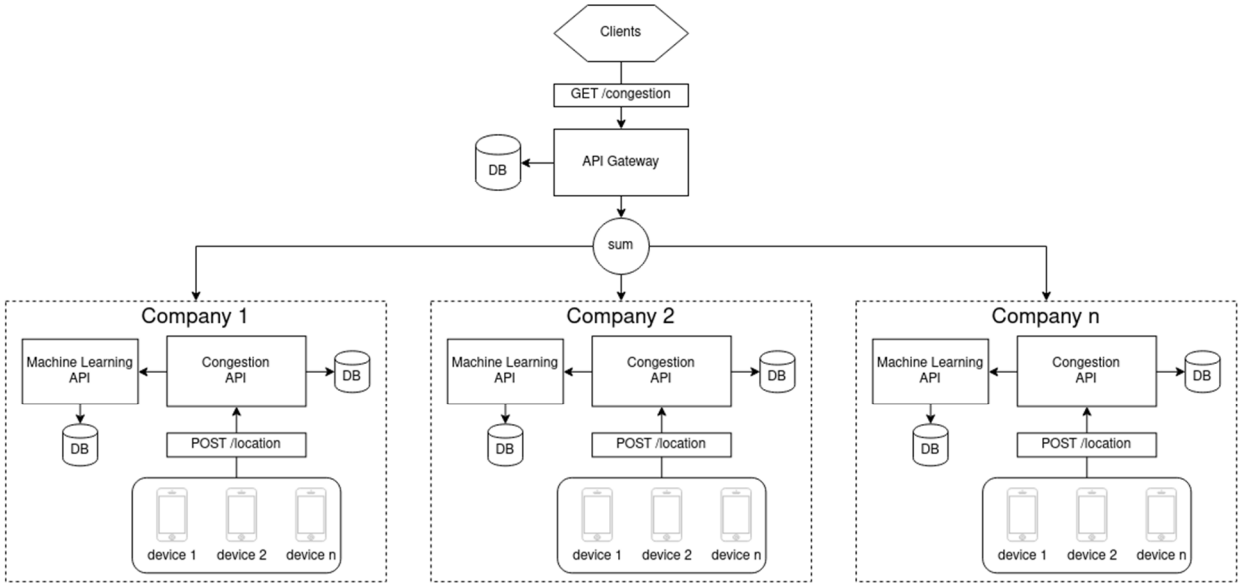

Figure 7. High-level architecture diagram of the API Gateway

[1] Prophet: https://facebook.github.io/prophet/

[2] PostGIS: https://postgis.net/

[3] Haversine formula: https://rosettacode.org/wiki/Haversine_formula

[4] Vincenty's formula: https://www.movable-type.co.uk/scripts/latlong-vincenty.html

[5] ARIMA: https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average