MTA Data Stream and Predictive Models

Rigers Qeraj, Himani Arora, Mohneesh Patel, James Guevara

Abstract— **This project proposes a real-time application to provide New York MTA data to customers in a more informative manner. We do so by using MTA streaming updates in conjunction with real-time weather and historical turnstile data to give well-rounded information on fastest routes available, destination ETA. In addition we also provide a metric to inform riders on crowd conditions in their trip. Our application is built on top of GTFS real-time format[1]. We used turnstile data for the past 2 years to employ Random Forest Regression to predict station entries. Our best model gives 0.90 R-squared on the validation set whereas the average R-squared achieved is generally 0.74.**

I. INTRODUCTION

The New York City Subway is one of the largest rapid transit systems in the world. It has the most number of stations with over 400 in operation. It is also the seventh busiest rapid transit rail system delivering over 1.76 billion annual rides in 2015 with increasing number of riders per year[2]. 6,106,694 customers rode the subway in a single day alone on September 23, 2015 making it the highest ridership recorded since 1985.[3]

A large number of these riders use the NYC Subway to commute to work or school and it is crucial for them to use a route that is efficient and maximizes punctuality. To help riders navigate the system, the MTA provides official mobile applications that give real-time train updates and service alerts with the ability to plan routes[4]. Apart from this, there are many third-party applications that have additional features. Roadify gives crowd-sourced updates on travel conditions by other users.[5] Citymapper gives local weather conditions and also tells the best spot to hop on and off the train for the nearest exit.[6] All these applications use mainly real-time data to give information to users. One disadvantage of such an approach is that users can only know of an event once it has already happened. For example, according to historical trends, Times Square - 42^{nd} St station is very crowded on Mondays at 8AM, chances of a rider catching the first train is low. The same is true for a particular train on a route that is more affected by rain and can lead to delays. Instead of waiting for other users to post about crowded stations or waiting for status updates from the MTA regarding this delay, it is helpful if such events can be predicted in advance so that riders can be warned of this delay beforehand and include it in their travel plans.

We overcome this limitation by harnessing the vast amount of historical data that MTA provides. We use this data to model how transit times vary utilizing a number of factors. In this project we focus on finding the effect time of day and weather has on ridership and route-specific delays, both of which can lead to longer than anticipated transit times. To calculate ridership, we used turnstile entry data of the past 2 years, supplied by the MTA, to build a predictive model that tells how crowded a station is on a scale from 1 to 6; 1 being least crowded. For route specific delays we used historical delay data along the 1 train line and again correlated it with weather and time. Combining this with actual trip updates enables us to give the fastest route, predict delays and warn riders about station crowdedness before they start their journey.

For the scope of our project we selected a subset of stations that formed our inventory and restricted our analysis to these stations and lines.

II. DATA

A. MTA Datamine Stream

New York City Transit (NYCT) system provides real time data feed that provide information about the transit system status[7]. The information is updated every 30 seconds and requires users to pull information from the MTA system. The data for the entire subway system is split into different feeds. In our application we used the feed that provides information for the 1,2,3,4,5,6 and S (shuttle) lines. Every 30 seconds the feed provides information for all trips in progress as well potential future trips. According to the MTA trips are schedules up to 30 minutes in advance but are not assigned to trains until minutes or sometimes seconds before departure[8]. The feed is composed of a single header(timestamp) and multiple entities. There are 3 types of entities: *trip_update, vehicle,* and *alert*. A sample of the feed is included in the Appendix.

1) trip_update: entities that provide information pertaining to a single trip. It includes the arrival and departure times for the next stop as well as subsequent stops along the trip. Once a train has left a stop_station the stop_station is removed and is no longer present in the feed. Each *trip_update* field is composed of *stop_time_updates*. *stop_time_updates* is where the trip schedules are found. The entity also has a *trip_id* field to identify individual trips. This is analogous to identifying an individual train.

2) vehicle: entities that provide the current location of the *trip_id* i.e. location of each train. The location is expressed relative to the next stop in the sequence: IN_TRANSIT_TO, COMMING_TO, STOPPED_AT

3) alert: entities provide information that affects trips. MTA currently uses only one alert type: Train Delayed[8]. *alert* entities aren't always present and are included only when there is a delay in the system. *alerts* are associated with a *trip_id* and route. This is to signify which specific trips and routes are affected.

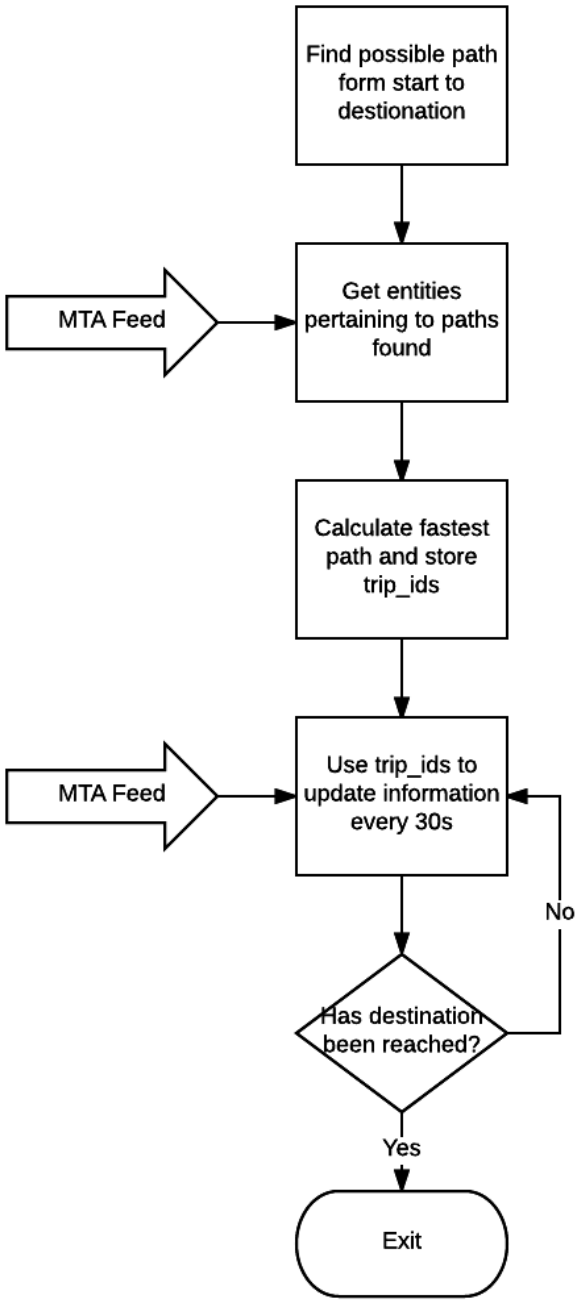

Fig. 1. Flowchart of MTA stream process

B. MTA Datamine Historical

MTA historical data is provided in a similar fashion as the real-time feed, however it is updated every day at midnight to include the previous aggregated daily results.

C. MTA Ridership Data

The MTA releases a weekly log of cumulative entries and exits for every turnstile in every station for 4 hour intervals [6]. We used data from January 2015 to April 2017 to build our model. Each row has a unique identifier for every station as well as every turnstile. Additionally, it has information about the subway lines that operate at each station, date and time when the data was recorded, and the cumulative number of entries and exits at each turnstile

D. Weather Data

Weather Underground provides historical weather data at hourly intervals. This data includes temperature, humidity, precipitation, and rain/snow data. For the ridership model, we used data from January 2015 to April 2017, using temperature, rain, and snow features. For the delay model, we used data from November 2014 to October 2014 (only using rain and snow features).

III. MTA Stream Analysis

A. Accessing the Stream

Accessing the MTA data stream is relatively simple. It is only required to submit a GET request to the MTA feed url[1] and it responds with data encoded in the GTFS format. In order to access the information present in the feed it is required to use the 'Protocol Buffers' a method of serializing data developed by Google[9]. Google provides bindings for accessing the data in a few different languages. For the purpose of our application we have chosen to use Python. Our goal was to create a streaming application using SPL and Python. However, we were unsuccessful in this goal and a description of challenges and limitations is provided in the in the Limitations subsection.

B. Processing the Stream

When a user selects the start and destination stations of their trip, the station names are mapped to *stop_ids* that are used by MTA to denote individual stations (which are then used in the gtfs-feed). The application then finds possible paths from the start to the destination. The application then uses the start, destination, and possible transfer *trip_ids* to extract only the relevant entities from the feed. Our application decides which trips to take by comparing the arrival and destination times of the individual paths possible. It selects the path with fastest arrival time and then continuously tracks the related trips (multiple trips if a transfer is required) to provide up to date information. The updated information includes updated arrival times and any potential delays that may have occurred.

IV. Predicting Subway Crowdedness

A. Data Cleaning

Since the MTA turnstile data only had cumulative entries we first converted these to the absolute values for the 4-hour period in which the data was recorded. We assumed the ridership to be uniform for this interval and thus estimated the hourly entries by equally dividing the entries for the 4-hour interval. Once we had the hourly entries for each turnstile, we then grouped the rows by each station and then aggregated the turnstile hourly entries that correspond to

[1] http://datamine.mta.info/mta_esi.php?key=API_KEY&feed_id=1

the same date and hour of day. This resulted in a dataset that has a single row that gives the value of the number of people who entered that station at each hour of each day from January 2015 to April 2017. We then joined this with New York hourly weather data with indicators for rain, snow and temperature.

B. Feature Extraction

To find discriminative features, we compared how entries varied across the various features. Our findings suggested that entries were mostly dependent on the hour of the day and day of the week; following the general trend of high crowd during rush hours and much smaller values during weekends. However, the rush hour data differed from station to station; we believe the difference is due to locations near commercial or residential areas.

We also analyzed ridership with weather, however, the results were not that obvious. Rain did not seem to affect ridership significantly and although heavy snow clearly resulted in a drop in the entries. The number of days it snowed is very small in comparison to the total number of days and therefore it cannot be expected to be modeled very well. We also saw the variation in ridership with weather, and found that extremely low temperatures definitely reduced crowdedness, however, on an average day, the ridership was not affected much by the temperature.

Based on this, our final feature vector consisted of day of the week (Monday, Tuesday etc.), hour of the day, month, temperature in Fahrenheit and indicators for rain (0 or 1) and snow (0 or 1). Since day, hour and month are categorical variables they were encoded as one-hot vectors.

C. Model Selection

In the previous section we noticed that the pattern of entries was different for different stations. This motivated us to build a separate model for each station in our inventory. Moreover, since the actual number of entries may not be of much significance to the rider, we scaled the entries to lie between the range of 1-6 to make it more interpretable, For each station, we divided its data into training (80%) and validation (20%) sets. Since the goal was to predict the number of entries which is a real-valued output, we modeled it as a regression task. We tested a number of models such as Least Squares, SVM, Decision Trees and Random Forests. The best model was achieved by Regression Random Forests where the number of indicators was a hyper-parameter for which the model achieved best validation results.

V. Predicting Route-Specific Delays

We built a model to predict route-specific delays using several features: hour, day of the week, a snow indicator, and a rain indicator. We obtained data from November 2014 to October 2015 in the MTA historical dataset, which provides trip updates (including delays) every 30 seconds. The historical data is saved in 5 minute increments; however, we only obtained data at 30 minute increments (due to the fact that it would take too long to get all the data). We

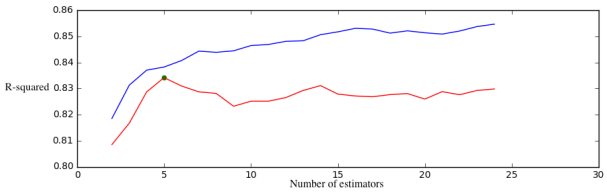

Fig. 2. Variation of train R-squared (blue) and test R-squared (red) with the number of random forest estimators for 42^{nd} St station. We can see that beyond a certain point, increasing the number of estimators decreases the test R-squared while the train R-squared steadily rises. The best value for number of estimators (green) is selected as the one that gives the maximum test R-squared.

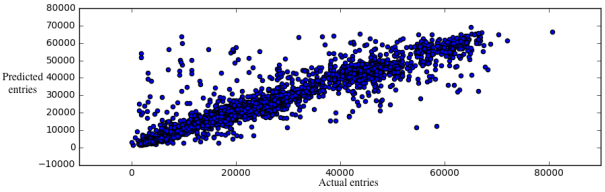

Fig. 3. Predicted entries versus the actual entries for the best model (corresponding to 0.84 R-squared)for 34^{th} Street Penn Station.

obtained weather data in the same time range to correlate the weather and delay. In total, we obtained 8684 data samples (each of which includes the features noted above).

We made a couple important assumptions when building the model. First of all, we assumed that each route operates in a consistent manner (i.e. route 1 operates the same way across Mondays at the same hour). We also assume that a delay applies to an entire route (not just a single train in a route).

For training, we used 3 classifiers: AdaBoost (with decision trees), Logistic regression, and a decision tree classifier. They all obtained similar accuracy on route 1 data, which is shown on Table 2. However, there are several important caveats. First of all, out of the 8684 data samples, only 1084 are delays. Secondly, despite the high accuracy, this is mostly due the fact that the classifiers almost always predict no delay. Third, the features are discrete and the combination of them only totals 672 (the product of 24 hours, 7 days in a week, rain/no rain, and snow/no snow). We suspect that this fact, along with the fact that we only predict 2 responses (delay or no delay), means that this small set of features don't allow for a robust enough model for predicting delays (and to the extent that they do, the models we built are probably equivalent to the empirical probabilities of there being a delay for the corresponding set of feature values). Also, there is virtually no snow in the weather data, so that has no real effect, while Figure 4 suggests that no significant correlation exists between rain and delays.

Figures 5 and 6 show delays per hour and delays per day of the week (for route 1), respectively (we summed over all the delays for each hour and each day for the respective plots). From Figure 5, we can see that is moderate variability in delay with respect to hour, with a significant dip in delays

TABLE I

R-SQUARED VALUES

Station	Training R-Squared	Testing R-Squared	Number of Estir
14 Street	0.71	0.68	5
34 Street	0.85	0.84	18
42 Street	0.69	0.69	14
59 Street	0.92	0.90	13
72 Street	0.71	0.69	18
116 Street	0.67	0.64	8

in the hours between 8am and 12pm. From Figure 6, we can see that there is a significant dip in delay on Sunday, and minor jump in delay on Friday.

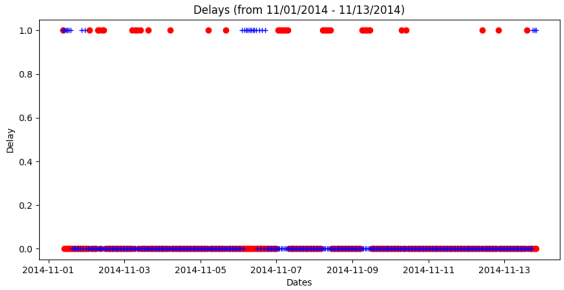

Fig. 4. Actual delay at each day (in red), ranging from 11/01/2014 to 11/13/2014. Also shows whether or not it is raining at each hour in each day (in blue).

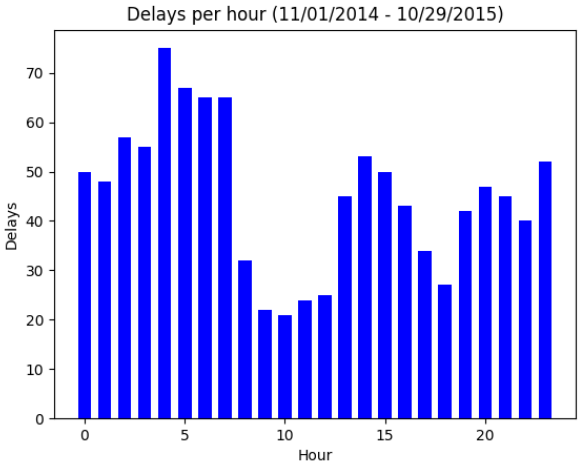

Fig. 5. The total number of delays at each hour, over the time period of 11/01/2014 to 10/29/2015

TABLE II

CLASSIFIER ACCURACIES FOR ROUTE 1

Classifier	Training Error	Testing Error
AdaBoost	0.880	0.866
Decision Tree	0.877	0.871
Logistic Regression	0.877	0.871

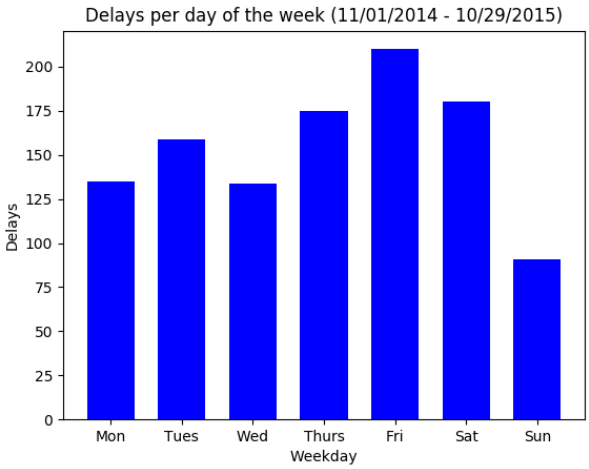

Fig. 6. The total number of delays on each day of the week, over the time period of 11/01/2014 to 10/29/2015

VI. IMPLEMENTATION

The back-end of our application is separated in 2 parts:

1) MTA Stream: we have implemented the analysis of the MTA stream with an AWS Lambda function and connected it with an API endpoint. This endpoint is live and responds in real. Link and instructions are included in the Appendix.

2) Web Application: New York Ditie! is implemented in Flask, it is a micro web framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine. The front-end is in Javascript, HTML, and CSS for visual appeal. In the background, we run the Regression Random Forest Machine Learning algorithm which takes into account the weather data, MTA streaming data, and MTA historical turnstile data to calculate crowdedness of subway stations.

VII. CHALLENGES

A. Path Finding

Due to the large number of stations and routes the possible paths one can take is tremendous. Therefore finding all possible paths requires it to be an automated process. The scope of our project was to experience and create a real-time application. For this reason we have only selected a few stations on the 1,2,and 3 lines to run our project and provide real time updates only on this specific stations.

B. Turnstile Data Cleaning

The MTA turnstile data had many inconsistencies which posed as a challenge during the data cleaning phase. Sometimes different naming conventions have been used to refer to the same station e.g. 42^{nd} Street station has been referred to by 'TIMES SQ-42 ST 1237ACENQRSW', 'TIMES SQ-42 ST 1237ACENQRS', '42 ST-TIMES SQ 1237ACENQRS'. Moreover, since the MTA gives the cumulative entries it was difficult to find when the turnstile counter was reset as there was no definite was of knowing, except when the actual entries started to become negative. Additionally, entries were

not updated at regular 4 hour intervals which made it difficult to calculate hourly entries.

C. MTA Delay Classification

Regarding the delay model, the main challenge was how to actually build a model. The MTA streams data every 30 seconds, which gives trip updates for each train/route, and which trains/routes experience delays. Thus, first of all, since data is obtained every 30 seconds, multiple delay messages can refer to only one actual delay (for a single train/route). Due to this ambiguity, we simply chose to output 1 if there was at least 1 delay within an interval of 1 hour, and 0 otherwise. Secondly, we chose a one hour interval due to the fact that we obtain weather data at hourly intervals. Thirdly, there was very little data where there was rain or snow, so the rain and snow features have virtually no effect on predicting delay. Finally, since the combination of features is only 24*7*2*2 (24 hours, 7 weekdays, and rain and snow indicators), and the output is only 1 or 0 (for delay or no delay), the classification model most likely just matches the empirical probabilities of there being a delay or no delay with respect to those features. Another minor challenge was due to the fact that there was a limit to how many times we can call the weather underground API. Also, initially we were going to build a model using only less than 2 weeks of data, but we later found more historical data (though no data is available for 2016 or 2017, so it may be somewhat outdated).

D. SPL and Python

To create a real streaming application we decided to take advantage of the IBM SPL streaming language and the Python bindings available to access the GTFS MTA feed. IBM provides an SPL Python API to create streaming applications using Python. The caveat was that Python SPL works with Python3 while the Google bindings for GTFS are available for Python2.7. Compiling our own .proto file bindings for use in Python3 proved unsuccessful.

VIII. Conclusion

In this project we designed a real-time application that provides fastest route, destination ETA, and regular status updates of trains from the New York MTA data. We supplemented this application with our predictive models of subway crowdedness which we implemented in Flask. We also modeled the delays of trains on line 1 to gain further insight. We showed how these parameters vary with time; and studied their correlation with weather. Our project, thus used both real-time streaming data as well as large amounts of historical data allowing us to learn the fundamentals of data processing.

IX. Future Work

A. Automatic route

A natural expansion of this project would be for the application to auto-generate the paths necessary from the start to any destination station. It would generate a path regardless of the number of transfers. Once the path is generated the *stop_ids* can be entered into our application to get the fastest path and subsequently update the information pertaining to the chosen path. Additionally the application would take into consideration weekend and holiday schedules.

B. Event scheduling

During certain events, e.g. sports events, concerts, etc. being able to predict train crowdedness can offer a great advantage to riders. Matching event schedules and finding which routes are affected can help riders prepare better or take different routes. Without prior knowledge riders can easily miss trains due to large number of customers.

APPENDIX

Please find supplementary results below for deeper understanding of our work.

AWS Implementation: We have a live implementation of the MTA Stream process running on AWS Lambda. A sample can be seen by using the following URL: `https://ed35eyqphd.execute-api.us-east-1.amazonaws.com/prod/getTrip?start=14St&end=116St-ColumbiaUniversity` "14 St" and "116 St - Columbia University" are query parameters that tell the API which stations to use. The API returns a json document with *trip_ids* and other relevant information. To get trip updates simply append the query parameters `tripstart=TRIP_ID` and if first query returns a transfer, append `tripend=TRIP_ID`. A potential URL looks like this: `https://ed35eyqphd.execute-api.us-east-1.amazonaws.com/prod/getTripUpdate?start=14St&end=116St-ColumbiaUniversity&tripstart=130550_2..N01R&tripend=132800_1..N02R`

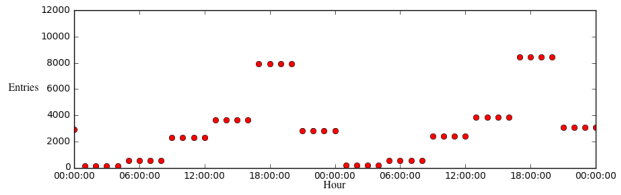

Fig. 7. Hourly entries for 116^{th} Columbia station for 2 days in March, 2017

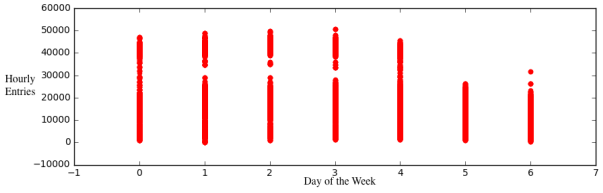

Fig. 8. Hourly entries for 116^{th} Columbia station as a function of day of the week where 0-4 is Monday to Friday and 5,6 is Saturday and Sunday respectively. It can be seen that the entries vary depending on what day of the week it is.

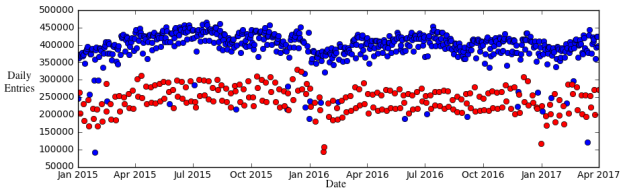

Fig. 9. Daily entries for 42nd Times Square station for 2 years. Here, blue points are the entries for weekdays and red represents entries on weekends. It is clear that the entries on weekends is lower than that on weekdays leading us to believe that Times Square is more crowded on weekdays.

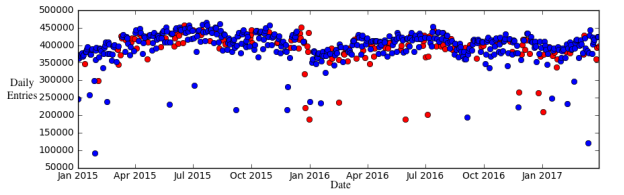

Fig. 10. Daily entries for 42nd Times Square station for 2 years for the days it rained (red) and did not rain (blue). No pattern can be inferred which shows that probably rain does not affect ridership as much.

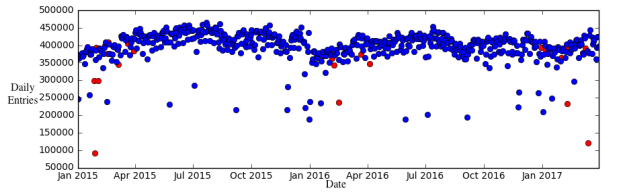

Fig. 11. Daily entries for 42nd Times Square station for 2 years for the days it snowed (red) and did not snow (blue). It is apparent that snow heavily reduces the ridership, however, the number of days it snowed is a small percentage of the total number of days.

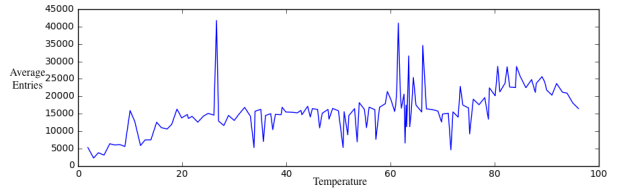

Fig. 12. Average hourly entries for 42nd Times Square station for 2 years as a function of hourly temperature. No distinct pattern can be found except that hourly entries are low for very low temperatures (¡20).

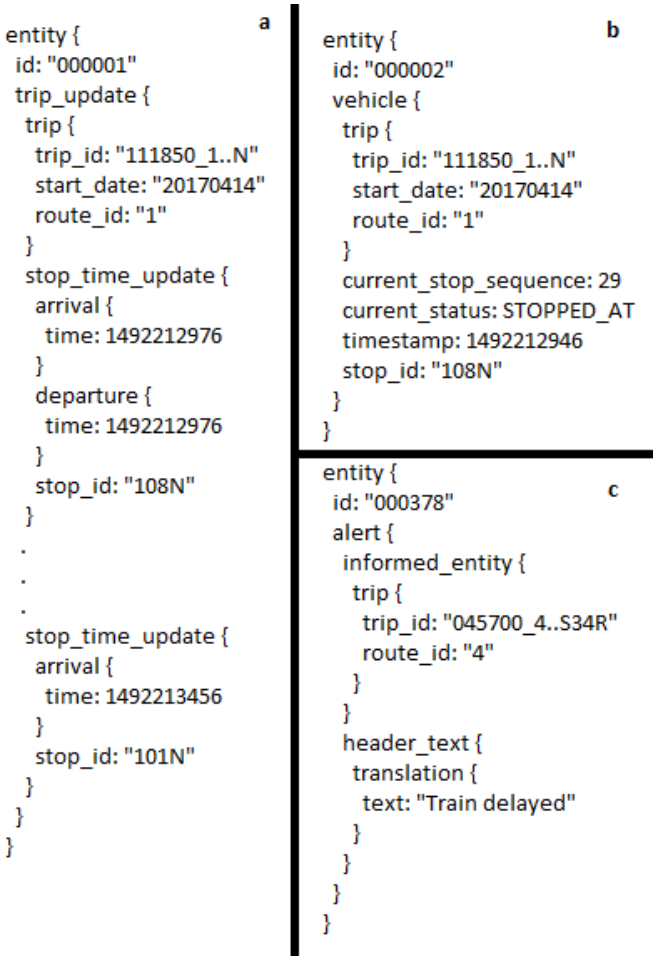

Fig. 13. Sample of entities present in the feed a) *trip_update* b) *vehicle* c) *alert*

References

[1] Transit — Google Developers, Google. [Online]. Available: https://developers.google.com/transit/. [Accessed: 04-May-2017]

[2] Subways Facts and Figures. [Online]. Available: http://web.mta.info/nyct/facts/ffsubway.htm.

[3] More Than 6 Million Customers Ride Subways on Five Separate Days in September, MTA — news. [Online]. Available: http://www.mta.info/news-subway-new-york-city-transit-ridership-record-breaking/2014/10/22/more-6-million-customers-ride.

[4] App Gallery, MTA — App Gallery. [Online]. Available: http://web.mta.info/apps/

[5] Roadify Transit, Roadify Transit. [Online]. Available: http://www.roadify.com/. [Accessed: 04-May-2017]

[6] Citymapper, Citymapper - The Ultimate Transport App. [Online]. Available: https://citymapper.com/.

[7] MTA Real-Time Data Feeds, MTA Real-Time Data Feeds — MTA. [Online]. Available: http://datamine.mta.info/.

[8] GTFS - Realtime Reference For The New York City Subway. NYC: MTA, 2012. [Online]. Available: http://datamine.mta.info/sites/all/files/pdfs/GTFS-Realtime-NYC-Subway%20version%201%20dated%207%20Sep.pdf

[9] Developer Guide — Protocol Buffers — Google Developers, Google. [Online]. Available: https://developers.google.com/protocol-buffers/docs/overview.