DriveLaB: An Experimental Platform for Telematics

Kasper F. Pedersen and Kristian Torp
Department of Computer Science
Aalborg University, Denmark
{kasperf,torp}@cs.aau.dk

Abstract—Speed is a major killer in traffic. This paper presents the DriveLaB telematics platform that monitors drivers in real-time and provides feedback if they are speeding. The platform has been developed in collaboration with a major Danish insurance company and used in three larger field trials. The platform uses an Android/iOS smartphone app at the client side and does all data processing at the server-side, e.g., map-matching. The paper first provides an overview of the entire platform and then focuses on three major technical challenges: 1) A trip scoring algorithm that allows for comparison of scores across trip length, 2) A notification system that provides both positive and negative real-time feedback to the drivers, without leading the drivers’ attention away from traffic, and 3) Auto start of the cross-platform app to make it convenient for drivers to participate in the field trials. Three field trials, with 32 participants driving 57,933 km has been conducted. The results of these trials validates that trip scores can be compared across trip length. Further, we demonstrate that doing all data-processing on the server side is a viable approach also for real-time notifications and that drivers are notified with reasonable delays. Finally, we show that the client auto start is fast, robust, and convenient.

Index Terms—platform, smartphone, GPS, traffic, ITS

I. INTRODUCTION

Speed is the major killer in traffic. To get drivers to reduce their speed many information campaigns and education programs have been proposed. This helps for shorter periods, which unfortunately means that speed still today is the cause of many traffic accidents.

A dedicated app on a smartphone has emerged as a good platform for warning drivers about speeding. A smartphone approach has the advantage that most drivers already own one. Further, the deployment of a speed-warning app is simple because drivers are already familiar with installing apps.

This paper presents a complete platform for warning drivers about speeding. The platform consists of a smartphone app and a full set of backend services. From a technology point-of-view the platform relies on existing and upcoming technologies such as high bandwidth, low latency, and cheap wireless communication. Further, the platform relies on 1Hz GPS receivers found in modern smartphones and an effective real-time map-matching algorithm. To potentially reaching a large audience, the platform supports the two major mobile operating systems Android and iOS.

From the driver’s point-of-view the platform is a smartphone app that is easy to setup, is very convenient in daily usage, and provides feedback that is both timely and informative. The app is easy to setup because it is just another app that only requires a one-time setup. The app is convenient to use because the tracking is automatic, i.e., when the app is first setup the driver does not need to do anything to start or stop tracking. The platform provides real-time feedback, i.e., the driver is warned immediately when speeding. Further, all trips are scored such that drivers can track their development and compare their driving skills/scores directly to others.

The platform is fully functional and has been tested for more than two years in three field trials. 32 drivers has participated and delivered 58,000 km and almost 4000 trips. In this paper, we introduce the platform and the results of the three field trials. The main contributions are the following.

- A platform that provides real-time and summary feedback
- A platform that provides positive and negative feedback
- A trip score algorithm that makes it possible to compare trips independently of length
- A thin cross-platform app where map and data-processing all run at the server-side
- A platform that have undergone extensive testing through three field trials
- An experimental detailed evaluation of auto-start, real-time notifications, and trip scoring

The remaining parts of the paper is structured as follows. Section II describes the overall system architecture of DriveLaB. Section III provides implementation details for both the client and the server side. The next three sections, Section IV to Section VI provide the detail for trip scoring, driver notification, and client auto start, respectively. Section VII lists the main results of the three field trials. Section VIII lists related work. Finally, Section IX concludes the paper.

II. SYSTEM ARCHITECTURE

DriveLaB consists of three major system components. A client component designed for both Android and iOS, a middle-end server application, and a backend database.

Fig. 1: DriveLaB System Architecture

In Figure 1, the communication components are represented by squares and the communication flow is shown by arrows. As seen in Figure 1, the client utilizes two different communication technologies. A request/result communication technology is used to retrieve historical data, such as previous trips meta-data and a scoreboard, from the backend. This is done through the middle-end such that storing database login information on the client is avoided. As an extra security measure, only
the middle-end requires database access, which means the database can be hosted on an internal network. A continues bidirectional communication technology is used for real-time communication between the client and the middle-end. GPS records are sent from the client to the middle-end, where the records are map-matched to a road network. Each road contains a speed limit, which is either known in advance or deduced from the road classification. Speeding is detected by comparing the speed limit of the map-matched road against the speed stored in the GPS record. If the driver is speeding, a notification is issued. In addition, trip information such as score, road name, average speed, and traveled distance is sent from the middle-end to the client in real time. All GPS records sent from the client to the middle-end are forwarded to the backend database for permanent storage.

III. IMPLEMENTATION

Various technologies are used to implement the DriveLaB platform. One of these technologies is an HTTP REST API, supporting the request/result communication seen in Figure 1. The API offers five distinct Data Transfer Objects (DTOs) to support communication between the client and the middle-end.

A total of six requests are available through the REST API, two POST and four GET requests. Account creation and account login constitutes the two POST requests. The four GET requests offer retrieval of a driver’s total score, a list of trips, the latest trip, and a cross-driver scoreboard. The list of trips is requested by specifying an offset and size where the result is ordered such offset = 0 and size = 10 will retrieve the 10 oldest trips for a specific driver. By utilizing the offset and size parameters the client is able to lazy load trips while browsing the trip page. Lazy loading trips reduces the data consumption as recent trips are browsed most often.

The continues bidirectional communication is achieved by utilizing the WebSocket protocol [3]. This protocol offers a bidirectional byte-stream and has no predefined message structure. Therefore, a custom data wrapper is introduced such each DTO is wrapped along with its type. The DTO can then be parsed by reading the type. The wrapper also contains a trip identifier used by the middle-end to locate the ongoing trip and update its state.

A notification algorithm is needed to support the real-time feedback mentioned in Section II. The algorithm is implemented on the server-side such that adjustments impact all users simultaneously and no updates of the app is needed. However, network latency may cause notifications to be outdated when received by the client. Therefore, every notification is timestamped such that the client can assess the age of each notification. A notification is ignored, by the client, if it is older than two seconds.

The smartphone client is developed by utilizing the Xamarin framework [5]. This framework is capable of targeting both Android and iOS that both are supported by the DriveLaB platform. Xamarin builds on top of the .NET framework and offers language support for C#, F#, and Visual Basic.NET. The codebase for the DTOs are shared between the client and the middle-end as they are both .NET applications. Sharing the DTO codebase remove concerns of not having the same DTO version on the client and the middle-end. ASP.NET Core is chosen for the middle-end server application as it offers support for Windows, Mac, and Linux while remaining a .NET application. At the time of writing, the middle-end runs on an Ubuntu 14.04 server.

The backend database is a PostgreSQL 10.5 database, with the PostGIS 2.5 extension running on an Ubuntu 18.04 server. Geodata is provided by geofabrik in the .osm.pbf file format composed by OpenStreetMap (OSM). A road network is extracted from the .osm.pbf file by utilizing the OSM2PO tool. The road network is stored in the database where it is used by the map-matching algorithm [10]. The map-matching results are stored along with the associated GPS record. GPS records are batch inserted to reduce concurrent connections and thereby improve scalability. Currently the batch size is set to 50 GPS records, however, the parameter is adjustable. The database schema is a multi star-schema to ease data processing.

IV. TRIP SCORING

The trip scoring algorithm is based solely on the drivers ability to stay under the speed limit. This is deliberate because the score is then easy to understand for the drivers, i.e., the score is not related to fast acceleration or breaking.

In most countries, speeding is punished relative to the magnitude of the speed violation. Therefore, penalties are put into classes spanning over predefined speeding intervals. These classes are the foundation for the scoring algorithm. Adjusting the trip scoring to other penalty classification strategies is trivial, i.e., altering a single C# method.

Figure 2 shows how the scoring algorithm divides speeding into five penalty coefficients. The left axis shows the growth of the speeding percent. The right axis illustrates the speeding classes from the Danish Road Traffic Act. The values in the middle (0, 1, 2, 4, and 8) show the coefficient used in the scoring algorithm. As an example, a speed violation of 65% has a weight of 8. A speed violation of 27% has a weight of 2. Altering these coefficients is trivial. It is a common practice for the police to use a tolerance to counterbalance inaccuracy in the measuring equipment. Furthermore, GPS receivers are also slightly inaccurate. Of these reasons, a 5% tolerance is added to the scoring algorithm. This tolerance is illustrated by the green box at the bottom of Figure 2.

$$s_t = \max \left( \frac{d_t - \sum_{i=1}^{5} d_{ti} * k_i}{d_t} * 10, 0 \right),$$ (1)

where \( k = \{0, 1, 2, 4, 8\} \)

Equation 1 defines the trip score calculation, where \( s_t \) is the score for a given trip \( t \), \( d_t \) is the total distance of trip \( t \), \( d_{ti} \) is the total distance driven in penalty group \( i \), \( \{0, 5, 10, 30, 60, 80\} \) illustrated in Figure 2. Finally, \( k \) is the set of weight coefficients used for scaling the penalty distance, also shown in Figure 2. The scoring algorithm outputs 0 (zero) for any result < 0 and the scoring interval is therefore \( [0, 10] \). The haversine formula is used to calculate the distance between individual GPS locations.
The goal of the scoring algorithm is to make the penalty relative to the length of the trip. This ensures that the driver is able to improve the score while driving under the speed limit even though penalties have been received. Note that using trip length for the trip scoring is simpler than using trip duration, as near-zero speed GPS records should then be discarded.

Figure 3 shows four trips, each visualizing different behaviors of the scoring algorithm. The lines, as well as the coefficients, are colored in accordance with Figure 2. The black numbers show the scoring progress, where the last score is the final trip score. Trip 1 in Figure 3 illustrates that the highest weight coefficient penalizes speeding significantly compared to the lower weight coefficients found in trips 2, 3, and 4. Furthermore, trip 1 shows that a score can be improved due to the penalties being relative to the length of the trip. From a driver’s perspective, the score cannot be less than 0. However, the scoring system internally allows for a negative score such that a score well below 0 is treated differently than a score just below 0. An example of this can be seen on the last three kilometers of trip 2. Trips 3 and 4 show that the same score can be achieved as long as the distribution of speeding coefficients remains the same.

By grouping all trips for a driver, a single total score can be calculated. Such a total score makes it possible to compare drivers and thereby rank them on a scoreboard. Calculating a driver’s total score, only require altering the summation in Equation 1 such that \( \sum_{i=1}^{n} k_i \cdot \sum_{t=1}^{n} d_{ti} \), where \( n \) is the number of trips.

V. Notification

The notification system is based on the "carrot and stick" principle. The driver is praised if driving longer distances without speeding and warned in real-time when speeding.

The praising uses Google’s Text-to-Speech (TTS) Android API [4]. A similar TTS API is used for the iOS implementation [1]. The TTS API reads a text string sent from the middle-end, such that the message can be easily changed. Currently, the message is "you have driven under the speed limit for x kilometers, well done!" where \( A = \{4, 8, 16, 32, ... \} \) and \( x \in A \). The text string is received at the kilometer intervals represented by \( A \). The \( x \) value is reset to 4 km whenever the speed limit is violated.

A warning is issued when the driver exceeds the speed limit by 5, 30, or 60 percent. These values are chosen in accordance to the Danish Road Traffic Act as shown in Figure 2. The warnings are presented to the driver as three distinct sounds played on the smartphone. The two following principles are used for issuing warnings.

- Be as silent as possible - Issuing too many sound warnings distracts the driver. Further, it can provoke the driver to disable the sound notifications or uninstall the app.
- Be as fair as possible - Issuing unfair warnings could make the driver distrust the system and therefore not lowering the speed when warned.

These principles are used to develop the six policies discussed in the following.

A. Warning Policies

The primary goal, of the notification system, is to be trustworthy such that warnings causes driver to lower the speed when exceeding the speed limit. Therefore, a notification system should warn the driver cautiously and not exaggerate the use of sounds. The following six policies are developed to accomplish such a notification system.

1) Decreasing the current penalty class: Oscillating between two penalty classes should not cause warnings. To achieve this, a current penalty class (cpc) variable is introduced. The cpc variable stores the penalty class of the last warning issued, with an initial value of 0. A warning is only issued if its penalty class exceeds the cpc. A side-effect of this is that each of the three warnings can only be received once if the cpc cannot be decreased. Therefore, the cpc is decreased, when the speed has stabilized at a lower penalty class. For decreasing the cpc a buffer \( b \) is introduced. The buffer is illustrated using the following array notation [old, newer, newest]. Each GPS record received is converted to a penalty class and stored in the buffer. The buffer size inflicts a delay in decreasing the cpc. The larger the buffer the longer the delay. Experimentation showed \( |b| = 5 \) performs satisfactory and is therefore used throughout the study. All the penalty classes stored in the buffer must be lower than the cpc in order for the cpc to be decreased.

Consider the function \( f(cpc, b) = (npc, ps) \) where npc is a new penalty class and ps is a boolean that indicates if a sound should be played.
Equation 2 do not satisfy $\max(b) < cpc$ where $\max(b)$ outputs the maximum value found in a buffer $b$. The penalty class output by Equation 2 is therefore equal to the $cpc$ input. However, the output of Equation 3 is decreased as $\max([1, 0, 0, 1, 1]) = 1$ and $1 < 2$. Assigning $x$ to 0, 1, 2, or 3 will yield the same output in Equation 4. This illustrates that the penalty class is not required to decrease gradually.

2) Delay warnings at startup: Warnings are withheld until the buffer is full. This is due to the real-time map-matching algorithm is less accurate in the initialization phase.

$$f(3, [3, 2, 1, 0, 2]) = (3, false) \quad (2)$$
$$f(2, [1, 0, 0, 1, 1]) = (1, false) \quad (3)$$
$$f(x, [0, 0, 0, 0, 0]) = (0, false) \quad (4)$$

Equation 2 do not satisfy $\max(b) < cpc$ where $\max(b)$ outputs the maximum value found in a buffer $b$. The penalty class output by Equation 2 is therefore equal to the $cpc$ input. However, the output of Equation 3 is decreased as $\max([1, 0, 0, 1, 1]) = 1$ and $1 < 2$. Assigning $x$ to 0, 1, 2, or 3 will yield the same output in Equation 4. This illustrates that the penalty class is not required to decrease gradually.

2) Delay warnings at startup: Warnings are withheld until the buffer is full. This is due to the real-time map-matching algorithm is less accurate in the initialization phase.

$$f(0, [1, 1, -,-,-]) = (0, false) \quad (5)$$
$$f(0, [1, 1, 1, 1, 1]) = (1, true) \quad (6)$$

In Equation 5 the symbol $\text{null}$ represent a null value. Therefore, only Equation 6 satisfies the condition $\exists pc \in b : pc \text{ is null}$, issuing a warning to the driver.

3) None-Consecutive GPS Records: The GPS signal can be lost while driving through a tunnel or a forest. Therefore, the buffer is able to contain penalty classes based on none-consecutive GPS records. However, warning must be based on consecutive GPS records to avoid treating the driver unfairly. To solve this issue, a maximum buffer age (mba) argument is added such $f'(cpc, b, mba) = (npc, ps)$. Each element in $b$ is assigned the timestamp from the corresponding GPS record. Consider a function $age(b)$ that outputs the time-span between the first and the last timestamp in $b$. Then $f'$ must satisfy the expression $age(b) \leq mba$ for a warning to be issued. Simple experiments showed 7 seconds to be an appropriate value for the $mba$ parameter, for a buffer size of 5.

$$f'(0, [1, 1, 1, 1, 1], 7) = (0, false) \quad (7)$$
$$f'(0, [1, 1, 1, 1, 1], 7) = (1, true) \quad (8)$$

In Equation 7, the age of the buffer is greater than 7 and the warning is therefore withheld. The $cpc$ will therefore remain unchanged. However, in Equation 8 the age of the buffer equals 7, resulting in a warning.

4) Prolonged violation: Each penalty class is calculated using the current speed and a speed limit deduced from the map-matching result. However, map-matching algorithms do not guarantee a correct result.

Therefore, the likelihood of a warning to be faulty increases as the consecutive number of penalty classes used to compute the warning decreases. On the other hand, increasing the number of penalty classes also increases the delay of the warning. To address these two issues a new parameter $sv$ is introduced such $f''(cpc, b, mba, sv) = (npc, ps)$. The $sv$ parameter defines how many of the newest buffer elements are used to compute if a warning should be given. For illustration purposes, the Python-like notation $b[a : b]$ is introduced. $a$ defines an array offset where $b$ defines a limit such if $|b| = 5$ then $b[2 : 5]$ returns the last 3 array elements. Using this notation we can formalize the following condition $\forall pc \in b[b] - sv : |b| : pc > cpc$. A warning is withhold ($ps = \text{false}$) if the condition is not satisfied.

$$f''(0, [1, 1, 0, 0, 1], 7, 3) = (0, false) \quad (9)$$
$$f''(0, [1, 1, 1, 1, 1], 7, 3) = (1, true) \quad (10)$$

When setting $sv = 3$, the newest 3 penalty classes must agree that the driver is speeding. Equation 9 exemplifies a suppressed warning due to insufficient knowledge that the driver is speeding. Equation 10, however, shows that if the latest $sv$ elements are indicating speed violation, a warning is issued.

5) Conservative warnings: A new issue arise when the latest $sv$ penalty classes vary in severity, however all indicates speeding. The driver should be warned cautiously, such that anything but the least severe warning could result in the driver being treated unfairly.

$$f''(0, [0, 0, 1, 2, 3], 7, 3) = (1, true) \quad (11)$$
$$f''(0, [0, 0, 2, 3, 3], 7, 3) = (2, true) \quad (12)$$
$$f''(0, [0, 0, 3, 3, 3], 7, 3) = (3, true) \quad (13)$$

Equation 11, 12 and 13 adopts the aforementioned approach such $npc = \min(b[b] - sv : |b|)$.

6) Remind the Driver: A driver may not notice a warning due to music played on the radio or other background noise. If a driver does not react to a warning and continues to be speeding, it is assumed that the warning was not heard. A new warning is then issued to remind the driver about the speeding. These warnings, are only issued if the driver has been speeding without any additional warning for a longer period.

This functionality is enabled by introducing a time threshold $tt$ such $f'''(cpc, b, mba, sv, tt) = (npc, ps)$. A new warning is sent if the time since the last notification exceeds the threshold as long as $npc > 0$. The $tt$ parameter is set to 30 seconds throughout the study.

B. The Warning Algorithm

The six policies presented are all convoluted into Algorithm 1. Line 4 to 6 ensures that warning are only issued if the buffer is full as described in Section V-A2. Furthermore, the buffer is ensured to contain consecutive data such the policy in Section V-A3 is fulfilled. Lines 7 to 9 ensure that warnings are based on multiple data points in accordance to Section V-A4. Also, the policy found in Section V-A5 ensuring a fair treatment of the driver is handled at these lines. Decreasing the $cpc$ in accordance to Section V-A1 is done at line 10 and 11. Finally, line 15 to 17 enforce the policy found in Section V-A6.
An auto-start functionality is added to the DriveLaB platform such that setting up the app is the only effort required by the driver. Furthermore, an increased number of trips are likely to be logged when starting the app is automated.

Android permits code execution based on events also known as broadcast actions. One of these actions occurs when a Bluetooth device connects (ACL_CONNECTED) [2], and another when a Bluetooth device disconnects (ACL_DISCONNECTED) [2]. In-vehicle Bluetooth equipment can be used for client auto start as long as they can connect automatically to the phone. Such equipment is pre-installed in most modern vehicles. Otherwise, inexpensive alternatives (e.g., plugging an adaptor into the cigar lighter socket) can be used for client auto start as long as they can connect or disconnects, which is infrequently.

Waking apps frequently using broadcast actions is generally discouraged. This is due to the likelihood of causing memory thrashing on a low memory system. The broadcast actions used by DriveLaB are only sent when Bluetooth equipment connects or disconnects, which is infrequently.

Within the app, the user is able to see all Bluetooth devices paired to the phone. Each of these devices can be marked as a known device. When a Bluetooth device connects, it is compared to the known device. If identical, a new trip is started and GPS records are sent to the server. Similarly, when a Bluetooth device disconnects, it is compared to the known device. If identical, the current trip is stopped and the connection to the server is closed.

### Algorithm 1: Get penalty class algorithm

```plaintext
Algorithm 1: Get penalty class algorithm

1: function GETPENALTYCLASS(\(cpc, b, mba, sv, tt\))
2: \(\text{ps} \leftarrow \text{false}\)
3: \(\text{npe} \leftarrow 0\)
4: if \(\exists \text{pc} \in b : \text{pc} \notin \text{npe} \land \text{age}(b) > \text{mba}\) then
5: \(\quad \text{return} \langle \text{npe}, \text{ps} \rangle\)
6: end if
7: \(\quad \text{npe} \leftarrow \text{min}(b[|b| - \text{sv} : |b|])\)
8: if \(\text{npe} > \text{cpc}\) then
9: \(\quad \text{ps} \leftarrow \text{true}\)
10: else if \(\text{max}(b) \leq \text{cpc}\) then
11: \(\quad \text{npe} \leftarrow \text{max}(b)\)
12: else
13: \(\quad \text{npe} \leftarrow \text{cpc}\)
14: end if
15: if \(\text{timeSinceLastWarning()} > \text{tt} \land \text{npe} > 0\) then
16: \(\quad \text{ps} \leftarrow \text{true}\)
17: end if
18: return \langle \text{npe}, \text{ps} \rangle
19: end function
```

### VI. Client Auto Start

Three field trials have been done in collaboration with a large Danish insurance company. The trials were conducted in 2016, 2017, and 2018 spanning approximately 60 days. Further, the client app has been available through Google Play and Apple’s App Store since 2017. Table I shows the basic statistics about the field trials. There are 32 active participants, which have at least one trip. These participants have driven 3,976 trips and collected 9,514,919 GPS records over the 57,933 km driven. As mentioned in Section III, a notification is disregarded, by the client, if it is more than 2 seconds old. Table I, therefore, divides notifications into \(\text{Played notifications}\), which are played by the client, and \(\text{All notifications}\) including ignored notifications. 21% of notifications are considered too old. This shows that sending notifications over a modern cellular network does not add a significant delay. The approach of doing all data processing on the server-side is therefore deemed feasible for an experimental platform such as DriveLaB.

The age and gender distribution of the 32 active participants is seen in Figure 5. Please note that a user may appear more than once as the platform has tested over a 3 year period.

### A. Trip Score Evaluation

Section IV states that drivers can be directly compared by calculating a single aggregated score for each. This requires that the trip score can be compared across trip lengths.

Figure 6 groups trips into length intervals to evaluate if the trip length has an impact on the score. The trips scoring the least are found in the 5 to 10 km interval. The best scoring

![Fig. 4: Flowchart of Android Client Auto Start](image)

**TABLE I: Field Trial Statistics**

| Metric       | Count  |
|--------------|--------|
| Active participants | 32     |
| GPS points    | 9,514,919 |
| Trips         | 3,976  |
| Kilometers    | 57,933 |
| All notifications | 14,480 |
| Played notifications | 11,476 |

256
trips are found in the 0 to 5 km interval. However, the median remains in the 9 to 10 score range across all trip lengths. The low variance in the median score indicates that the trip length does not impact the score significantly.

Similar to the trip length, the approach for calculating an aggregated score does not account for day of the week. Therefore to compare the score of the drivers it is important that all workdays can be compared. Figure 7 groups trips based on weekday. Similar to the trip length, the median score remains in the 9 to 10 range across all days. Based on these findings, we conclude that all workdays are comparable. Please note that both Saturday and Sunday have less variance in the lower quartile. Also, both of these days have a significantly lower minimum value than the workdays. This could indicate that people drive safer during weekends, however, a greater sample size is needed to be conclusive.

A trip score is based on GPS records received from a smartphone. With a growing user base, a variety of phone models and makes starts to appear. Therefore, it is important that trips are comparable, across smartphone makes and models. To evaluate this, multiple trips has been conducted, each tracked with at least two different phones. The resulting trip scores are plotted in Figure 8. The plot shows an maximum inaccuracy of 5%. For experiments where money is not involved between the drivers and the insurance company we clearly find that this is a sufficient accuracy.

**B. Notification Evaluation**

In Section V two principles are presented. The first principle dictates that the number of notifications sent to the driver should not remove the focus from driving or provoke disabling the notifications. On the client, notifications can be disabled based on the notification type (penalty class), i.e., PC1, PC2, or PC3.

Figure 9 shows how the app is configured for each notification issued. For both PC1 and PC2 around 13% of the notifications issued are actively deactivated. Additionally, 17% of the notifications did not issue a sound due to the phone being in silence mode. This leaves approximately 70% producing a sound. The increase of sound deactivations seen for PC3 may be caused by other factors than over exposure because this is the least issued warning class, as seen in Figure 10. One possible explanation is that the police-siren sound actually used for these warning class is too loud for some drivers.

Figure 10 shows that notifications for PC1 is by far the most issued notification type. This is as expected and could indicate that the notifications may inflict a speed reduction and thereby not reaching a higher penalty class.

75% of all trips are represented by Figure 11. This leaves 25% of all trips having more than 10 minutes between each warning. Figure 11 shows that warnings are issued every 9 minute for the majority of trips. Only a few trips has a warning issued by a 1 minute interval.

Figure 12 show the average duration of a trip. As can be seen more than half the trips are less than 20 minutes, which on average would mean to notifications per trip.

We conclude that issuing a warning every 9 minute performs well in practice because there are only few sound deactivations as seen in Figure 9.
C. Client Auto-Start Evaluation

With regards to the client auto start, no issues has been reported during the field trials by the drivers. However, internal quality control revealed that the Bluetooth equipment powered by a car’s ignition can cause problems in rare cases due to the internal workings of the ignition system.

An ignition switch has four stages, Lock, Accessory, On, and Start. Figure 13 illustrates the problem that can arise when transitioning from the On to the Start state. In this transition, all electronics equipment loose power while the engine is being started, including the Bluetooth equipment. However, the power is returned shortly making the Bluetooth device issue a new connection before the phone has terminated the first connection.

To test the responsiveness of the Bluetooth solution, two controlled experiments is conducted. The first experiment consists of 10 connections and disconnections on an idle smartphone. On average, the client app started 2 seconds after the Bluetooth connection and terminated 5 seconds after the disconnection. The second experiment consists of 8 connections and 8 disconnections on an rebooted smartphone. The client app starts on average 6 seconds slower compared to the experiment done on an idle smartphone. App termination is delayed by 4 seconds on average and thereby yield similar results compared to the idle smartphone.

Bluetooth equipment creating a connection based on accelerometer readings are found to start the trips as early as entering the car.

D. Battery Consumption

For the DriveLaB app to be convenient to use, it is important that it does not drain the battery. Figure 14 shows how fast the
A regression line is plotted to show the expected discharge rate. A solid line shows the discharge for a phone with the screen turned on and a dashed line when the screen is turned off. The Huawei P9 outperforms the Huawei P8 even though the screen is turned on. This is most likely due to the smaller and older battery found in the P8. However, both phones can be expected to last longer than 10 hours. This is clearly acceptable as an average person spends around one hour in a car a day in Denmark. The iPhone 6 yields significantly worse battery performance than the two Android phones, however, it is also the phone with least battery capacity. Almost four hours can be expected for the DriveLaB app to run on the iPhone 6.

VIII. RELATED WORK

Previously, DriveLaB has been demoed as a platform for reducing speeding [9]. However, the purpose of this current paper is to provide a technical details about three major components of DriveLaB, i.e., trip scoring, notifications, and client auto start.

A study shows how sensors available in today’s smartphones can be used for a client auto-start system [6]. The study shows accuracies greater than 85%, and can distinguish between both a driver and a passenger. The client auto-start presented in this study, utilizes generic external Bluetooth equipment. An event-driven Bluetooth implementation is chosen over the sensor based due to its simplicity. Further, newer versions of Android and iOS have decreased support for persistent background services to improve battery life. This complicates the censor based solution further.

Mapping cell-tower based trajectories to a transportation network has been investigated [7]. Such implementations offer better battery performance compared to utilizing GPS receivers. However, the cell-tower based trajectories are less accurate, and therefore additional historical information is used for improving the precision. The cell-tower approach is good for analyzing the main traffic-flow, but it is too imprecise for selecting speed limits on individual roads.

Nericell [8], is a project conducted by Microsoft Research, aiming at detecting potholes, bumps, braking, and honking using a mobile platform. These detections are achieved using sensors already present in a modern smartphone. Similar to DriveLaB, the smartphone is sending the data to a server. Different from DriveLaB the data is pre-processed at the client (phone) before being sent. Furthermore, DriveLaB offers bidirectional communication such that information can be sent back to the client at any given time.

IX. CONCLUSION

This paper has presented the DriveLaB platform that makes it cheap, simple, and informative for drivers to participate in experiments that focuses on speeding. The platform is a modern IT version of the physical signs along roads related to road safety.

The DriveLaB platform consists of an app that runs both on Android and iOS and a set of server-side services. Once the app is setup, automatic tracking via Bluetooth makes it very convenient for drivers to participate because there is no manual start/stop of trips. Auto-tracking is available for all Android smartphones as a generic solution. However, auto-tracking on an iOS smartphone requires additional hardware.

The services on the server side do all the computation, e.g., map-matching and trip scoring. There is only one copy of the map stored server side. This makes the app very thin and ensure all drivers use the same map.

The platform has extensively tested in three field trials. The main results of the field trial are that the auto-tracking is robust. We show that the trip-score algorithm makes it possible to compare scores across trip lengths and speed limits, which is highly relevant for insurance companies. Based on 14,480 notifications we conclude that doing all computation on the server-side is a viable solution for an experimental platform. Only 24% of notifications are not played due to network latency. Please note that server-side services are highly relevant in today’s cloud-computing setup.

In summary, the DriveLaB platform is fully implemented and has been running for more than three years. It is thus a stable platform for conducting field trials related to the telematics area.

Future work includes scaling experiments to thousands of users. From a technical perspective this requires caching the map in main memory. We estimate that this can scale the map-matching algorithm performance by two to three orders of magnitude as network and disk latencies are eliminated. Further, relying on a micro-service architecture can increase the horizontal scalability of the platform. Such a service could be used to extent the platform, e.g., with crowd-sensing functionality to collect speed limits from the drivers.

ACKNOWLEDGEMENT

This work is supported by the ELVITEN project www.elviten-project.eu grant agreement No 769926.

REFERENCES

[1] AVSpeechSynthesizer AVFoundation Apple Developer Documentation. https://developer.apple.com/documentation/avfoundation/avspeechsynthesizer. (Accessed 2019-05-01).

[2] BluetoothDevice — Android Developers. https://developer.android.com/reference/android/bluetooth/BluetoothDevice. (Accessed 2019-05-01).

[3] RFC 6455 - The WebSocket Protocol. https://tools.ietf.org/html/rfc6455. (Accessed 2019-05-01).

[4] TextToSpeech — Android Developers. https://developer.android.com/reference/android/speech/tts/TextToSpeech. (Accessed 2019-05-01).

[5] Xamarin App Development with Visual Studio — Visual Studio. https://visualstudio.microsoft.com/xamarin/. (Accessed on 2019-05-01).

[6] Chu, V., Raman, J., Shen, A., Kansal, V., Bahl, and R. R. Choudhury. I am a smartphone and I know my user is driving. In Communication Systems and Networks (COMSNETS), pages 1–8, 2014.

[7] Dash, K. K., Koo, T., Holleczek, G., Yap, S. P., Krishnaswamy, and A. Shi-Nash. From mobile phone data to transport network—gaining insight about human mobility. In MDM, pages 243–250, 2015.

[8] P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In Proceedings of the 6th ACM conference on Embedded network sensor systems, pages 323–336. ACM, 2008.

[9] T. F. Olsen, K. F. Pedersen, D. Rasmussen, and K. Torp. DriveLab: A platform for reducing speeding. In MDM, pages 274–275, 2018.

[10] N. R. Velaga, M. A. Quddus, and A. L. Bristow. Developing an enhanced weight-based topological map-matching algorithm for intelligent transport systems. Transportation Research Part C: Emerging Technologies, 17(6):672–683, 2009.