Crowdsensing Based Public Transport Information Service in Smart Cities

Károly Farkas, Gábor Fehér, András Benczúr, and Csaba Sidló, *Member, IEEE*

***Abstract*—Thanks to the development of technology and the emergence of intelligent services smart cities promise to their inhabitants enhanced perception of city life. For example, a live timetable service of public transportation can increase the efficiency of travel planning substantially. However, its implementation in a traditional way requires the deployment of some costly sensing and tracking infrastructure. Mobile crowdsensing is an alternative, when the crowd of passengers and their mobile devices are used to gather data for almost free of charge.**

In this paper, we put the emphasis on the introduction of our crowdsensing based public transport information service, what we have been developing as a prototype smart city application. The front-end interface of this service is called TrafficInfo. It is a simple and easy-to-use Android application which visualizes real-time public transport information of the given city on Google Maps. The lively updates of transport schedule information relies on the automatic stop event detection of public transport vehicles. TrafficInfo is built upon our Extensible Messaging and Presence Protocol (XMPP) based communication framework what we designed to facilitate the development of crowd assisted smart city applications. The paper introduces shortly this framework, than describes TrafficInfo in detail together with the developed stop event detector.

***Index Terms*—Crowdsensing, Public transport, GTFS, Publish/subscribe, XMPP, Android, Smart cities**

I. INTRODUCTION

SERVICES offered by smart cities aim to support the everyday life of inhabitants. Unfortunately, the traditional way of introducing a new service usually implies a huge investment to deploy the necessary background infrastructure.

One of the most popular city services is public transportation. Maintaining and continuously improving such a service are imperative in modern cities. However, the implementation of even a simple feature which extends the basic service functions can be costly. For example, let's consider the replacement of static timetables with lively updated public transport information service. It requires the deployment of a vehicle tracking infrastructure consisting of among others GPS sensors, communication and back-end informatics systems and user interfaces, which can be an expensive investment.

Manuscript submitted on October 19, 2014, revised on December 8, 2014.

K. Farkas is with the Inter-University Centre for Telecommunications and Informatics, Debrecen, Hungary and the Budapest University of Technology and Economics, Budapest, Hungary (corresponding author, e-mail: farkask@hit.bme.hu).

G. Fehér is with the Inter-University Centre for Telecommunications and Informatics, Debrecen, Hungary and the Budapest University of Technology and Economics, Budapest, Hungary (e-mail: feher@tmit.bme.hu).

A. Benczúr and Cs. Sidló are with the University of Debrecen, Hungary and the Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary (e-mail: benczur@info.ilab.sztaki.hu, sidlo@sztaki.hu).

An alternative approach to collect real-time tracking data is exploiting the power of the crowd via participatory sensing or often called mobile crowdsensing[1] [1], which does not call for such an investment. In this scenario (see Fig. 1), the passengers' mobile devices and their built-in sensors, or the passengers themselves via reporting incidents, are used to generate the monitoring data for vehicle tracking and send instant route information to the service provider in real-time. The service provider then aggregates, cleans, analyzes the data gathered, and derives and disseminates the lively updates. The sensing task is carried out by the built-in and ubiquitous sensors of the smartphones either in participatory or opportunistic way depending on whether the user is involved or not in data collection. Every traveler can contribute to this data harvesting task. Thus, passengers waiting for a ride can report the line number with a timestamp of every arriving public transport vehicle at a stop during the waiting period. On the other hand, onboard passengers can be used to gather and report actual position information of the moving vehicle and detect halt events at the stops.

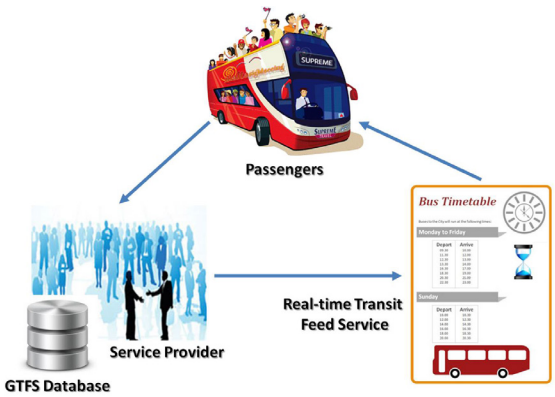


Fig. 1. Real-time public transport information service based on mobile crowdsensing

In this paper, we focus on the introduction of our crowdsensing based public transport information service, what we have been developing as a prototype smart city application. The front-end interface of this service, called TrafficInfo, is a simple and easy-to-use Android application which visualizes real-time public transport information of the given city on Google Maps. It is built upon our Extensible Messaging and Presence Protocol (XMPP) [2] based communication framework [3]

[1]We use the terms *crowdsensing*, *crowdsourcing* and *participatory sensing* interchangeably in this paper.

what we designed to facilitate the development of crowd assisted smart city applications (we also introduce shortly this framework in Sec. III). Following the publish/subscribe (pub/sub) communication model the passengers subscribe in TrafficInfo, according to their interest, to traffic information channels dedicated to different public transport lines or stops. Hence, they are informed about the live public transport situation, such as the actual vehicle positions, deviation from the static timetable, crowdedness information, etc.

To motivate user participation in data collection we offer a day zero service to the passengers, which is a static public transportation timetable. It is built on the General Transit Feed Specification (GTFS, designed by Google) [4] based transit schedule data and provided by public transport operators. TrafficInfo basically presents this static timetable information to the users which is updated in real-time, if appropriate crowdsensed data is available. To this end, the application collects position data; the timestamped halt events of the public transport vehicles at the stops (our automatic detector is described in Sec. IV-D); and/or simple annotation data entered by the user, such as reports on crowdedness or damaged seat/window/lamp/etc. After analyzing the data gathered live updates are generated and TrafficInfo refreshes the static information with them.

The rest of the paper is structured as follows. After a quick overview of related work in Sec. II we introduce shortly in Sec. III our generic framework to facilitate the development of crowdsourcing based services. In Sec. IV, we show our live public transport information service together with the developed stop event detector. Finally, in Sec. V we summarize our work with a short insight to our future plans.

II. Related Work

In this section, we discuss the challenge of attracting users to participate in crowdsensing and review the relevant works in the field of crowd assisted transit tracking systems.

A crowdsourcing based service has to acquire the necessary information from its users who are producers and consumers at the same time. Therefore it is essential for the service provider to attract users. However, we face a vicious circle here. The users are joining the service if they can benefit from it and at the same time they contribute to keep running the service which can persuade others also to join. But how can the users be attracted if the service is not able to provide the expected service level due to the lack of contributors? This also means that the service cannot be widely spread without offering a minimum service level and until it has a sufficiently large user base.

Moovit[2] is a similar application to TrafficInfo which is meant to be a live transit app on the market providing real-time information about public transportation. It faces the above mentioned problem in many countries. Moovit has been successful only in those cities where it has already a mass of users, just like in Paris, and not successful in cities where its user base is low, e.g., in Budapest. In order to create a sufficiently large user base Moovit provides, besides live data, schedule based public transportation information as a day zero service, too. The source of this information is the company who operates the public transportation network. The best practice is for providing such information is using GTFS [4]. According to the GTFS developer page, currently 263 public transportation companies provide transit feeds from all over the world. Moovit partially relies on GTFS and is available in 350 cities attracting more than 6.5 million users. We also adopted this solution in TrafficInfo.

Several other mobile crowdsensing based transit tracking ideas have been published recently. For instance, the authors in [5] propose a bus arrival time prediction system based on bus passengers' participatory sensing. The proposed system uses movement statuses, audio recordings and mobile cell-tower signals to identify the vehicle and its actual position. The authors in [6] propose a method for transit tracking using the collected data of the accelerometer and the GPS sensor on the users' smartphone. The authors in [7] use smartphone sensors data and machine learning techniques to detect motion type, e.g., traveling by train or by car. EasyTracker [8] provides a low cost solution for automatic real-time transit tracking and mapping based on GPS sensor data gathered from mobile phones which are placed in transit vehicles. It offers arrival time prediction, as well.

These approaches focus on the data (what to collect, how to collect, what to do with the data) to offer enriched services to the users. However, our focus is on how to introduce such enriched services incrementally, i.e., how can we create an architecture and service model, which allows incremental introduction of live updates from participatory users over static services that are available in competing approaches. Thus, our approach complements the above ones.

III. Framework for Crowdsensing Based Smart City Applications

In this section, we shortly describe our generic framework [3], which is based on the XMPP publish-subscribe architecture, to aid the development of crowdsensing based smart city applications. TrafficInfo is implemented on top of this framework.

A. Communication Model

XMPP [2] is an open technology for real-time communication using Extensible Markup Language (XML) [9] message format. XMPP allows sending of small information pieces from one entity to another in quasi real-time. It has several extensions, like multi-party messaging [10] or the notification service [11]. The latter realizes a publish/subscribe (pub/sub) communication model [12], where publications sent to a node are automatically multicast to the subscribers of that node. This pub/sub communication scheme fits well with most of the mobile crowdsensing based applications. In these applications, the users' mobile devices are used to collect data about the environment (publish) and the users consume the services updated on the basis of the collected data (subscribe).

Hence, we use XMPP and its generic publish/subscribe communication model in our framework to implement interactions. In this model, we define three roles, like *Producer*,

[2]http://www.moovitapp.com/

Consumer and *Service Provider* (see Fig. 2). These entities interact with each other via the core service, which consists of event based pub/sub nodes.

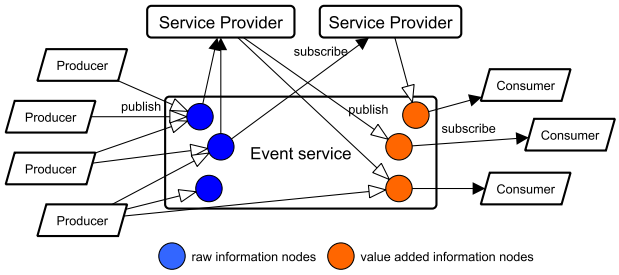


Fig. 2. Crowdsensing model based on publish/subscribe communication

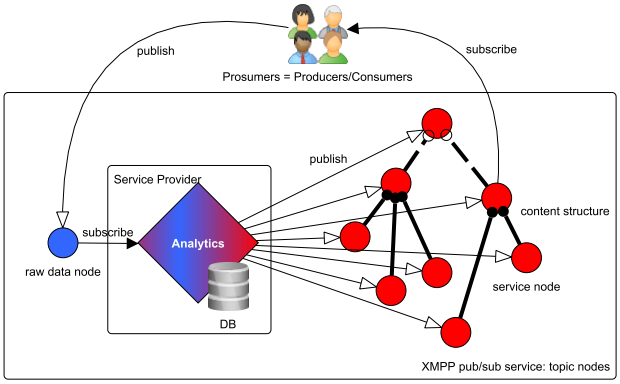


Fig. 3. Mobile crowdsensing: the publish/subscribe value chain using XMPP

Producer: The Producer acts as the original information source in our model producing raw data streams and plays a central role in data collection. He is the user who contributes his mobile's sensor data.

Consumer: The Consumer is the beneficiary of the provided service(s). He enjoys the value of the collected, cleaned, analyzed, extended and disseminated information. We call the user as *Prosumer*, when he acts in the service as both Consumer and Producer at the same time.

Service Provider: The Service Provider introduces added value to the raw data collected by the crowd. Thus, he intercepts and extends the information flow between Producers and Consumers. A Service Provider can play several roles at the same time, as he collects (Consumer role), stores and analyzes Producers' data to offer (Service Provider role) value added service.

In our model, depicted in Fig. 2, Producers are the source of original data by sensing and monitoring their environment. They publish (marked by arrows with empty arrowhead) the collected information to event nodes (raw information nodes are marked by blue dots). On the other hand, Service Providers intercept the collected data by subscribing (marked by arrows with black arrowhead) to raw event nodes and receiving information in an asynchronous manner. They extend the crowdsensed data with their own information or extract cleaned-up information from the raw data to introduce added value to Consumers. Moreover, they publish their service to different content nodes. Consumers who are interested in the reception of the added value/service just subscribe to the appropriate content node(s) and collect the published information also in an asynchronous manner.

B. Architecture

We can directly map this model to the XMPP publish/subscribe service model as follows (see Fig. 3):

- Service Providers establish raw pub/sub data nodes, which gather Producers' data, for the services they offer.
- Consumers can freely publish their collected data to the corresponding nodes with appropriate node access rights, too. However, only the owner or other affiliated Consumers can retrieve this information.
- Producers can publish the collected data or their annotations to the raw data nodes at the XMPP server only if they have appropriate access rights.
- Service Providers collect the published data and introduce such a service structure for their added value via the pub/sub subscription service, which makes appropriate content filtering possible for their Consumers.
- Prosumers publish their sensor readings or annotations into and retrieve events from XMPP pub/sub nodes.
- Service Providers subscribed to raw pub/sub nodes collect, store, clean-up and analyze data and extract/derive new information introducing added value. This new information is published into pub/sub nodes on the other side following a suitable structure.

The pub/sub service node structure can benefit from the aggregation feature of XMPP via using collection nodes, where a collection node will see all the information received by its child nodes. Note, however, that the aggregation mechanism of an XMPP collection node is not appropriate to filter events. Hence, the Service Provider role has to be applied to implement scalable content aggregation. Fig. 3 shows XMPP aggregations as dark circles at the container node while empty circles with dashed lines represent only logical containment where intelligent aggregation is implemented through the service logic.

IV. Real-time Public Transport Information Service

In this section, we shortly overview the architecture of our public transport information service, then describe TrafficInfo, its front-end Android interface together with our stop event detector.

A. Service Architecture

Our real-time public transport information service architecture has two main building blocks, such as our crowdsensing framework described in Sec. III and the TrafficInfo application (see Fig. 4). The framework can be divided into two parts, a standard XMPP server and a GTFS Emulator with an analytics module.

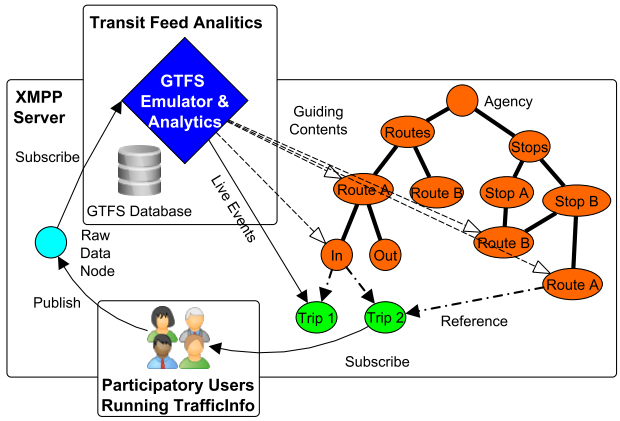


Fig. 4. Real-time public transport information service architecture

The XMPP server maps the public transport lines to a hierarchical pub/sub channel structure. We turned the GTFS database into an XMPP pub/sub node hierarchy. This node structure facilitates searching and selecting transit feeds according to user interest.

Transit information and real-time event updates are handled in the *Trip* nodes at the leaf level. The inner nodes in the node hierarchy contain only persistent data and references relevant to the trips. The users can access the transit data via two ways, based on *routes* or *stops*. When the user wants to see a given trip (vehicle) related traffic information the route based filtering is applied. On the other hand, when the forthcoming arrivals at a given stop (location) are of interest the stop based filtering is the appropriate access way.

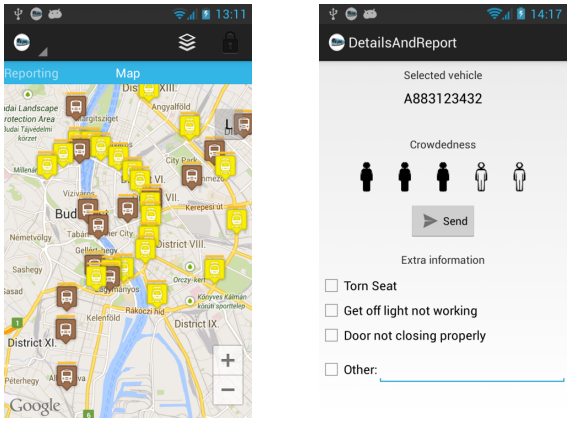
The GTFS Emulator provides the static timetable information, if it is available, as the initial service. It basically uses the officially distributed GTFS database of the public transport operator of the given city. However, it also relies on another data source, which is OpenStreetMap (OSM), a crowdsourcing based mapping service [13]. In OSM maps, users have the possibility to define terminals, public transportation stops or even public transportation routes. Thus, the OSM based information is used to extend and clean the information coming from the GTFS source. The analytics module is in charge of the business logic offered by the service, e.g., deriving crowdedness information or estimating the time of arrivals at the stops from the data collected by the crowd.

TrafficInfo handles the subscription to the pub/sub channels, collects sensor readings, publishes events to and receives updates from the XMPP server, and visualizes the received information.

B. TrafficInfo Features

TrafficInfo has three main features, but most of the users will benefit from its visualization capability that visualizes public transport vehicle movements on a city map.

1) Visualization: An example of this primary feature can be seen on Fig. 5a displaying trams 1, 4, 6 and buses 7 and 86 on the Budapest map in Hungary. The depicted vehicles can be filtered to given routes. The icon of a vehicle may reflect various attributes, such as the number, progress or crowdedness of the specific vehicle. Clicking on a vehicle's icon a popup shows all known information about that specific vehicle.



(a) Vehicle visualization (b) User feedback form

Fig. 5. TrafficInfo screenshots

2) Information Sharing: The second feature is about information sharing. Passengers can share their observations regarding the vehicles they are currently riding. Fig. 5b shows the feedback screen that is used to submit the observations. The feedback information is spread out using our crowdsourcing framework and displayed on the devices of other passengers, who might be interested in it. It is up to the user what information and when he wants to submit, but we are planning to provide incentives to use this feature frequently.

3) Sensing: The third feature is collecting smartphone sensor readings without user interaction, which is almost invisible for the user. User positions are reported periodically and are used to determine the vehicle's position the passenger is actually traveling on. In order to create the link between the passenger and the vehicle, we try to identify the movement of the user through his activities. To this end we are using various sensors, e.g., accelerometer, and try to deduct the timestamped stop events of the vehicles (our automatic stop event detection mechanism is described in Sec. IV-D). The duration between the detected stops coupled with GPS coordinates identifies the route segment, which the user actually rides.

Besides the GPS coordinates Google also provides location information on those areas, where there is no GPS signal. Usually this position is highly inaccurate, but the estimated accuracy is also provided. We also use the activity sensor, which guesses the actual activity of the user. Currently, the supported activities are: *in vehicle*, *on bicycle*, *on foot*, *running*, *still*, *tilting*, *walking* and *unknown*. Accuracy is provided here, as well.

The collected sensor readings, on one hand, are uploaded to the XMPP server, where the analytics module processes and shares them among parties who are subscribers of the relevant information; on the other hand, are used locally. For example, user activity is analyzed on the server side and it is used to create non real-time stop patterns through machine

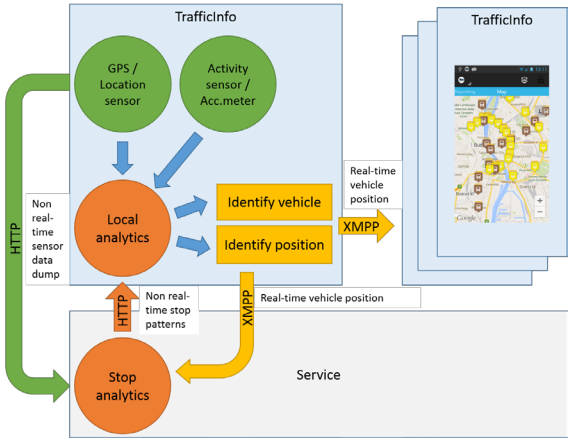


Fig. 6. Sensor data flows in TrafficInfo

learning. These patterns are delivered back to the application, where further local analytics can use them for, e.g., identifying the vehicle and providing position information. These sensor data flows are depicted in Fig. 6. Note that, at the moment, stop events are detected locally on the device due to resource usage reasons and only the detected events with a timestamp are reported back to the server. Based on this information the server side analytics estimate the upcoming arrival times of the given vehicle and disseminate live timetable updates to the subscribers.

C. Service Levels

Running TrafficInfo in a small city is different than in big cities, like Budapest. The cause of this difference is the unavailability of static public transportation information in, e.g., GTFS format. If even the static public transportation schedule is not presented by the application, people will likely not use it. Furthermore, fewer users will generate less live traffic data which makes the whole application useless. Hence, it is clear that we should apply a different approach in cities where static public transportation information has not been available, yet.

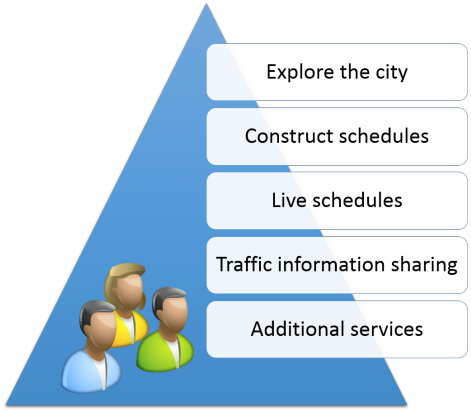


Fig. 7. Service levels vs. user base

Fig. 7 depicts the different service levels that we offer with the growing number of users. These service levels are the following.

1) City Explorer: In a new city, at the beginning we assume that there is zero knowledge in our system about the city's public transportation. The goal is to gather the relevant information in a fast and inexpensive way. When a reliable GTFS or OSM information base of the given city is available, we import this data into our databases. In other situations, we use crowdsourcing to gather this information. We assume that some users will install the TrafficInfo application either to contribute to city exploration or just simply for curiosity (or some incentive mechanism has to be introduced to grab users). We expect no other contributions than installing the application, carrying the smartphone during the day, traveling on public transportation and answering some simple questions asked by the application. The smartphones, using their built-in sensors, collect all the necessary data without user interaction. The questions are used to annotate the collected data.

Every day the captured data is uploaded in a batch to the server for analysis. At the same time, the application downloads information about what to ask on the following day(s).

Fig. 8 depicts an uploaded activity log of a particular user. In this example, the information source is the Google activity recognition module mentioned above. The blue bars show the detected activity during the capture time. In addition, another sensor module recorded the motion, too. Its output is the red bars, recognizing only still or moving states. The height of the bars expresses the confidence of the recognition. Althought the values represented with blue and red bars are coming from two different sensors, they usually have the same results for the still state. There are only a few differences, where the activity recogniton shows unknown event, while the motion sensor signals still state. It is not displayed on the figure, but the GPS position and its accuracy are also logged for every event.

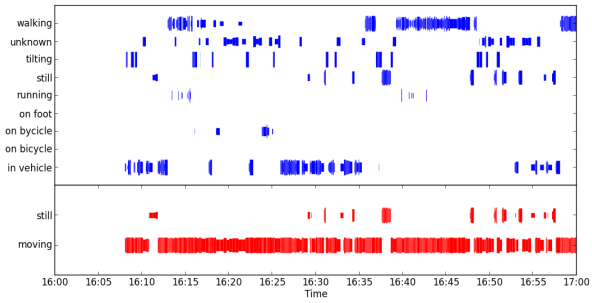


Fig. 8. Captured sensor data during user activity

The captured logs are processed by the server during the night, when the users are typically inactive and the system tries to guess the public transport stops and routes in the city. The more users report the same information the higher the chance is to guess the transportation system correctly. A database stores all the possible stop locations together with

their confidence. This database is then downloaded to the application which will ask simple questions to the users to identify stops. For instance, the application might ask: *"Are you standing at a stop, waiting for public transport?"* We expect simple answers for simple questions until we can construct the public transportation stop database. Routes are explored in a similar way. When the user travels between already known stops, we assume that there is a public transport route among these stops. The application might ask the user about the route type and the line number.

2) Schedule (Re)Construction: Once the public transportation stops and routes are explored in most parts of the city, we can assume with high confidence that more users join and use the application. Visualizing stops and routes aids users to get orientation. However, the exploration of the city is continuing, the sensor readings are always collected, but questions are asked only regarding to the partially explored areas.

When the number of users exceeds a certain level and the trips can be guessed, the automatic detection of the stop events comes into the picture. The detected events are reported to the server by the application. The server filters this data and analyzes the patterns of each transport line. As more stop events are captured the patterns are more complete and finally the public transportation schedule is constructed.

3) Live Schedule: TrafficInfo providing public transportation stops, routes and schedules is assumed to attract many users, similarly to those applications that are available in big cities based on GTFS data. One advantage of TrafficInfo is that it provides an alternative way to collect all necessary information from scratch which does not require the cooperation of the public transport operator company, rather relies on the power of the crowd.

When the number of users is high enough and (static) schedule information is available, the continuously collected position and stop event data is used to create and propagate lively updates. These updates refresh the timetable if necessary and reflect the actual public transport traffic conditions.

4) Information Sharing on Public Transport Conditions: On-line users are able to send and receive information about the vehicle's conditions they are actually riding. This requires user interaction on a voluntary basis as current sensors are not able to detect crowdedness, torn seats, bad drivers, etc. If the application has a wide user base we can always expect some volunteers to report on such conditions. The application provides easy to use forms to enter the relevant data (see Sec. IV-B2).

5) Additional Services: When TrafficInfo is running in a full-fledged manner, it can cooperate with other services targeting public transportation. For example, a rendezvous service can be paired to the TrafficInfo application to organize dates on public transportation vehicles.

D. Stop Event Detection

One of the fundamental functions of TrafficInfo is to detect stop events of public transport vehicles. We implemented such a detector locally on the mobile device. The reason behind that is twofold. First, cheaper devices produce bogus raw GPS location data that, if directly transmitted to the XMPP server, would mislead the service. Second, raw logs are generated at a very high rate and it would cause a substantial burden to transmit the raw logged data to the server in real-time for further processing. Instead, only when stop events are detected a summary of information, e.g., the timestamp of the event and the time elapsed since the last stop event, will be transmitted.

Fig. 9. GPS position trajectory (blue) and the real tram route (red) as logged by a Samsung Galaxy S3 device

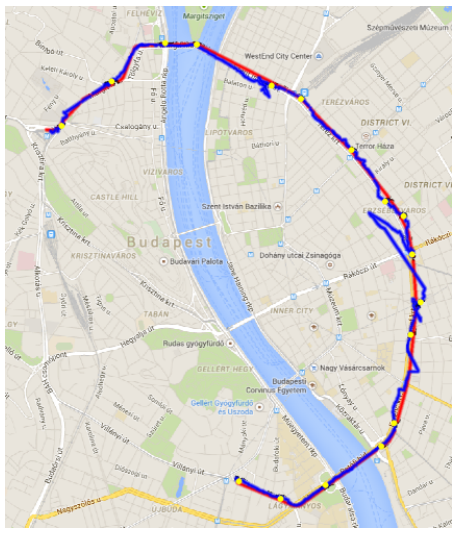


Fig. 10. GPS position trajectory (blue), the real tram route (red) and stops (yellow dots) as logged and detected by a Nexus4 device

To illustrate the challenge of stop event detection, we show the logged trajectory on tram routes 4 and 6 in Budapest from two devices, a Samsung Galaxy S3 and a Nexus4 smartphone, in Fig. 9 and Fig. 10, respectively. In case of Nexus4 (Fig. 10), yellow dots indicate the predicted locations of the stop events. Note that Nexus4 with network information provides correct position data, similar in quality to the Galaxy S3 device.

Unfortunately, we were not able to collect GPS position data from all the devices we used in our experiments even if the device was equipped with GPS sensor.

Our solution for stop event detection is based on features. Hence, we generated several features from the experimental usage logs collected during the testing period. The measurement object we used to collect context data is summarized in Table I. It includes among others GPS, WiFi, network and acceleration sensor readings, etc.

TABLE I
SEMANTICS OF TRAFFICINFO MEASUREMENT LOGS

Field Description	Examples, Possible Values
Event type	Initialization, manual, sensor
Timestamp	Time when the event occurred
Track (tram, bus line)	Tram 6
Phone type	E.g., Nexus4 including IMEI
Acceleration	Absolute or axes X, Y and Z
GSM signal strength	As defined in the relevant standard
Android GPS, network and passive location accuracy, longitude and Latitude values	Android GPS Location Provider data, accuracy radius with 68% confidence
CellID, WiFi MACID	LAC (Location Area Code) and CID (Cell ID)
Vehicle number	ID of the transport vehicle
Direction	Onward or backward
Arrived at	Time of arrival at the stop
Manual input	- Stopped at Station - Revoke Stopped at Station - Leaving Stop - Revoke Leaving Stop - Stopped at Traffic Light - Revoke Stopped at Traffic Light - Revoke Last Input

The features we defined are the following:

- Latitude, Longitude: raw GPS data;
- AccAbsMax and AccAbsMin: maximum and minimum value of acceleration in the past 20 seconds;
- Last Annotation Time: in seconds, depending on the annotation type (Stopped at Station or Leaving Stop);
- Closest Station: distance calculated from raw GPS data;
- GPS Distance: distance traveled during the last 20 seconds based on raw GPS data.

We collected Android sensor and location data by using the Android Location API[3]. The device can have multiple LocationProvider subclasses based on network, GPS and passive sensors, and the location manager has a method to select the best provider. Accessing the sensors requires three level permissions: ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION, INTERNET. The GPS sensor can be accessed by the NMEA listener[4]. Accelerometer is accessible through the Google Location Services API, part of Google Play Services, a high level framework that automates the location provider choice.

For classification we used the J48 decision tree implementation of the Weka data mining tool[5]. The final output of our detector is the detected stop event, including location and timestamp. With the combination of the defined features and models we could detect stop events with high accuracy within 13 seconds after the arrival at the station.

We measure the accuracy of the method by computing the precision, recall and AUC (Area Under the Curve) [14] of our classifiers in a 10-fold crossvalidation setting. We consider AUC as the main stable measure for classifier performance that does not depend on the decision threshold separating the predicted stop events. The best classifier reached precision 0.97, recall 0.95, F measure 0.96. The corresponding best AUC was 0.86, which means that a random time point when the tram is at a stop is predicted 86% more likely a stop than another random time point when the tram is in between two stops. In general, an AUC between 0.8–0.9 is considered in the literature to be good to excellent.

V. SUMMARY

In this paper, we shortly introduced our XMPP based communication framework that we designed to facilitate the development of crowd assisted smart city applications. Then we presented our crowdsensing based real-time public transport information service, implemented on top of our framework, and its front-end Android application, called TrafficInfo, in detail together with our stop event detector. This detector was developed to automatically detect halt events of public transport vehicles at the stops.

As future work, we plan to develop TrafficInfo further and enhance the different services of all the introduced service levels. Moreover, we intend to recruit a noticeable user base and carry out field experiments with these real users. Their feedback is important to plan the directions for improvements.

ACKNOWLEDGMENT

The publication was supported by the TÁMOP-4.2.2.C-11/1/KONV-2012-0001 project. Károly Farkas has been partially supported by the Hungarian Academy of Sciences through the Bolyai János Research Fellowship.

REFERENCES

[1] R. Ganti, F. Ye, and H. Lei, "Mobile Crowdsensing: Current State and Future Challenges," *IEEE Communications Magazine*, pp. 32–39, Nov. 2011.

[2] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core," RFC 6120 (Proposed Standard), Internet Engineering Task Force, Mar. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6120.txt

[3] R. L. Szabo and K. Farkas, "A Publish-Subscribe Scheme Based Open Architecture for Crowd-sourcing," in *Proceedings of 19th EUNICE Workshop on Advances in Communication Networking (EUNICE 2013)*. Springer, Aug. 2013, pp. 1–5.

[4] Google Inc., "General Transit Feed Specification Reference." [Online]. Available: https://developers.google.com/transit/gtfs/reference/

[5] P. Zhou, Y. Zheng, and M. Li, "How Long to Wait?: Predicting Bus Arrival Time with Mobile Phone based Participatory Sensing," in *Proceedings of the Tenth International Conference on Mobile Systems, Applications, and Services (MobiSys 2012)*, Jun. 2012.

[6] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson, "Cooperative Transit Tracking Using Smart-phones," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys 2010)*, Nov. 2010, pp. 85–98.

[3] http://developer.android.com/guide/topics/sensors/index.html

[4] https://developer.android.com/reference/android/location/GpsStatus.NmeaListener.html

[5] http://www.cs.waikato.ac.nz/ml/weka/

[7] L. Bedogni, M. Di Felice, and L. Bononi, "By Train or by Car? Detecting the User's Motion Type Through Smartphone Sensors Data," in *Proceedings of IFIP Wireless Days Conference (WD 2012)*, 2012, pp. 1–6.
[8] J. Biagioni, T. Gerlich, T. Merrifield, and J. Eriksson, "EasyTracker: Automatic Transit Tracking, Mapping, and Arrival Time Prediction Using Smartphones," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys 2011)*, Nov. 2011, pp. 1–14.
[9] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)," W3C, W3C Recommendation REC-xml-20081126, Nov. 2008. [Online]. Available: http://www.w3.org/TR/2008/REC-xml-20081126/
[10] P. Saint-Andre, "XEP-0045: Multi-User Chat," XMPP Standards Foundation, Standards Track XEP-0045, Feb. 2012. [Online]. Available: http://xmpp.org/extensions/xep-0045.html
[11] P. Millard, P. Saint-Andre, and R. Meijer, "XEP-0060: Publish-Subscribe," XMPP Standards Foundation, Draft Standard XEP-0060, Jul. 2010. [Online]. Available: http://xmpp.org/extensions/xep-0060.html
[12] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, Jun. 2003.
[13] M. M. Haklay and P. Weber, "OpenStreetMap: User-Generated Street Maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, Oct. 2008. [Online]. Available: http://dx.doi.org/10.1109/MPRV.2008.80
[14] J. Fogarty, R. S. Baker, and S. E. Hudson, "Case Studies in the Use of ROC Curve Analysis for Sensor-based Estimates in Human Computer Interaction," in *Proceedings of Graphics Interface 2005*, ser. GI '05. School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society, 2005, pp. 129–136. [Online]. Available: http://portal.acm.org/citation.cfm?id=1089508.1089530

Károly Farkas received his Ph.D. degree in Computer Science in 2007 from ETH Zurich, Switzerland, and his M.Sc. degree in Computer Science in 1998 from the Budapest University of Technology and Economics (BME), Hungary. Currently he is working as an associate professor at BME. His research interests cover the field of communication networks, especially autonomic, self-organized, wireless and mobile ad hoc networks, and mobile crowdsourcing. He has published more than 70 scientific papers in different journals, conferences and workshops and he has given a plenty of regular and invited talks. In the years past, he supervised a number of student theses and coordinated or participated in several national and international research projects, such as CityCrowdSource of EIT ICTLabs. Moreover, he acted as program committee member, reviewer and organizer of numerous scientific conferences, thus he took the general co-chair role of the IEEE PerCom 2014 conference and the TPC co-chair role of the CROWDSENSING 2014 and the CASPer 2015 workshops. He is the coordinator of the local Cisco Networking Academy and was the founding initiator of the Cisco IPv6 Training Laboratory and the BME NetSkills Challenge student competition at BME. Between 2012 - 2015 Dr. Farkas has been awarded the Bolyai János Research Fellowship of the Hungarian Academy of Sciences.

Gábor Fehér graduated in 1998 at the Budapest University of Technology and Economics on the Faculty of Electronic Engineering and Informatics. In 2004 he received a PhD. degree, the topic of his thesis was resource control in IP networks. Currently he is an associate professor at the same university. Besides giving lectures, he is also contributing to various national and international research projects. From 2004 he is continuously involved in three consecutive EU founded IST/ICT projects. He has teaching activity on the faculty's Smart City specialization working with microelectronics, sensor networks and smartphones. He and his students are working on more projects with crowdsouring and crowdsensing, from the basic research up to the prototype applications.

András Benczúr received his Ph.D. at the Massachusetts Institute of Technology in 1997. Since then his interest turned to Data Science. He is the head of 30 doctoral students, post-docs and developers at the Institute for Computer Science and Control of the Hungarian Academy of Sciences (SZTAKI). He is site coordinator in the Hungarian Future-ICT project, and cloud computing activity leader in the Budapest node of EIT ICTLabs. He serves on the program committees of leading conferences including WWW, WSDM, ECML/PKDD, he was Workshop Chair for WWW 2009 and main organizer of the ECML/PKDD Discovery Challenge 2010. In 2012 he was awarded the Momentum grant of the Hungarian Academy of Sciences for his research in Big Data.

Csaba Sidló started working on data warehousing projects and application driven research problems of extremely large data sets in 2000. He joined the Institute for Computer Science and Control of the Hungarian Academy of Sciences (SZTAKI) in 2004; he is now head of the Big Data Business Intelligence research group. His main interest is Big Data analytics and business intelligence on scalable distributed architectures. His industrial projects include master data entity resolution, integration and analytics of log, web and location data. He is currently involved in several big data projects for web, telecom and sensor data analytics. Csaba authored several research papers and book chapters, and has a PhD in Informatics from Eötvös University, Hungary.