CHAPTER 25

Smart Timetable Service Based on Crowdsensed Data

Károly Farkas
Budapest University of Technology and Economics, Hungary,
farkask@hit.bme.hu

Abstract

Rapid technological development and the introduction of smart services make it possible for modern cities to offer an enhanced perception of city life for their inhabitants. For instance, a smart timetable service of the city's public transportation lines updated in real-time can decrease unnecessary waiting times at stops and increase the efficiency of travel planning. However, the implementation of such a service in a traditional way requires the deployment and maintenance of some costly sensing and tracking infrastructure. Fortunately, for this purpose mobile crowdsensing can be a viable and almost free of charge alternative. In this case, the crowd of passengers and their mobile devices are used to gather data.

In this chapter, we place emphasis on the introduction of a crowdsensing based smart timetable service, which has been developed 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 public transport information of the given city on Google Maps in real-time. The live updates of transport schedule information rely on the automatic stop event detection of public transport vehicles. TrafficInfo is built upon an Extensible Messaging and Presence Protocol (XMPP) based communication framework which was designed to facilitate the development of crowd assisted smart city applications.

How to cite this book chapter:
Farkas, K. 2016. Smart Timetable Service Based on Crowdsensed Data. In: Capineri, C, Haklay, M, Huang, H, Antoniou, V, Kettunen, J, Ostermann, F and Purves, R. (eds.) *European Handbook of Crowdsourced Geographic Information*, Pp. 339–351. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/bax.y.

The chapter introduces this generic framework shortly, then describes the prototype smart timetable service.

Keywords

Smart cities, Crowdsensing, Public transportation, XMPP, GTFS

Introduction

More and more modern cities offer smart services, which are services using modern infrastructure and/or providing value added functions to ease the everyday life of inhabitants. Unfortunately, the traditional way of introducing a new service usually implies a huge investment to deploy and maintain 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 that extends the basic service functions can be expensive. For instance, let us consider the replacement of static timetables with a live public transport information service updated in real-time. It requires the deployment of a vehicle-tracking infrastructure consisting of among others GPS sensors, communication infrastructure, back-end systems and front-end user interfaces, which can be a cost intensive investment.

An alternative approach to collect real-time tracking data is exploiting the power of the crowd via participatory sensing or often called mobile crowdsensing, which does not call for such an investment. In this scenario (see Figure 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. Moreover, they 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 real-time 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 at the stop can report the line number with a timestamp of every arriving public transport vehicle 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.

In this chapter, we focus on the introduction of a crowdsensing based smart timetable service, which has been developed as a prototype smart city application. The front-end interface of this service, called TrafficInfo, is a simple and easy-to-use Android application. It visualizes live public transport information of the given city on Google Maps. TrafficInfo is built upon an Extensible

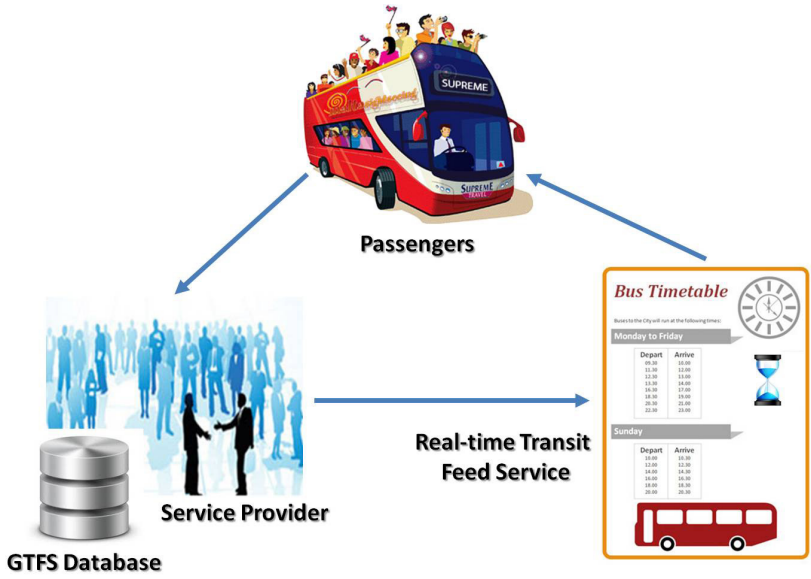


Figure 1: Live public transport information service based on mobile crowdsensing.

Messaging and Presence Protocol (XMPP) (Saint-Andre 2011) based communication framework (Szabo & Farkas 2013). This framework was designed to facilitate the development of crowd assisted smart city applications (and will be introduced shortly in the upcoming section). Following the publish/subscribe (pub/sub) communication model the passengers subscribe in TrafficInfo to traffic information channels according to their interest. These channels are dedicated to different public transport lines or stops. Hence, the passengers are informed about the live public transport situation. For instance, they can see the actual vehicle positions, deviation from the static timetable, crowdedness information, travel conditions, etc.

To motivate user participation in data collection an initial service is offered to the passengers, which is a static public transportation timetable. It is built on the General Transit Feed Specification (GTFS) (Google Inc. 2006) based transit schedule data and provided by public transport operators. GTFS is the best practice for providing such information, and is available in 350 cities attracting more than 6.5 million users. According to the GTFS developer page, currently GTFS data is available for 879 transit agencies worldwide. TrafficInfo basically presents this static timetable information to the users which is then updated in real-time, if appropriate crowdsensed data is available. To this end, the application collects the following information: position data; the timestamped halt events, detected automatically, of the public transport vehicles at the stops; simple annotation data entered by the user, such as reports on crowdedness and

travel conditions. After analyzing the data gathered live updates are generated and TrafficInfo refreshes the static information with these updates.

The rest of this chapter is structured as follows. A quick overview of crowd assisted transit-tracking systems is provided in the next section. Then, our generic framework to facilitate the development of crowdsensing based services is introduced shortly. Next, we describe the prototype smart timetable service. Finally, we give a short summary.

Crowd Assisted Transit-tracking Systems and Approaches

This section gives an overview of crowd assisted transit-tracking solutions.

Moovit (Moovit Developers 2014) is meant to be a live transit app on the market providing real-time information about public transportation. 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 an initial service, too. The source of this information is the company who operates the public transportation network. Moovit partially relies on GTFS.

Several other mobile crowdsensing based transit-tracking ideas have been published recently. For instance, Zhou, Zheng and Li (2012) 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. Thiagarajan et al. (2010) propose a method for transit tracking using the collected data of the accelerometer and the GPS sensor on the users' smartphone. Bedogni, Di Felice and Bononi (2012) use smartphone sensors data and machine learning techniques to detect motion type, e.g. traveling by train or by car. EasyTracker (Biagioni et al. 2011) 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 to offer enriched services to the users. The focus of our work, in turn, is on how to introduce such enriched services incrementally. Namely, how one can 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. Hence, our work, in essence, complements the above ones.

Generic Framework for Crowdsensing Based Smart City Applications

In this section, our generic framework (Szabo & Farkas 2013) to aid the development of crowdsensing based smart city applications is described shortly. This

framework is based on the XMPP publish/subscribe architecture. TrafficInfo is implemented on top of this framework.

Communication Model

XMPP (Saint-Andre 2011) is an open technology for real-time communication using Extensible Markup Language (XML) 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 or the notification service. The latter realizes a publish/subscribe (pub/sub) communication model, 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 publish/subscribe communication model in our generic framework to implement interactions. In this model, we defined three roles, like *Producer*, *Consumer* and *Service Provider* (see Figure 2). These entities interact with each other via the core service, which consists of event based pub/sub nodes.

Producer: The Producer acts as the original information source in the 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. The user is called as *Prosumer*, when he acts in the service as both Consumer and Producer at the same time.

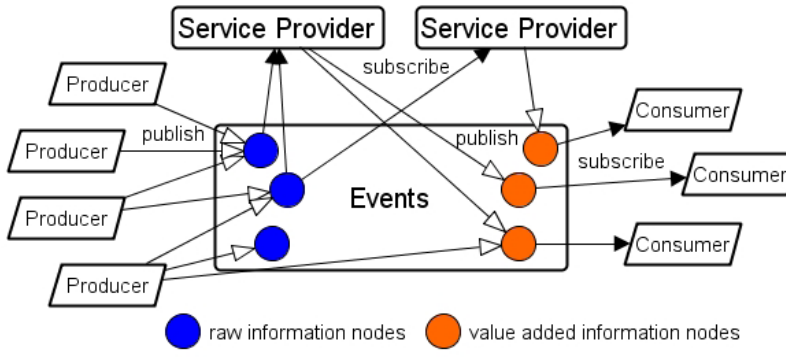


Figure 2: Crowdsensing model based on publish/subscribe communication.

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. Moreover, multiple Service Providers can act concurrently and offer different value added services to different Consumers.

In the model, depicted in Figure 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.

Framework Architecture

This model can be directly mapped to the XMPP publish/subscribe model as follows (see Figure 3):

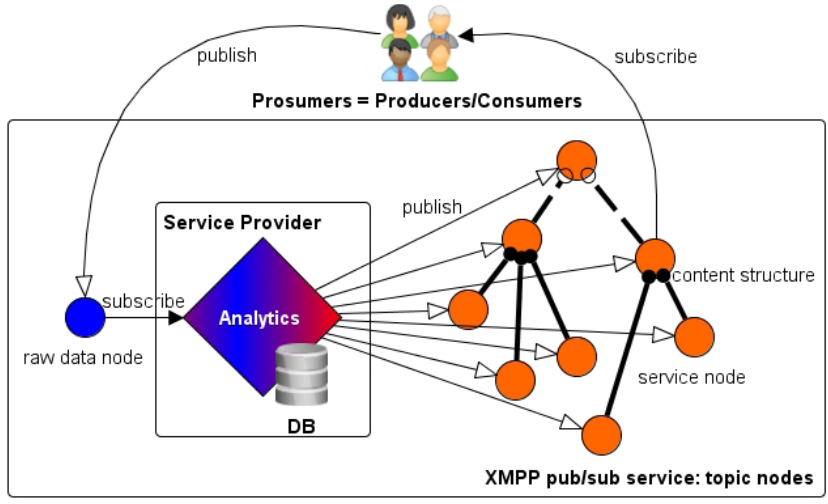


Figure 3: Mobile crowdsensing – the publish/subscribe value chain using XMPP.

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. Figure 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.

Smart Timetable Service

In this section, the architecture of the prototype smart timetable service is delineated first, then TrafficInfo, its front-end Android interface together with the developed automatic stop event detector is described.

Service Architecture

The prototype smart timetable service architecture has two main building blocks, such as the generic crowdsensing framework described in the previous section and the front-end application called TrafficInfo (see Figure 4). The framework can be divided into two parts, a standard XMPP server and a GTFS Emulator with an Analytics module.

XMPP Server

The XMPP server maps the public transport lines, stored in GTFS (Google Inc. 2006) format, to a hierarchical pub/sub channel structure. Thus, the GTFS

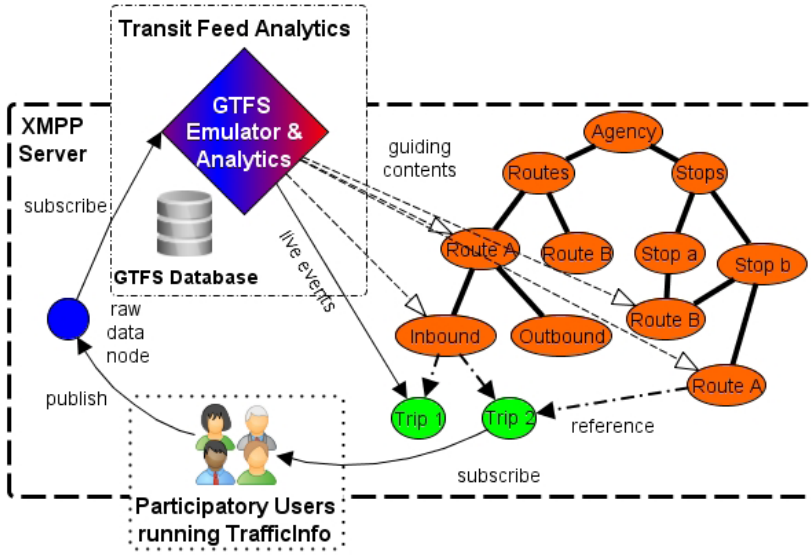


Figure 4: Smart timetable service architecture.

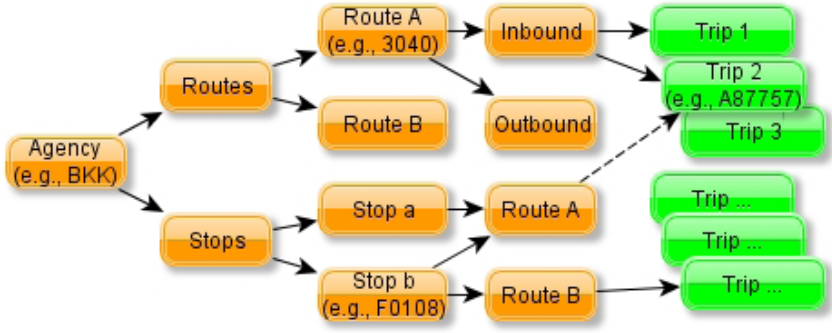


Figure 5: Publish/subscribe model for GTFS feeds.

database is turned into an XMPP pub/sub node hierarchy. This node structure facilitates searching and selecting transit feeds according to user interest. The pub/sub node model for content filtering in a transport information feed is depicted in Figure 5.

The root of the pub/sub tree is the *Agency* node referring to the public transport operator. 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.

For instance, the leaf node with trip ID 'BKK-Routes-3040-Inbound-A87757' (cf. the bracketed labels in the nodes of Figure 5) handles transit feed and its real-time updates. It is related to *Trip 2* in the *inbound* direction and belongs to *Route A* of *Agency BKK* (operator at Budapest, Hungary). On the other hand, node 'BKK-Routes-3040' stores persistent transit information with regard to *Route A* (e.g. route name, short name, stops, head-signs). References to all the currently active *inbound* trips are found in node 'BKK-Routes-3040-Inbound'. Similarly, node 'BKK-Stops-F0108' stores persistent data with regard to the given stop (e.g. stop name, GPS coordinates) and lists the routes this stop is part of. Furthermore, the *trip ID* of every active trip is listed in the route node.

GTFS Emulator, Analytics Module

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) (Haklay & Weber 2008), a crowdsourcing based mapping service. 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 resulted data set reflects more accurately the actual situation in the given territory because the OSM data is updated more frequently than the GTFS data set.

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.

Front-end Application

The front-end application, called 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.

TrafficInfo

TrafficInfo has four main functions, such as visualization, information sharing, sensing and stop event detection. These functions are discussed below.

Visualization

Most of the users benefit from the visualization capability of TrafficInfo that visualizes public transport vehicle movements on a city map. An example of

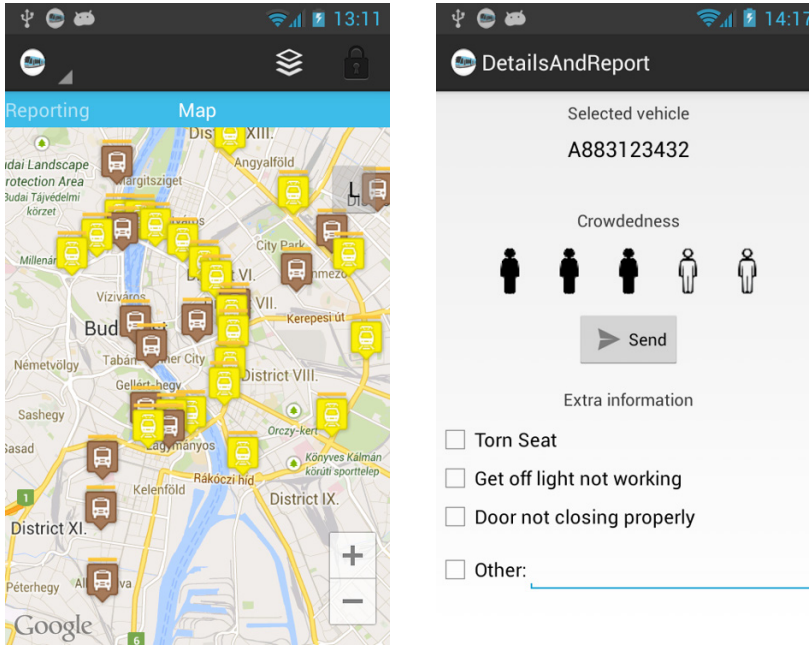
this primary function can be seen on Figure 6a displaying trams of line 1, 4, 6 and buses of line 7 and 86 on the Budapest map in Hungary. The depicted vehicles can be filtered to given routes. The icon of a vehicle reflects various attributes, such as the number, progress or crowdedness of the specific vehicle. Clicking on a vehicle's icon a popup shows all known information about that specific vehicle.

Information Sharing

The second function serves for information sharing. Passengers can share their observations regarding the vehicles they are currently riding. Figure 6b shows the feedback screen that is used to submit reports. The feedback information is spread out using the 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.

Sensing

The third function is collecting smartphone sensor readings without user interaction, which is almost invisible for the user. It is done automatically in the



a) Vehicle visualization (b) User feedback form

Figure 6: TrafficInfo screenshots.

background, the only thing the user has to do is to start TrafficInfo. 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, the movement of the user is identified through his activities. To this end various sensors are used, e.g. accelerometer, and the timestamped stop events of the vehicles are deducted. The duration between the detected stops coupled with GPS coordinates identifies the route segment, which the user actually rides. With regard to the energy consumption of the sensor readings we carried out some measurements. Our results showed that in case of a normal daily scenario, such as traveling approx. 1 hour to the workplace in the morning and 1 hour back home in the afternoon, the readings and local processing consume 1.48 Wh energy on average. This is equivalent roughly to 20% of the capacity of an average smartphone battery (2,000 mAh, 3.7 V).

Besides the GPS coordinates Google also provides location information in those areas, where there is no GPS signal available. Usually this position is highly inaccurate, but the estimated accuracy is also provided. Moreover, the activity sensor, which guesses the actual activity of the user, is also used by TrafficInfo. Currently, the supported activities are: *in vehicle*, *on bicycle*, *on foot*, *running*, *still*, *tilting*, *walking* and *unknown*. The sensor monitoring part of TrafficInfo is active only if the activity recognition reports *in vehicle* status. Otherwise, sensor monitoring is suspended. Note, that the accuracy of activity recognition can be varying. However, in our experiments the *in vehicle* activity was recognized with more than 85% accuracy, which made this tool useful for our purposes.

The collected sensor readings on one hand are uploaded to the XMPP server. There the Analytics module processes and shares them among participants who are subscribers of the relevant information. On the other hand, they are used locally. For instance, based on the timestamps of the detected stop events the server side analytics estimate the upcoming arrival times of the given vehicle and disseminate live timetable updates to the subscribers.

Stop Event Detection

The fourth, most challenging function of TrafficInfo is to detect stop events of public transport vehicles without user interaction. TrafficInfo implements such a detector locally on the mobile device. When stop events are detected a summary of information is transmitted to the XMPP server. This summary consists of the location and timestamp of the event and the time elapsed since the last stop event. The final decision is made by the server based on the periodic reports from the passengers. It is a majority decision, so if the majority of the reports indicate a stop event within a given time window the detection is made otherwise not.

The stop event detection mechanism is based on features. Hence, several features were generated from the experimental usage logs collected during the

measurements. In this work, we investigated our detection mechanism only on trams and left buses/trains as part of future work. The approximately 1GB measurement data were collected by 10 volunteers during the 1 month measurement period using Samsung Galaxy S3 and Nexus4 smartphones. The gathered context data included among others GPS, Wi-Fi, cellular network and acceleration sensor readings. For classification the J48 decision tree implementation of the Weka data-mining tool (Hall et al. 2009) was used. With the combination of the defined features and models the detector can detect stop events automatically with relatively high accuracy (with 0.86 AUC – Area Under the Curve) within 13 seconds after the arrival at the station. The place of the stop event is decided by investigating the GPS position and/or the Wi-Fi/cellular network fingerprint of the environment, so stops at stations can be distinguished from other stops with high probability.

Summary

In this chapter, after a short literature review a generic, XMPP based communication framework was introduced which was designed to facilitate the development of crowd assisted smart city applications. Then a prototype crowdsensing based smart timetable service was presented. Its front-end Android application, called TrafficInfo, together with an automatic stop event detector was introduced in detail. This service was implemented on top of the introduced generic framework. It updates static public transport timetables and delivers the updated information to its subscribers in real-time.

Acknowledgement

This work was partially supported by the TÁMOP-4.2.2.C-11/1/KONV-2012-0001 and the EITKIC 12-1-2012-0001 projects. The author would like to acknowledge the support and help of the participants of these projects, especially the contribution of Timon Tomás, Ádám Zsolt Nagy, Róbert Szabó, Imre Lendák, Bernát Wiandt, András Benczúr, Csaba Sidló and Gábor Fehér. Károly Farkas has been partially supported by the Hungarian Academy of Sciences through the Bolyai János Research Fellowship.

References

Bedogni, L., Di Felice, M., & Bononi, L. 2012. By Train or by Car? Detecting the User's Motion Type Through Smartphone Sensors Data. In: *Proceedings of IFIP Wireless Days Conference (WD 2012)*. Dublin, Ireland on 21–23 November 2012, pp. 1–6.

Biagioni, J., Gerlich, T., Merrifield, T., & Eriksson, J. 2011. 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)*. Seattle, WA, USA on 1–4 November 2011, pp. 1–14.

Google Inc. 2006. *General Transit Feed Specification Reference*, 25 September 2006. Available at: https://developers.google.com/transit/gtfs/reference/ (Last accessed 25 June 2015).

Haklay, M. M., & Weber, P. 2008. OpenStreetMap: User-Generated Street Maps. *IEEE Pervasive Computing, 7*(4): 12–18. DOI: http://dx.doi.org/10.1109/MPRV.2008.80

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, H. I. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explorations, 11*(1): 10–18. Available at: http://www.kdd.org/sites/default/files/issues/11-1-2009-07/p2V11n1.pdf.

Moovit Developers. 2014. Moovit. Available at: http://www.moovitapp.com/ (Last accessed 25 June 2015).

Saint-Andre, P. 2011. Extensible Messaging and Presence Protocol (XMPP): Core, RFC 6120 (Proposed Standard), Internet Engineering Task Force, March 2011. Available at: http://www.ietf.org/rfc/rfc6120.txt (Last accessed 25 June 2015).

Szabo, R. L., & Farkas, K. 2013. A Publish-Subscribe Scheme Based Open Architecture for Crowd-sourcing. In *Proceedings of 19th EUNICE Workshop on Advances in Communication Networking (EUNICE 2013)*. Chemnitz, Germany on 28–30 August 2013, pp. 1–5.

Thiagarajan, A., Biagioni, J., Gerlich, T., & Eriksson, J. 2010. Cooperative Transit Tracking Using Smart-phones. In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys 2010)*. Zurich, Switzerland on 3–5 November 2010, pp. 85–98.

Zhou, P., Zheng, Y., & Li, M. 2012. How Long to Wait?: Predicting Bus Arrival Time with Mobile Phone based Participatory Sensing. In: *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys 2012)*. Low Wood Bay, Lake District, UK on 25–29 June 2012.