Network Proximity and Physical Web

Yousef Ibrahim Daradkeh
College of Engineering at Wadi Aldawaser
Prince Sattam bin Abdulaziz University, Saudi Arabia
daradkehy@yahoo.ca

Dmitry Namiot
Faculty of Computational Mathematics and Cybernetics
Lomonosov Moscow State University
Moscow, Russia
dnamiot@gmail.com

***Abstract*—** The Physical Web is a tool (an approach) to connect any physical object to the web. The Physical Web lets "render" physical objects in web, usually, with the help of mobile devices. This approach lets us navigate and control physical objects in the world surrounding mobile devices. There are different ways to enumerate physical objects. In this paper, we will describe the model based on the network proximity. In this model, the circulated information depends on the proximity to the network nodes in the wireless networks. We will discuss the possibility to use network proximity for interactions with physical objects.

***Keywords*—**Physical Web; network proximity; Bluetooth; Wi-Fi

I. INTRODUCTION

The Physical Web is a term that describes the process of presenting everyday objects in Internet. It aims to offer users the way to manage their daily tasks at using everyday objects that are soon to become smart and remotely controllable. This approach lets us navigate and control physical objects in the world surrounding mobile devices. Also, it helps perform everyday tasks depending on the surrounding physical objects. Of course, one of the first questions on this path is the way to enumerate physical objects.

One of the most often used approaches for physical objects markup is the deployment of wireless tags. Wireless tags can support standard protocols like Bluetooth and Wi-Fi. So, for mobile devices (mobile users) the detection of tags is actually the detection of wireless nodes. Note, that in this approach other mobile devices can play a role of tag too. And the network proximity here describes data models based on the detection of surrounding network nodes.

In this paper, we would like to discuss several approaches for building mobile systems based on the detection of physical objects via network proximity. Note, that the classical models for interaction with physical objects are a subject of Internet of Things (Web of Things) [1]. In this paper, we will mostly discuss the services which could be initiated by the presence of surrounding physical object. Such services do not always incur two-way data exchange with the physical objects. In the most cases, it is enough to detect and identify the object.

The proximity is a very conventional way for context-aware programming in mobile world. There are many practical use cases, where the concept of the location can be replaced by that of proximity. Proximity can be used as a main formation for context-aware browsers [2]. The context-aware browser will reveal data chunks depending on the current context.

The usage of network proximity for context-aware systems is very transparent. At his moment, network modules are most widely used "sensors" for mobile phones. All smartphones nowadays have Wi-Fi (Bluetooth) modules. So, Wi-Fi (Bluetooth) related measurements are included into standard interfaces of mobile operating systems. The above-mentioned measurements include the visibility for network nodes and signal strength. By the definition, the distribution for Bluetooth signal, for example, is limited. So, if any Bluetooth node is visible from a mobile device (a mobile phone, for example), then this device is somewhere nearby that node (it is so-called Bluetooth distance). The same is true for Wi-Fi access point. And this proximity information (network proximity) can replace location data. There are two main reasons for this replacement. At the first hand, we can target here all indoor application [3]. Obtaining GPS (Global Positioning System) data indoor is not reliable and sometimes even impossible. In the same time, modern offices usually have plenty of wireless nodes. The second reason is much more interesting. The wireless node could be moveable. So, our context information will follow to the moved object.

For network proximity-based context-aware applications, any existing or even especially created Bluetooth node could be used as a presence-sensor that can play the role of a trigger. This trigger can open access to some content, discover existing content, as well as cluster nearby mobile users [4,5].

The rest of the paper is organized as follows. In Section II, we discuss iBeacons. In Section III, we discuss Google Physical Web. In Section IV, we describe Bluetooth Data Points.

II. IBEACONS

The iBeacon is a wireless tag (beacon), based on Bluetooth Low Energy (BLE) standard [6]. Shortly, any beacon is set to transmit a set of numbers several times per minute, so that any mobile device with BLE support nearby can detect it. The beacon's repetitive transmission is called also as "advertising". The BLE standard specifies a structure for the

data that must be transmitted. An application on a mobile device can then detect this parcel of information, unpack it, and use it for providing context-aware services.

The above-mentioned advertising includes a unique ID for a tag and two application-dependent numbers (so-called minor and major).

As per Apple's manual, a proximity universally unique identifier (UUID) is 16 Bytes, and major and minor codes are 2 Bytes each. The common usage for UUID is the identification for a place. For example, it could be a particular shop, café, etc. Major and minor codes could be used to a description of an area within a physical space associated with the above-mentioned UUID. For example, a retailer might use the major and minor code to identify, respectively, a given retail store and a specific shelf, where a beacon will be placed.

On iOS, a given application can scan for up to 20 tags (proximity UUIDs). It is, probably, one of the biggest limitations for iBeacons technology. The mobile application should statically declare UUIDs for the tags in questions [7]. For a mobile application, this declaration lets register to be notified if a Beacon with a given UUID comes within range (or goes out of range) of the device [8]. From the notification, a mobile application can obtain minor and major codes and they can then be used to uniquely identify a given beacon.

The application can then use this data, often (almost always) in tandem with a cloud service, to decide what action to take, if any, when the beacon is detected.

Beacons could be placed anywhere where potential users might wish to either trigger some form of action in a mobile application, or have that application log the fact that it came near to the beacon. For example, commuters in London are to be targeted with branded messages directly to their smartphones, as 500 buses in the capital are equipped with Bluetooth iBeacon technology [9].

There are legal and technical problems behind iBeacons. The legal problems are associated with the company Apple, who owns this technology. The main technological problem is the need for the static description of observer tags. Of course, the underlying system (iOS) can read data from all tags in the proximity, but dispatches only some of them to an application. It means that the only one company (Apple) has the whole picture.

Google comes with the own protocol for BLE [10]. Eddystone is the protocol specification that defines a Bluetooth low energy (BLE) message format for proximity beacon messages. It describes several different frame types that may be used individually or in combinations to create beacons that can be used for a variety of applications. At this moment, we can see the following frames (types of data) in the protocol:

Eddystone-UID: an opaque, unique 16-byte Beacon ID composed of a 10-byte namespace ID and a 6-byte instance ID. The Beacon ID may be useful in mapping a device to a record in external storage. The namespace ID may be used to group a particular set of beacons, while the instance ID identifies individual devices in the group. It is an analog for a minor/major pair in iBeacon from Apple.

Developers typically can use the namespace ID to signify own company or organization, so they know the owner for a beacon.

You can generate a namespace identifier with a UUID generator. But because standard UUIDs are 16-byte identifiers and namespace identifiers are only 10 bytes, we can simply drop the middle six bytes from the UUID. Google also prescribes a second technique a one-way hashing algorithm for generating a UID out of a URL. So you can algorithmically convert a domain name into a unique namespace ID. The instance identifier is meant to uniquely identify a specific beacon. Because the field is 6 bytes long, there are 2^{48} = 281 trillion combinations.

Eddystone-URL: a URL in a compressed encoding format. Once decoded, the URL can be used by any client with access to the Internet. It is a link to the Google Physical web, we will discuss below.

Eddystone-TLM frame broadcasts telemetry information about the beacon itself such as battery voltage, device temperature, and counts of broadcast packets. It contains the packet version (always a one-byte value of 0 for now), the beacon temperature (2 bytes), the beacon battery level (2 bytes), the number of seconds the beacon has been powered (2 bytes) and the number of "PDU" packet transmissions the beacon has sent (2 bytes.)

Actually, the general idea (pattern) is the same as with the "classical" iBeacons. Tags broadcast some ID, an application uses ID for getting data from the cloud. URL here is just a special case for ID. We can simulate URL transmission just by mapping tag's ID to some URL in the cloud-based datastore. Anyway, with obtained URL application should get access to the Internet for obtaining data.

Google provide Proximity Beacon API for setting attachment (data associated with) for BLE tags [11]. This API supports the following actions:

- Registering tags
- Publishing attachments to tags (associate data with tags)
- Retrieving attachments (data from tags)
- Monitoring beacons

Registered tag has got the following attributes:

- Advertised ID (required).
- Status.
- Expected stability.
- Latitude and longitude coordinates.
- Indoor floor level.
- Google Places API Place ID.
- Text description.

ISBN: 978-1-61804-327-6

Attachment is a string up to 1024 bytes long. It could be a plain string, JSON data or even encoded binary data. Attachments are stored in Google's scalable cloud.

There is also a very important remark for the development: on Android platform it is possible to obtain information about all "visible" tags.

Eddystone is a part of Nearby API [12]. Nearby uses a combination of Bluetooth, Wi-Fi, and inaudible sound (using the device's speaker and microphone) to establish proximity (Figure 1). Its implementation has been just announced

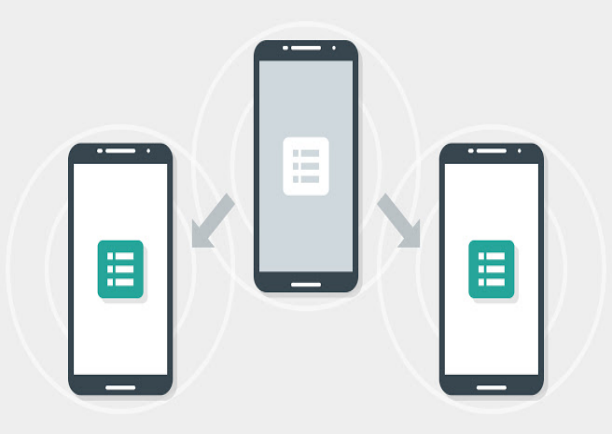

Fig. 1. Nearby API

III. Google Physical Web

Google Physical Web project is an example of integration Web technologies and physical world. Actually, it is part of a more generic problem: how to integrate Internet of Things and web technologies [13]. At its base, the Physical Web is a discovery service: a smart object broadcasts relevant URLs that any nearby device can receive. This simple capability can unlock exciting new ways to interact with the Web [14].

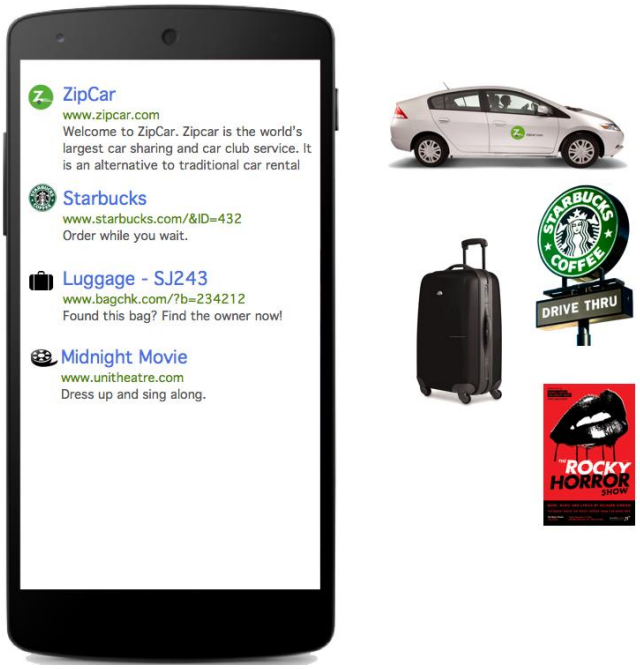

Fig. 2 The physical web

Figure 2 illustrates the basic idea. Actually, the physical objects here are (in the most cases) the same Bluetooth tags. In the current implementation, URL broadcast method involves a Bluetooth broadcast from each tag. The user's phone obtains this URL without connecting to the beacon. As per Google, this ensures the user is invisible to all beacons, meaning a user can't be tracked simply by walking past a broadcasting beacon. It is a very important principle, we would like to discuss separately. As per Google, this was very much by design to keep user's silent passage untrackable. But it is assumes also, that URL detection should be performed automatically, on the background. Again, as per Google's manual, once the user does click on a URL, they are then known to that website. With this solution, Google mostly follows to iBeacon usage (deployment model). Application on the mobile device automatically discovers nearby objects, obtains associated data (URLs in this case) and pushes this information to the user. It is a true push, iBeacons are using push notifications, supported by mobile OS [15]. Notification service is a popular functionality provided by almost all modern mobile OS (iOS, Android, etc). To facilitate customization for developers, mobile platforms support highly customizable notifications. And yes, the third-party push notification customization may allow an installed trojan application to launch phishing attacks or anonymously post spam notifications [16]. So, why do not switch to browsing mode instead of push notifications? Mobile applications may still obtain iBeacons data automatically, but show them only when a user directly requests them. It should like browsing. We must see the direct intention from mobile users to obtain nearby data. In this case, our application should form dynamically a web page (like CGI-script in the web) and show it to the user.

Technically, the Physical Web can use not only BLE as a transport layer. We can mention in this context UPnP

ISBN: 978-1-61804-327-6

technology, for example [17]. The UPnP (Universal Plug and Play) architecture offers pervasive peer-to-peer network connectivity of PCs, intelligent appliances, and wireless devices. The UPnP architecture is a distributed, open networking architecture that leverages TCP/IP and the Web to enable seamless proximity networking in addition to control and data transfer among networked devices [18]. The UPnP architecture defines a base set of standards and conventions for describing devices and the services they provide. It is designed to bring easy-to-use, flexible, standards-based connectivity to ad-hoc or unmanaged networks. In case of the Physical Web, the provided service is just an URL, associated with discovered device.

As the next possible solution, we can mention mDNS [19]. mDNS - multicast DNS service discovery, also known as zero configuration. It is an interface being used to announce and query services on the local network. Using mDNS allows a client to advertise the services of a given host without the direct help of a centralized DNS server. Again, the service here is just an URL.

We can mention here a very simple approach for creating the Physical Web for any Bluetooth/Wi-Fi device. We can define a SSID (name) for Wi-Fi access point (Bluetooth node) as some URL. SSID for Wi-Fi access point (Bluetooth node in the discoverable mode) is broadcasted (being advertised in terms of the Physical Web). Of course, this setup could be done programmatically. And mobile application (programmatically also) can get a list of available Wi-Fi access points (Bluetooth nodes in the so-called discoverable node). This list includes SSIDs (URLs). So, it is a typical Physical Web, even without BLE. This approach will work even without the dedicated tags. Wi-Wi access point (Bluetooth node in the discoverable mode) could be set programmatically (it is true for Android) right on the mobile phone. So, any smart phone could be turned into a Physical Web tag and provide advertising for some URL.

In the more generic form, this approach could be described as Beacon stuffing [20]. It is a low bandwidth communication protocol for IEEE 802.11 networks that enables Wi-fi access points to communicate with clients without association. This enables clients to receive information from nearby access points even when they are disconnected, or when connected to another access point. Originally, this scheme was developed for Wi-Fi as complementary to the 802.11 protocol. It works by overloading 802.11 management frames while not breaking the standard. The beacon-stuffing protocol is based on two key observations. First, clients receive beacons from access points even when they are not associated to them. Second, it is possible to overload fields in the beacon and other management frames to embed data. Access point embeds content in Beacon and Probe Response frames, while clients overload Probe Requests to send data. By the similar manner, this scheme will work for Bluetooth [21]. And of course, for the Physical Web we do not need the two-way communication.

Actually, the beacon-stuffing was the inspiration point for Bluetooth Data Points.

IV. Bluetooth Data Points

Bluetooth Data Points (BDP) [22] let us turn any Bluetooth node into tag. The main idea behind BDP is to associate some user-defined data with existing (or even especially created) wireless networks nodes. Originally, the project targets Bluetooth nodes in the discoverable mode, but the same principles will work for Wi-Fi access points too. This association is similar to the above-mentioned data attachments for beacons. The main difference is the definition (the description) for a tag. BDP is based on the idea of "zero scene preparation". For example, any mobile users should be able to create (open) Bluetooth node right on the own mobile phone, associate some data with this node and so, make them available for other mobile users in the proximity. Figure 3 illustrates this idea. As existing node, we see here Bluetooth node in the car. Many modern cars nowadays are actually Bluetooth nodes. Car's owner can attach data to the own node. Other mobile users in the proximity can "see" Bluetooth node and use its identification (SSID, MAC-address) as key for obtaining associated data from the cloud. It is so-called hyper-local data concept. Data not only present some local information but could be prepared locally also. Instead of the car (Bluetooth node in the car) we can use just another mobile phone. A Bluetooth node (a tag) could be created programmatically. And programmatically we can attach some data to it. So, just one mobile application (in publishing mode) is enough for creating a new data channel. And the same mobile application (in browsing mode) could be used for reading data in the proximity.

Fig. 3. BDP data flow.

The simplest use case is a mobile classified. A mobile users creates an advertising (announce), links it to the wireless node on the own mobile phone and so, it becomes available for reading for other mobile users in the proximity. If the mobile phone (the mobile tag) is moved, all associated data will be "moved" too. Data are not associated with latitude/longitude pair (as in geo-location systems), but with ID of wireless node. Data are visible in the proximity of the node (in the proximity of the author) only.

Google does not describe the above mentioned database for beacons data attachments. BDP uses the classical key-value model for data. Attachments area described individually for the each Bluetooth point. So, there is a key (MAC-address) and a JSON text with linked data. It is a typical key-value data model. This data model is one of the most used models for NoSQL approach. One of the available examples of Open Source key-value stores is Apache Accumulo [23]. It is distributed key-value store. Actually, the whole database for BDP is a distributed hash table. The table rows as key-value pairs to provide a fast way to look up by a key item as attribute given by the value of a column qualifier of a row. In order to support lookups by more than one attribute of an entity, additional indexes can be built.

Data is represented as key-value pairs, where the key is comprised of the following elements: *RowID*, *Column* (*Family*, *Qualifier*, *Visibility*) and *Timestamp*. All elements of the Key and the Value are represented as byte arrays except for *Timestamp*, which is a *Long*. Accumulo sorts keys by element and lexicographically in ascending order. Timestamps are sorted in descending order so that later versions of the same Key appear first in a sequential scan. Tables consist of a set of sorted key-value pairs [24].

In terms of data design, BDP store contains the following information:

(recordID, MAC_address, data_array)

Each record describes a one data chunk (information element) for the given (MAC_address) Bluetooth node. Of course, we could have more than one information element for the same node.

The typical query requests data by MAC_address. So, it is a direct scan via the primary index and it will be fast.

JSON array for data chunks let present the various elements within data attachments. For example, plain text, phone number, email address, hyperlink, link to Twitter/Facebook/Linkedin profile or even an encoded image.

The basic algorithm as it is described above is based on the ideas of browsing data rather than push them to mobile users. BDP's context-aware "browser" obtains a list of the visible Bluetooth node. Than for the each node browser can perform database scan (lookup) and get data associated with this node. This request simply returns nothing in the case of Bluetooth nodes without attachments. All collected data could be packed in JSON array and this array will be returned back to the "browser". And the browser will perform data rendering. Nodes in the array could be sorted by the obtained RSSI (signal strength). Figure 4 illustrates this.

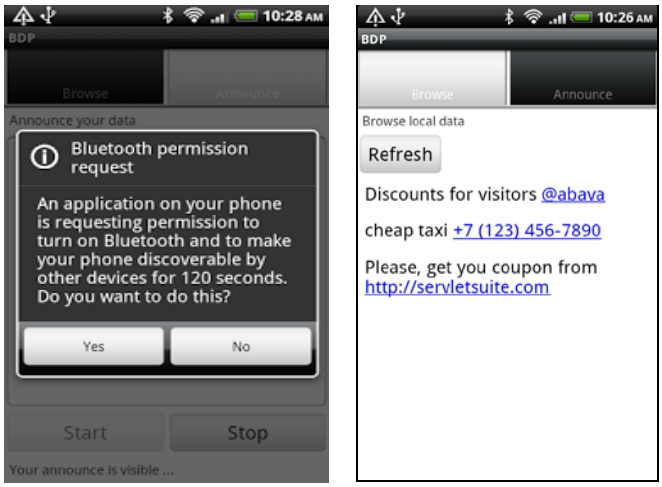

Fig. 4. BDP

As per collected statistics, the system can accumulate "browsing" events. An event here is the fact states that the device with address MAC_1 requests a data chunk provided by the device MAC_2 at the time *t*.

In the normal case, most of the nearby Bluetooth nodes will be "empty" (they will be out of BDP circle). So, we can decrease the number of database lookups with some cache. BDP uses a Bloom filter [25]. The Bloom filter is a method for representing a set of n elements $A = \{a_1, a_2,\ldots,a_n\}$ also called keys to support membership queries. Elements here are MAC-addresses for Bluetooth nodes.

V. Conclusion

In this paper, we discuss existing and upcoming software systems based on the network proximity. As a main result, we can present our list of requirements to the flexible solution, based on the wireless tags.

Any proposed system should support software-based tags. It should be possible to define tags and linked data with existing wireless infrastructure and/or existing mobile devices. We do not reject the idea of using dedicated hardware tags. We just highlight the fact that software-based systems are much more flexible, cheaper and finally allow much more services.

The wireless modules (Bluetooth, Wi-Fi) in the mobile devices make them the most popular and widely distributed sensors. With software based wireless tags, it is a most simple and convenient approach for context-aware programming in Smart Cities environments.

The push-based data delivery in case of wireless tags has got serious usability limitations. By our opinion, the browsing is a more promising approach for getting data in the proximity of tags. Finally, we can predict, that at the end of the day, mobile browsers will incorporate data about proximity. Any mobile browser is a mobile application too. And nothing prevents it, for example, to scan nearby wireless devices.

In the "browsing" mode collected statistics about data scanning is a direct analogue of web log, collected by any web server. This statistics is an important part of data mining for

ISBN: 978-1-61804-327-6

analyzing the behavior of mobile users and should be collected by beacons supporting frameworks.

Acknowledgment

We would like to thank prof. Manfred Sneps-Sneppe for the valuable discussions.

References

[1] Namiot, Dmitry, and Manfred Sneps-Sneppe. "On IoT Programming." International Journal of Open Information Technologies 2.10 (2014): 25-28.

[2] Namiot, Dmitry. "Network Proximity on Practice: Context-aware Applications and Wi-Fi Proximity." International Journal of Open Information Technologies 1.3 (2013): 1-4.

[3] Namiot, Dmitry. "On Indoor Positioning." International Journal of Open Information Technologies 3.3 (2015): 23-26.

[4] Namiot, Dmitry, and Manfred Sneps-Sneppe. "Geofence and network proximity." Internet of Things, Smart Spaces, and Next Generation Networking. Springer Berlin Heidelberg, 2013. 117-127.

[5] Sneps-Sneppe, Manfred, and Dmitry Namiot. "Spotique: A new approach to local messaging." Wired/Wireless Internet Communication. Springer Berlin Heidelberg, 2013. 192-203.

[6] Gomez, Carles, Joaquim Oller, and Josep Paradells. "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology." Sensors 12.9 (2012): 11734-11753.

[7] Cavallini, Andy. "iBeacons Bible 1.0." Online im Internet: http://meetingof ideas. files. wordpress. com/2013/12/ibeacons-bible-1-0. pdf 22 (2014).

[8] What are iBeacons? http://radar.oreilly.com/2015/04/what-are-ibeacons.html Retrieved: Jun, 2015

[9] Hundreds of London Buses to be Fitted with iBeacons http://performancein.com/news/2015/07/01/hundreds-london-buses-be-fitted-ibeacons Retrieved: Jul, 2015

[10] Eddystone https://github.com/google/eddystone Retrieved: Jul, 2015

[11] Proximity Beacon API https://developers.google.com/beacons/proximity/guides Retrieved: Jul, 2015

[12] Nearby API https://developers.google.com/nearby/ Retrieved: Jul, 2015

[13] Want, Roy. "The Physical Web." Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems. ACM, 2015.

[14] The physical web https://github.com/google/physical-web Retrieved: Jul, 2015

[15] Hansen, J., Gronli, T. M., & Ghinea, G. (2012). Towards cloud to device push messaging on Android: technologies, possibilities and challenges.

[16] Xu, Z., & Zhu, S. (2012, August). Abusing Notification Services on Smartphones for Phishing and Spamming. In WOOT (pp. 1-11).

[17] Miller, B., Nixon, T., Tai, C., & Wood, M. D. (2001). Home networking with universal plug and play. Communications Magazine, IEEE, 39(12), 104-109.

[18] UPnP, http://www.upnp.org, Retrieved: Jul, 2015

[19] Nordman, B., & Christensen, K. (2010). Proxying: The next step in reducing it energy use. Computer, 43(1), 91-93.

[20] Chandra, R., Padhye, J., Ravindranath, L., & Wolman, A. (2007, March). Beacon-stuffing: Wi-fi without associations. In Mobile Computing Systems and Applications, 2007. HotMobile 2007. Eighth IEEE Workshop on (pp. 53-57). IEEE.

[21] Banerjee, N., Agarwal, S., Bahl, P., Chandra, R., Wolman, A., & Corner, M. (2010). Virtual compass: relative positioning to sense mobile social interactions. In Pervasive computing (pp. 1-21). Springer Berlin Heidelberg.

[22] Namiot, D., & Sneps-Sneppe, M. (2014, October). CAT??? cars as tags. In Communication Technologies for Vehicles (Nets4Cars-Fall), 2014 7th International Workshop on (pp. 50-53). IEEE.

[23] Sen, R., Farris, A., & Guerra, P. (2013, June). Benchmarking apache accumulo bigdata distributed table store using its continuous test suite. In Big Data (BigData Congress), 2013 IEEE International Congress on (pp. 334-341). IEEE.

[24] Namiot, D., & Sneps-Sneppe, M. (2015). On Mobile Bluetooth Tags. arXiv preprint arXiv:1502.05321.

[25] Broder, A., & Mitzenmacher, M. (2004). Network applications of bloom filters: A survey. Internet mathematics, 1(4), 485-509.