Published in Proceedings of the 10th EAI International Conference on Ad Hoc Networks (ADHOCNETS 2018), Cairns, Australia, 20-23 September 2018, which should be cited to refer to this work.

HACIT2: a privacy preserving, region based and blockchain application for dynamic navigation and Forensics in VANET

Decoster Kevin[1] and Billard David[1]

University of Applied Sciences Western Switzerland in Geneva - HES-SO
{kevin.decoster,david.billard}@hesge.ch

Abstract. The current architecture for VANET related services relies on a Client-Server approach and leads to numerous drawbacks. Among them, data privacy concerns and service availability are of prime importance. Indeed, user data collected and stored in servers by providers may be used by third-party services. Particularly for navigation, users submit their GPS position in order to obtain road traffic information and alternative paths. These services treat user privacy for their own purpose (commercial or not) (Beresford and Stajano, 2004) even if GPRD (European Parliament, 2014) is now enforced in Europe. We propose an innovative approach using blockchain technology to avoid the use of third parties services, which enable dynamic navigation rerouting within a fixed geographic zone while ensuring user anonymity. Furthermore, the approach will allow for legal authority to enable forensic analysis of the ledger without unnecessary violation of the user anonymity and privacy.

Keywords: VANET, Raspberry Pi, Android, Navigation, Hyperledger Fabric, Privacy, Blockchain, Forensics.

1 Introduction

While many services offer dynamic rerouting navigation based on collaborative data (such as Google maps), none grants the user with a total control of its data. The proof of concept presented in this paper focuses on the collaboration of intelligent cars for determining the best driving route and avoiding traffic jams similarly to what current services do, but without the use of a central Internet service. By forbidding the use of centralized services like Google Maps or Tomtom Go Mobile, the traffic state shared by every users is kept at every peer's side in the form of a shared ledger. This ledger is updated using a consensus algorithm which guarantees that all peers share the same blockchain. Using the cellular network, a user can submit a transaction containing the newly measured road speed in order to update the ledger. Besides, using the event system of the peers, a user can listen for newly submitted transactions and, if necessary, updates its current navigation instructions by computing the new shortest path given the current road weight states. The proposed project is currently being implemented

using the IBM blockchain framework (IBM, 2017) on top of Hyperledger Fabric developed by the Linux Foundation (linux Foundation, 2016), which enable an extensive framework for blockchain technology implementation.

The challenge of this approach is the feasibility of the communication and computing in the mobile device. Indeed, running a blockchain node requires computations and communication capacities which can lead to difficulties in a dynamic mobile network. We will discuss the pros and cons and propose a system using an external device such as a Raspberry Pi to delegate the computing and the storage of the peer client. A consequent part consists in designing an efficient communication protocol between the mobile device and the computing unit.

To handle the geographic graph, OpenStreetMap (OpenStreetMap contributors, 2017) files, GraphHopper Java library (Graphhopper dev, 2017) and OSMAnd Android library (OSMAnd dev, 2017) are respectively used for the map file, the graph handler and the dynamic navigation UI on Android.

Finally, using this innovative approach, the application can enable forensics capabilities. As a matter of fact, legal officers should access navigation path in the immutable ledger without violating user anonymity. For instance, we can foresee that in case of an accident, a user would have an interest to prove its behaviour. This is possible using the data transaction chain.

2 Related work

The problem of navigation in VANET using only local information has been widely studied this last decade. For instance, the authors of (Wang et al., 2017) propose an anonymous and secure navigation schemes in VANET. While they satisfy all requirements for security and privacy, they still assume the use of third parties as Trusted Authorities (TA) to de-anonymize the car ids. Furthermore, they use direct vehicle communication (through Wifi or radio wave communication) within a dynamic ad-hoc network and as a result, only partial and local information regarding the traffic is shared among moving nodes, as opposed to a system that centralizes all road traffic information such as Google Map.

To the best of our knowledge, although the security in VANET is a well-researched field ((Raya and Hubaux, 2007)), no paper fully addresses the forensics concern. Indeed, no work already proposes a system enabling dynamic rerouting and forensics for the mobile device using a fully implemented blockchain technology. For instance, (Leiding et al., 2016) uses blockchain in VANET. However, they use it for monetary applications such as an automatic smart contract for insurance or tolling and uses Ethereum to host the smart contracts (see (Wood, 2014)). Without the need for a monetary support (and thus proof of work through mining), our blockchain can achieve consensus without computationally expensive proof-of-work, for instance with Practical Byzantine Fault tolerance (PBFT) algorithm.

3 Hyperledger Fabric

Hyperledger Fabric (HF), developed by the Linux Foundation, proposes a framework for developing permissioned blockchain technology. As opposed to bitcoin network, the access to the blockchain is controlled by an entity called the *Membership Service Provider*, which grant access to users and peers with the cryptographic material (certificate and keys) delivered by a certificate authority (CA).

The blockchain includes a ledger of transactions but also a representation of the world state through a key-value database. Access, queries, modifications and Smart contract are defined using the blockchain rule called Chaincode. This allows efficiently querying and modifying the dataset without having to analyse the whole chained data transactions.

We distinguish 3 different types of nodes:

- *Peers*: a basic node which stores an up-to-date copy of the ledger and chaincode rules. It continuously keeps tracks of information through the gossip protocol running among all peers, see (Shah et al., 2009).
- *Client* (user): Which consists of the end-user who owns an authorized cryptographic material. It runs an SDK which grants him access to the peers API functions. The client connects to a peer to submit transaction proposals.
- *Orderer*: The orderer is a special peer node, whose role is to gather and order validated transactions until a block can be made and broadcasted to all peers. It can either be running on a server (centralized) but can be chosen randomly among peers (peer elected to act as the Leader peer).

To update the ledger, a client creates a transaction and sends it to one or several peers for endorsement (depending on chaincode rules). Once the transaction meets the endorsement policy, it is forwarded back to the client who sends it to the orderer for verification and broadcasting. Upon verification, the orderer broadcasts the transaction to all peers that will check it and update the ledger accordingly. An extensive documentation of the Hyperledger Fabric framework can be found in (linux Foundation, 2016).

4 System model

Inside our system model, we distinguish the *traffic congestion detector* client, which is the module in charge of detecting a traffic jam situation and submitting speed changes to the shared ledger and the *dynamic navigation rerouting* server, which is the module in charge of detecting a road speed change (from an HF event) on a user's path and recompute the route accordingly.

4.1 The chaincode

The following listings show how the chaincode fits into our application. In Listing 1.2, we state the different assets accessible in the ledger. Note that all

users are only identified with a unique random ID and that a road asset contains an ID *roadId*, a list of submitted speeds *speeds* (and the corresponding list of associated timestamps *timestamps*) and the edge segment default speed $defaultSpeed$ initialized from OpenStreetMap predefined tag information. Finally, the transaction *SubmitSpeedChange* which, given a new speed *newSpeed* and new timestamp *newTimestamp* for a road asset whose ID is *assetId*, modifies the *roadAsset* using the function shown in listing 1.1.

Furthermore, in listing 1.1, we observe the transaction *onSubmitSpeedChange* that fires an event $SubmitSpeedChangeNotification$ when submitted. Basically, it simply appends the new speed and timestamps to the corresponding stored list within that asset. It ensures that the list size is no longer than N, a predefined constant, defined in function of the number of edges in the geographically bound map. Indeed, greater is N, more storage for the initial state database will be required.

Last but not least, all aforementioned functions are accessible through a REST API on the device storing the peer node, this allows cross-platform and convenient communication in the local network interface between our graph handler and the Hyperledger Fabric peer.

```
function onSubmitSpeedChange(x) {
    push(submitSpeedChange.roadAsset.timestamps, x.newTimestamp, MAX_NUMBER);
    push(submitSpeedChange.roadAsset.speeds, x.newSpeed, MAX_NUMBER);
    return getAssetRegistry('org.hacit.hes.RoadAsset')
        .then(function (assetRegistry) {
            var event = getFactory().newEvent('org.hacit.hes', '
    SubmitSpeedChangeNotification');
            event.roadAsset = x.roadAsset;
            emit(event);
            return assetRegistry.update(x.roadAsset);
        });}
```

Listing 1.1: Chaincode

```
event SubmitSpeedChangeNotification { --> RoadAsset roadAsset }
participant User identified by assetId { o String assetId }
asset RoadAsset identified by roadId {
  o String roadId
  o Integer [] timestamps
  o Double defaultSpeed
  o Double [] speeds }
transaction SubmitSpeedChange {
  o Double newSpeed
  o String assetId
  o Integer newTimestamp
  --> RoadAsset roadAsset }
```

Listing 1.2: Ledger Model

4.2 Traffic congestion detector client

Given an accumulated list of the user's GPS coordinates, we find the corresponding edge and extracts its road ID r using a map matching algorithm provided by the GraphHopper library (Newson and Krumm, 2009). Fig. 1b shows the process for a user to update its current speed cs measured at timestamps ts to

the corresponding road asset r (in the ledger) and thus, the process to create the proper corresponding transaction tx.

The challenge in this step is to decide whether or not the vehicle is in a traffic jam, or simply stopped at the red light for example. Basically, it gathers GPS coordinates until it detects that the road has changed. By computing the average speed in the middle part of this GPS trace, we can apply a threshold to decide if there is congestion.

4.3 Dynamic navigation rerouting server

The other module is the dynamic navigation server, which listens for ledger update (*i.e.* new events submitted by peers). Alongside the main algorithm, we created a speed extractor, that given all submitted speed changes for a given road, extract the current road speed while removing outliers, using unsupervised clustering.

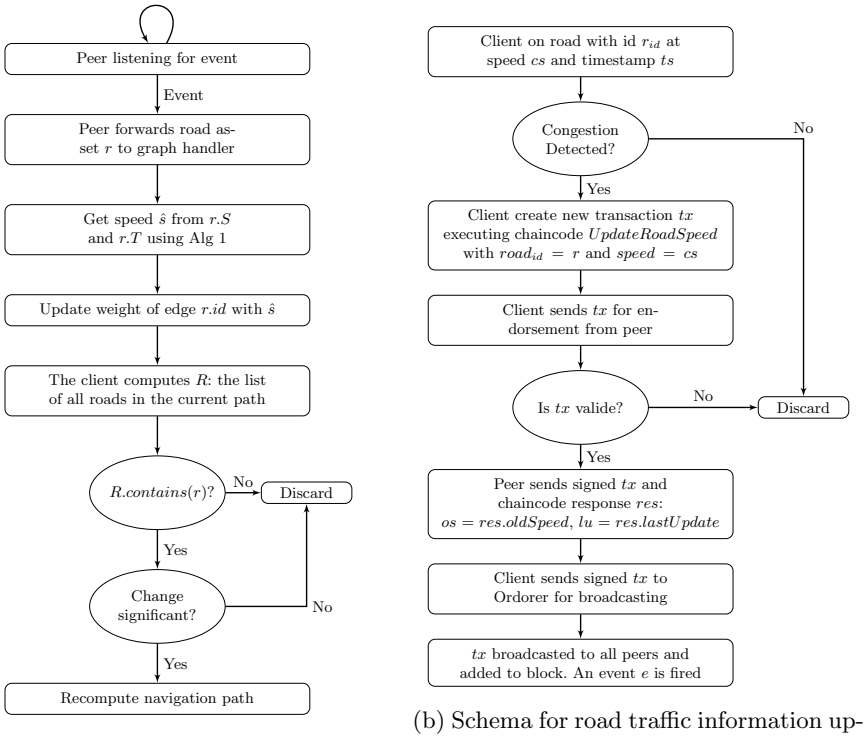
Algorithm The process describing our dynamic navigation is shown in Fig.1a and can be summarized as follows:

- An event is fired for a given road id: The HF module connects to our OSM graph handler with a POST request through the *localhost* interface, in order to send the road id and the corresponding list of speeds S and timestamps T (defined in Listing 1.2).
- The Graph Handler feeds the speeds and timestamp list to a speed detector, which will perform Algorithm 1 in order to find the weight from the most recent speed cluster centre while removing outliers.
- If the road modified is within the future road path, and if the change is significant, the user recomputes the navigation path against the up-to-date weighted graph.

Speed extractor The HF peer forwards the list of speeds S and timestamps T contained within the road Asset whose is firing the event. The goal of the speed extractor (described in Algorithm 1) is to cluster the 2D dimensional array $X = [T, S]$ and finds the earliest cluster using the $DBScan$ algorithm, initially proposed in (Ester et al., 1996), using the Java library (Apache Fondation, 2017). Once the centre cx, cy is found, we forward cy (*i.e.* the speed) to the graph handler.

If there is not enough data or if the clustering fails, we simply use the weighted average Equation (1) so that speed measurements with earlier timestamps have more weight:

$$\begin{aligned} \hat{s} &= \frac{\sum_{i=1}^{N} w_i s_i}{\sum_{i=1}^{N} w_i} \quad \text{with} \;\; w_i = 1 - \frac{t_i}{\sum_{j=1}^{N} t_j} \\ \Rightarrow \hat{s} &= \frac{\sum_{i=1}^{N} s_i}{N-1} - \frac{\sum_{i=1}^{N} t_i s_i}{(N-1) \cdot \sum_{i=1}^{N} t_i} \end{aligned} \tag{1}$$



(a) Schema for dynamic navigation rerouting

(b) Schema for road traffic information update to shared ledger

Fig. 1: Procedures

5 Communication system

This section aims to describe how the system communicates between all the modules presented in the previous Section 4. Particularly, how and where are executed the HF peer loop, the Graph handler loop and the UI client for dynamic navigation in both the external device (*i.e.* Raspberry Pi) and mobile device (*i.e.* Smartphone).

5.1 Peers on external device

A Raspberry Pi is a small computer having a dedicated operating system running Linux. The version 3 Model B contains 1GB RAM, Wifi antenna and an external SD card for storage, see (Upton and Halfacree, 2014). Therefore, it has enough capabilities to run efficiently the HF peer node and the graph handler on such device. It communicates with the mobile device through wifi as shown in Fig. 2a,

Algorithm 1 Speed extractor algorithm

Require: X a 2-D array

```
1: function Extract speed(X)
2:     cx ⇐ Null
3:     cy ⇐ −Inf
4:     if shape(X)[2] > 15 then
5:         db ⇐ dbscan(X, eps : 4.5, min : 5)
6:         labels ⇐ db.labels_
7:         for k in unique(labels) do
8:             mask ⇐ (labels == k)
9:             x ⇐ X[mask]
10:            [cxx, cyy] ⇐ mean(X)
11:            if cyy < cy then
12:                cx ⇐ cxx
13:    if cx is Null then
14:        t ⇐ X[0, :]
15:        v ⇐ X[1, :]
16:        cx ⇐ weightedAverage(t, v)          ▷ "Equation (1)"
       return cx
```

to exchange new road path to the UI, or new speed estimate alongside GPS coordinate.

Then, we have two servers, the graph handler running on port 4567 and the HF module running on port 8080 and we expose the REST client with the following API:

- The mobile Android app:
  1. *updatePosition*: PUT request to submit current speed and position.
  2. *getPathStatus*: GET request returning *true* if path was modified since last query.
  3. *getPath*: GET request to retrieve the list of navigation instructions. It will immediately update the current navigation instruction to the user interface.
  4. *initializeJourney*: POST request to initialize the graph handler with the destination and starting point.
  5. *updatePath*: PUT request to force the re-computation of the shortest path using Dijkstra algorithm (Dijkstra, 1959), return the newly computed navigation instruction
- The HF module Client:
  1. *putRoadAsset*: PUT request executed whenever the HF module detects a new event (*i.e.* a road speed update) and forward the corresponding road asset to the graph handler.
- The graph handler Client (to HF module server):
  1. *postTransaction*: Executed by the graph handler whenever a traffic jam is detected. It forwards the transaction to the HF module, in addition

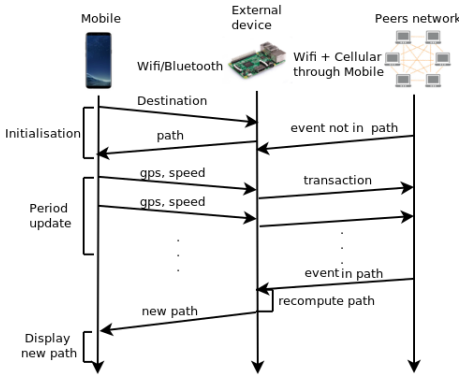
with the road ID, new speed, and timestamp. After some time, this should fire an event for all listening peers on the network.

We assume that the initial ledger state contains all edges within the imported OSM file. This can be done while creating and instantiating the production chaincode by the administrator before deploying the peer's network.

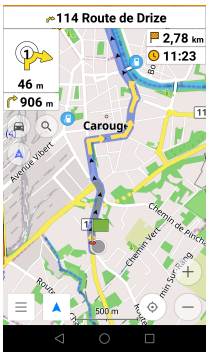
5.2 Internet connection and UI client on mobile device

A prerequisite for our system to work, the HF module needs to be connected to the internet. Therefore, we share the connectivity of the Android device with the raspberry pi and thus connect it to the internet through the Android Wifi Access Point. The pre-configured device will automatically try to connect to the router (*i.e.* The android device) and will obtain IP address 192.168.43.155 once the user enables *shared connectivity*. This IP is then used to perform the REST client calls, expressed in the previous part, between the two devices.

Finally, once initialization is completed and first navigation path instructions are received, the application opens *OSMAnd* dynamic navigation (see Fig. 2b) through Intent and the *OSMAnd* API. While the UI shows the navigation, a background process is continuously checking for new path updates, and if any, sends the updated path instructions to the *OSMAnd* UI. This causes *OSMAnd* to drop the current instructions for the new ones, and thus, updating the UI.



(a) Communication between devices



(b) OSMAnd UI

Fig. 2: The system

5.3 Limitations

The presented blockchain application, running on the mobile device, leads to several complications:

- *The communication burden and latency:*
  The system relies on the cellular network shared by the mobile device to listen to other peers and submit transactions. Therefore, we expect a fair use of the user's cellular plan. Nowadays, it is common to have unlimited bandwidth usages but most of the data will be synchronized through WIFI before the user drives with the up-to-date system. Moreover, the responsiveness of the system is bounded to the block broadcasting frequency, which can be tuned to meet a trade off between resources usage and more frequent information updates. We are confident that submitting a new block every few minutes is feasible.
- *The storage cost:*
  Table 1 shows the different ledger sizes regarding the size of the initial asset for the transactions and road asset initialization. Overall, we expect a ledger of around $842MB$ $(317 + 525)$ for one day of utilization for the average Geneva traffic per day (500000 transactions). By having a ledger pruning mechanism inside the chaincode rules, we can guarantee that the data size does not exceed a certain limit. Given the storage size available on our device, we can easily extend these limits to more than dozen of gigabytes.
- *The geographically bounded application:*
  In this project, we assumed the use of our system within the boundary of the city of Geneva. Indeed, we realized that most traffic information is only useful within close range for most of the users. From that conclusion, we assumed the use of a chaincode (*i.e.* ledger) per geographically bounded zone (*e.g.* city). In the future, other zones will be used and a system acting as CA to assign dynamically users to the proper chaincode (*i.e.* area) will be studied and implemented.
- *User incentive* As every dynamic navigation system, the efficiency of the routing is directly related to the quality and quantity of the data. More user use the system, more precise can the routing be. As for every blockchain application, the principal of decentralization makes the system more complicated to use for the end-user and is often the bottleneck for world-wide acceptance.

5.4 Case study

Let's imagine a person using our system going to work in a croweded city. At first, he will have to take the station (external device), which was synchronizing/charging at home, with him. The station will automatically connect to the mobile device while the user opens the specific system app. Upon navigation initial instruction, the station will listen for new transaction (*i.e.* road traffic update) and if necessary, forward new instructions to the end-user through the mobile application. In the background, the station

Table 1: Ledger size per asset size

Asset	Road		Transactions	
nb	1	68580	1	50000
size (MB)	0.48	317	0.11	525

will perform all processes to keep the distributed ledger up-to-date and handle user's GPS position (and thus, traffic information). Upon arrival, the user will synchronize the station with the local WIFI and charge it until next departure.

Such scenario will be numerically simulated to assess the performance of the overall system.

6 Forensics

The architecture of the proposed application allows any user to have access to the history of transactions and thus, it enables forensics.

6.1 Ledger Back-crawling

The architecture allows forensics capabilities for judiciary or insurance claims. With a public shared and immutable ledger provided by the permissioned blockchain, all users can access all submitted transaction and thus, the history of road speed modifications. In another words, by crawling the ledger of chained transactions and given a specific user ID, one can extract the transactions submitted by this user. Thus, given the list of all these submitted transactions, one can extract the road segment ID alongside the corresponding timestamps and speed to create a navigation timeline for this user. The user can then guarantee that he had signed the extracted transactions using his private key.

The efficiency of the forensics system is directly related to the frequency at which the user submits transactions. In another word, if the user does not submit any transaction, there will be no possibility to backtrack his whereabouts. As such, we only provide with this system a tool to help the judiciary or insurance to make the decision, as an extension to the main application.

6.2 A word on Privacy

Regarding our back-crawling algorithm and the confidentiality of our system in general, we grant access to the application through cryptographic material obtains from our CA. Even though the state database contains only the up-to-date list of road asset, the ledger contains the transactions submitted by any user, hidden behind his $userId$.

Although our approach is not anonymous, it is close to pseudonymous (similar to the bitcoin network, see (Androulaki et al., 2013)). However, one can analyse the ledger and extract meaningful pattern that can lead to the real user's identity.

Fortunately, Hyperledger Fabric will introduce in the version 1.2 a privacy technology known as Zero-Knowledge (ZK) Proof-based implemented has an Identity mixer (Au et al., 2006) and ZK-AT (Zero-Knowledge Asset Transfer). As a result, the identity of the participant issuing the transaction will stay hidden behind the identity mixer and thus, guaranteeing total privacy, similar as what is used in cryptocurrencies such as ZCash.

7 Conclusion

7.1 Summary

This project proposes an innovative variant for a decentralized system of navigation that emancipates the user from using a centralized service. Instead of communicating directly with nearby cars to retrieve only local traffic information, the users submit through cellular network global information about the traffic. This information is stored in assets within a permissioned ledger and can be updated with transactions. A system of event listens for new transactions (and thus, traffic update from other users) which forwards the information to another module handling the locally stored weighted graph. If necessary, the shortest path is recomputed and forwarded to the navigation user interface running on the mobile device making the navigation dynamic.

By using an external device such as a Raspberry Pi to run the blockchain and graph modules, we delegate the computing and storage cost to a unit able to easily handle all processes and make the experience as user-friendly as possible. The user can just plug the device to the car power and share his or her mobile wifi connection.

Furthermore, every user can access the shared ledger and thus, retrieve the list of submitted transactions for forensics purposes. By having a zero-knowledge mechanism implemented in the next version of Hyperledger Fabric, we will be able to guarantee that the identification of the user is not possible and thus, protecting his or her privacy.

7.2 Future works

Our work so far was to design and implement a system able to dynamically route user using blockchain technology. However, a numerical simulation must be undertaken in order to optimize several parameters such as the size of the array in a road asset (currently equals to 100), the real latency, the storage cost and the ledger pruning time, the bandwidth used or the percent of user using our application in order to make it efficient.

In that manner, we are currently working on a numerical traffic simulation of Geneva, where a percent of the agents (acting as a driver) use our decentralized system. The simulation uses the framework Sumo (Krajzewicz et al., 2012). After these parameters are optimized, we will perform a full-size test assessing the functionalities of our system.

Moreover, an interesting variant would be to use a Road Side Unit (RSU) to store the ledger. In other words, these units would act as a peer for the blockchain network. The cryptographic material identifying a user and the graph would still be stored on the user's mobile side alongside the SDK to create transactions. Once a user comes into range within the RSU, it will send the list of congestion information as transactions and expect the RSU to sign them and send back the list of transactions since the user last connects. However, as of now, the feasibility of such system is not known and must be studied.

Bibliography

Androulaki, E., Karame, G. O., Roeschlin, M., Scherer, T., and Capkun, S. (2013). Evaluating user privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 34–51. Springer.

Apache Fondation (2017). Commons Math Java library. http://commons.apache.org/proper/commons-math/userguide/ml.html.

Au, M. H., Susilo, W., and Mu, Y. (2006). Constant-size dynamic k-taa. In *International Conference on Security and Cryptography for Networks*, pages 111–125. Springer.

Beresford, A. R. and Stajano, F. (2004). Mix zones: User privacy in location-aware services. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 127–131. IEEE.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

European Parliament (2014). European Parliament legislative resolution of 12 March 2014 on the General Data Protection Regulation. Technical Report (COM(2012)0011 C7-0025/2012 2012/0011(COD)).

Graphhopper dev (2017). Graphhopper java Librairy. https://www.graphhopper.com/.

IBM (2017). IBM Blockchain Platform. https://ibm-blockchain.github.io/develop/. [Online; accessed 21-April-2018].

Krajzewicz, D., Erdmann, J., Behrisch, M., and Bieker, L. (2012). Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138.

Leiding, B., Memarmoshrefi, P., and Hogrefe, D. (2016). Self-managed and blockchain-based vehicular ad-hoc networks. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 137–140. ACM.

linux Foundation, T. (2016). HyperLedger Fabric docs. https://hyperledger-fabric.readthedocs.io/en/release/. [Online; accessed 21-November-2017].

Newson, P. and Krumm, J. (2009). Hidden markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 336–343. ACM.

OpenStreetMap contributors (2017). Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org.

OSMAnd dev (2017). OSMAND. https://osmand.net/.

Raya, M. and Hubaux, J.-P. (2007). Securing vehicular ad hoc networks. *Journal of computer security*, 15(1):39–68.

Shah, D. et al. (2009). Gossip algorithms. *Foundations and Trends® in Networking*, 3(1):1–125.

Upton, E. and Halfacree, G. (2014). *Raspberry Pi user guide*. John Wiley & Sons.

Wang, L., Liu, G., and Sun, L. (2017). A secure and privacy-preserving navigation scheme using spatial crowdsourcing in fog-based vanets. *Sensors*, 17(4):668.

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151.