Friedhelm Victor, Sebastian Zickau

Geofences on the Blockchain: Enabling Decentralized Location-based Services

Victor, Friedhelm; Zickau, Sebastian (2018): Geofences on the Blockchain: Enabling Decentralized Location-based Services. In: BlockSEA 2018 – The 1st Workshop on Blockchain and Sharing Economy Applications (co-located with The 18th IEEE International Conference on Data Mining (ICDM 2018) Singapore, November 17, 2018). Piscataway, New Jersey: IEEE.

Geofences on the Blockchain: Enabling Decentralized Location-based Services

Friedhelm Victor and Sebastian Zickau
Service-centric Networking
Telekom Innovation Laboratories | Technische Universität Berlin
Berlin, Germany
friedhelm.victor@tu-berlin.de, sebastian.zickau@tu-berlin.de

***Abstract*—A decentralized ride- or carsharing application is among the early proposals of what smart contracts on blockchains may enable in the future. To facilitate use cases in the field of location-based services (LBS), smart contracts need to receive trustworthy positioning information, and be able to process them. We propose an approach on how geofences can be defined in smart contracts, and how supplied positions can be evaluated on whether they are contained in the geofence or not. The approach relies on existing location encoding systems like Geohashes and S2 cells that can transform polygons into a grid of cells. These can be stored in a smart contract to represent a geofence. An oracle run by a mobile network provider can submit network-based positioning information to the contract, that compares it with the geofence. We evaluate the location encoding systems on their ability to model city geofences and mobile network cell position estimates and analyze the costs associated with storing and evaluating received oracle-positions in an Ethereum-based smart contract implementation. Our results show that S2 encodings perform better than Geohashes, that the one-time cost of geofence definition corresponds linearly with the number of grid cells used, and that the evaluation of oracle-submitted locations does not incur high costs.**

I. Introduction

In recent years the sharing economy has given rise to new companies. They enable the provisioning, use, and consumption of services and goods between individuals, typically coordinated through an online platform. Well known examples include AirBnB – which allows anyone to rent out their apartment – and Uber, which facilitates ride sharing and taxi services. Whereas the sharing economy is frequently described with peer-to-peer activity or marketplaces [1], [2], [3], the platforms that enable it are very centralized systems.

With the advent of blockchain technology, new opportunities for decentralized systems in the sharing economy emerge. Smart contracts – program code that is executed in a redundant, decentralized fashion – allow for novel application scenarios and business models. These typically cut out intermediaries that traditionally provided trustworthy interaction between parties. Smart contracts can provide trustworthy interaction, but in many use cases also need trusted information about the outside world, that isn't readily available in a blockchain context. For example, a contract designed to insure against flight delays needs information about delays as they are happening in the real world. It cannot simply request this information from a webserver on its own. All blockchain participants would need to perform the request on behalf of the contract, retrieving the same result. This would also have to be reproducible by anyone at any time. As this is not possible, such outside information can be submitted into the blockchain through prediction markets, where participants vote on a question posed by an inquirer, or through so-called *oracles* that are single entities claiming that certain events occurred in the outside world. Once external data is inserted, smart contracts can make decisions based on the available information.

To facilitate a use case such as a decentralized ride-sharing service, it would be very beneficial if a user's location information were accessible to a smart contract. Assuming such a contract is designed to automatically charge a user based on time and distance travelled, information about the start and end location would be needed. When sharing a car, it may be desirable to restrict the allowed driving area geographically. For example, a smart contract could enforce financial penalties if a specified geographic area is exceeded or the car has not been returned to a certain location in time.

While these are specific examples of location-based services (LBS) in which trustworthy location data may be useful, there are many other scenarios in which they would pose an advantage. To name a few:

- In supply chains, for the verification of the origin or processing locations of certain products and materials.
- In logistics, for cargo tracking and delay verification.
- In smart cities, for the authorization of social benefits or the restriction of local currency distribution.

For the use cases described above, a geographic region needs to be defined that can be compared with an inserted position inside a smart contract. Such inserted positions cannot necessarily be trusted, if they are a user's or a device's self reported position (as GPS positions for example can be spoofed [4]). The recently published Geocoin platform [5] for example, relies on a smartphone's GPS position. Location information that is hard to manipulate would prevent its users from tricking the system. The likelihood of attacks will depend on the magnitude of a potential monetary gain when forging a submitted location.

In this paper, we propose the approach of using an oracle operated by a mobile network operator, that can submit network-based position estimates to a smart contract. The resulting

position claims are not made by the users or devices, but on behalf of them, through an existing, reliable infrastructure that is significantly harder to manipulate. We further address the representation of geographical areas as geofences for smart contracts, and how received positions can be determined to be part of the geofence or not, so that additional contract routines can be triggered accordingly. Our key contributions are the following:

1) We provide an approach to represent geofences within smart contracts to facilitate blockchain-supported LBS.
2) We compare Geohashes and S2 cells for geofence and mobile network cell encodings in smart contracts, finding that S2 cells need less grid cells while achieving a similar polygon estimation performance.
3) We evaluate the system with an Ethereum-based implementation with respect to smart contract execution costs both for geofence definition and location insertion.

II. Background and Related Work

A. Smart contracts and the EVM

Smart contracts are computer programs that can be deployed onto a blockchain that supports it. The original idea of enforceable rules without a third party was first proposed by Nick Szabo [6]. Once deployed to a blockchain, users can interact with the program by sending transactions to it. Thereby functions can be specified, and parameters be submitted. The Ethereum Virtual Machine (EVM) is one of the most popular smart contract runtimes. It is used not only on the main Ethereum blockchain [7], but also on other blockchains that support it or provide compatibility. Programs can be written in high level languages like Solidity or Serpent, and are compiled to EVM bytecode. Each bytecode instruction has a cost attached to it that is measured in units of gas. Users need to pay for the amount of gas that is consumed by their transactions. As the most expensive EVM instruction is the creation or modification of storage variables, smart contracts need to be designed to use permanent storage sparingly.

B. Geofences, Background Tracking, and Positioning

Greenwald et al. use the term *geofencing* when messages are pushed to a subscriber triggered by an individual crossing a virtual border [8]. This virtual border is referred to as a *geofence*. Küpper et al. [9] describe the necessary technological device capabilities and protocols which enable geofencing applications and use cases through background tracking of devices. Current geofencing frameworks also include information about previously visited areas, Rodriguez Garzon and Deva call these types geofencing 2.0 [10].

To determine whether a geofence applies or not, a given position must be determined to be inside or outside of it. Given that spoofing should not be easy, we consider mobile network position estimations, as provided by mobile network operators. Cellular positioning methods range from $>$10km in rural areas to 50m-1,000m in urban areas [11]. Multiple positioning techniques like Angle of Arrival (AOA), Time of Arrival (TOA) and Time Difference of Arrival (TDOA) [12] offer improved accuracy over basic cell sector positioning systems. Frattasi and Della Rosa state that the median area of uncertainty for network-based positioning in urban areas is between 0.4-0.5km^2 [13]. Wymeersch et al. [14] give an outlook that with the future introduction of 5G networks potential performance of positioning could be down to $<$10cm, due to the combination of high carrier frequencies, large bandwidths, large antenna arrays, device-to-device communication, and ultra-dense networking.

C. Blockchain location protocols and startups

Several startups propose to use blockchain technologies that make use of location data. To the best of our knowledge, they do not rely on a mobile network operators infrastructure or implement geofences so far.

1) FOAM: FOAMs[1] core project is the introduction of a decentralized protocol for Proof of Location (PoL) on the Ethereum blockchain. FOAM introduces a *crypto spatial coordinate* that make use of Geohashes, a proof of location protocol, and a spatial index web app. It wants to incentivise the deployment of Low Power Wide Area Networks (LPWAN) hardware, employing techniques such as TOA and triangulation, to determine the location of a participant.

2) SIKORKA: Sikorka[2] is also based on the Ethereum blockchain technology. The approach is centered around the idea of proving presence at a certain location, by QR-code interaction or question-answering schemes. In 2018 Sikorka plans to integrate dedicated tamper-proof hardware in their protocol deployed at locations where smart contracts can be used to interact with museums and tourist sights for example.

3) Platin: Platin[3] is a PoL protocol that detects spoofing attempts using anomaly detection on the sensors obtained from a users mobile device. Additionally, they want to leverage high-trust users and dedicated devices as trust-anchors.

Finally, several ridesharing startups like DACSEE, Chasyr and Helbiz exist, that claim to reduce costs by employing blockchain technology. However, there seems to be very little information on whether blockchain-enabled LBS will be part of the solution.

D. Proof of location blockchains

Brambilla et al. [15] propose a decentralized, infrastructure-independent PoL scheme based on blockchain technology, including privacy preservation. The scheme considers a peer-to-peer network with connected mobile devices that also have an enabled short-range communication technology, such as Bluetooth. In this network there are *Prover* nodes (connected to the Internet) and *Witness* nodes. A Prover node i sends a signed location request with an identifier to a Witness j including the hash of the latest block on the blockchain. The location of the Prover must not be further away, then the maximum distance of the short-range protocol used. The Witness node signs its reply with its geographic location

[1] https://foam.space/
[2] http://sikorka.io/
[3] https://platin.io/

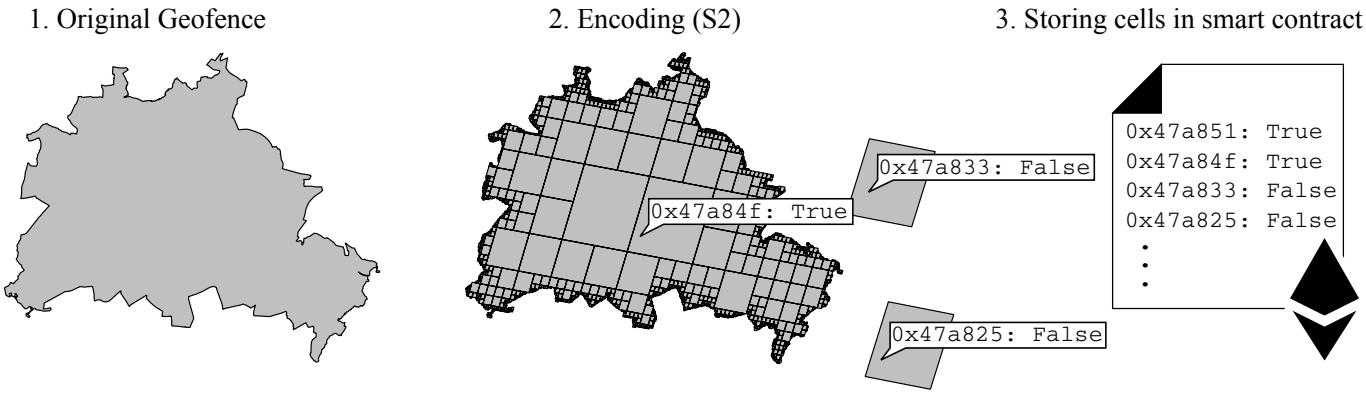

Fig. 1. A geofence is encoded with S2 cells and stored in a key-value map within a smart contract. Undefined and invalid keys are False (0).

and the identifier. The Prover verifies the response. If the verification is correct the PoL is broadcasted to the network.

Dasu et al. [16] propose a different approach for PoL. In order to increase scalability and security, they divide a blockchain infrastructure into a hierarchical tree of smaller, localized ledgers. Servers or cellular towers that have a well-defined location can issue *location certificates* and participate in proof of location mining. Trustworthiness is achieved by using cryptographically-signed IP packets.

E. Location Encoding Systems

For the representation of geofences and mobile network positioning estimates, a location encoding system, also known as geocoding systems, is required and needs to fulfill the requirements of a given use case. For blockchain use cases, storage and computational efficiency is of primary concern. Therefore, geofences need to be approximated using as little storage as possible, while maintaining the ability to determine whether a given location is inside or outside of it.

The most common location representation is the longitude and latitude coordinate system of the World Geodetic System (WGS 84) [17]. Most other systems encode these coordinates into a single code, the most famous being the Geohash[4]. Other systems partition the earth into tiles and present a representation of them, such as what3words[5]. Location encoding systems are designed with various requirements in mind, such as human memorizable, avoidance of similar characters, multiple resolutions with various numbers of digits, comparability, on/offline de/encoding, static, and license free usage. Here, we reduce the set of encoding systems to those that offer a hierarchical spatial data structure, and employ space filling curves.

a) Geohash: Example: u33db2m. The key characteristic of a Geohash are its use of alphabetic characters and numerals and a z-order space-filling curve. There are 12 precision levels, one for each character. Removing characters from the end of the code reduces the precision and increases its size. Geohashes have been used to represent Geofences [18].

b) Google S2: Example: 5163463834198867968, corresponding to the hex value 0x47a84f0000000000, often abbreviated to the token 47a84f. S2 cells have been developed by Google employee Eric Veach[6] to be used for geospatial indexing. It makes use of the idea of a hierarchical decomposition of a sphere. S2 cells are a compact representation of regions or points using a Hilbert space-filling curve. They can be represented with 64-bit integers, allowing for 31 precision levels. Similar to Geohashes, S2 cells can be reduced in terms of their precision, while providing more distinct precision levels. Each cell is represented by an 64 bit integer value (examples shown in Table I).

III. Concept

In this section we describe how we can specify geographical areas such as geofences within smart contracts. In combination with supplied geopositions, the contract can determine whether the supplied position overlaps with the geofence, and trigger actions accordingly. We then provide details on how a mobile network-based location oracle can provide suitable location information. Finally, we present several interaction strategies between the oracle and the smart contract and provide an algorithm that compares a provided location with the geofence.

A. Geofences in smart contracts

As illustrated in the related work, there are a multitude of options available to encode locations. To use and query geolocations in smart contracts, a hierarchical, grid encoding such as Geohashes or Google S2 cells are practical, because point in polygon can be performed quickly due to the hierarchical nature. To prepare a geofence for storage, we approximate it by encoding the covered area into grid shaped cells. S2 cells or Geohashes offer different cell sizes from one resolution level to the next. As shown in Figure 1, the process results in a set of identifiers, where each one describes a subset of the geofence. By increasing the number of cells used for the encoding, i.e. by modifying the minimum and maximum resolution level to be used, the approximation can be adjusted. Each cell identifier is stored in a key value map within a smart contract, where

[4]http://geohash.org/
[5]https://what3words.com/
[6]https://github.com/google/s2geometry

Fig. 2. Hierarchical S2 cell levels. Cell A can be reduced hierarchically to cell B and C. Highlighted is the token representation, that corresponds to Table I, in which the hierarchy can be seen in binary form.

TABLE I
EXAMPLE S2 CELLS AND RESOLUTION LEVELS

In Fig. 2	S2 cell binary	S2 cell level
A	0100 0111 1010 1000 0101 0001 1010 1100 0000 0000 0000 0000 0000 0000 0000 0000	13
B	0100 0111 1010 1000 0101 0001 1011 0000 0000 0000 0000 0000 0000 0000 0000 0000	12
C	0100 0111 1010 1000 0101 0001 1100 0000 0000 0000 0000 0000 0000 0000 0000 0000	11

each defined value is set to `True (1)`. The default value for undefined keys is to return `False (0)`.

In order to determine whether a given cell position is contained in the geofence, the key value map can be queried. If the exact cell has been used to define part of the geofence, a single lookup provides a positive answer. In the naive scenario that uses the same cell size for the entire geofence, such a single lookup would always suffice. However, as the geofence may be defined with varying cell resolutions in order to reduce the number of identifiers needed, it may also contain some large cells. As the provided cell location will be smaller, there may not be an immediate, direct match. Figure 2 illustrates the issue: assuming cell C is the geofence, a query for whether A is part of the geofence won't find a direct match in the key value map. To find a match, we increase the cell size by reducing its resolution level step by step until there is a match (cf. Table I), or the lowest geofence resolution has been exceeded.

B. The mobile network operator as a location oracle

To perform location tracking, the smart contract needs to be supplied with reliable updates of the location of a device. In our approach we rely on the existing infrastructure of a mobile network operator. An operator typically has a vast network of cell towers to offer widespread connectivity. At the same time, a terminal belonging to a user or an IoT device, must be locatable in terms of the network cell that provides it with connectivity. The network cell in turn has a geographical coverage area that indicates where the terminal must be located at. With the existing mobile network protocols, a mobile device can report its cell location with a location update. The network can also request the current cell by paging the terminal. Further positioning techniques have been introduced in the related work section II-B. Given the existing mobile network protocols and infrastructure, a mobile network operator can act as a location oracle. A shortcoming may be that an operator can only provide information about a terminals location when it is inside a serviced area. If the device is roaming, positioning may not be possible. To resolve the issue, multiple mobile network operators could collaborate to provide an oracle that covers multiple countries, and could therefore also be used for international transport tracking.

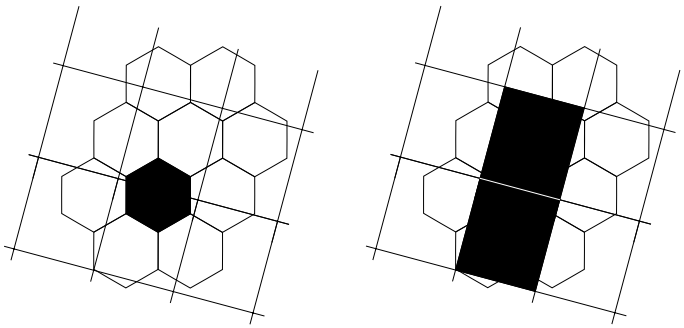

Fig. 3. A simplified mobile network cell displayed on the left, is encoded to be fully contained within two grid cells on the right.

C. Resolution alignment

To make cell coverage areas compatible with the geofence definition in the smart contract, again an encoding process needs to be applied. Here, a choice needs to be made at what resolution the area is encoded. As illustrated in Figure 3, the process can lead to a set of multiple grid cells. While sometimes significantly extending the coverage and thereby the position claim, the fact that it is only an approximation can be beneficial for network operators as they typically want to keep information about exact cell coverage areas private.

An important consideration is the alignment of encoding resolutions between geofence and network position estimate. The maximum grid cell size used in the position estimate, determines the smallest grid cell size of the geofence. As we will show in the evaluation, this has direct consequences accuracy of the geofence encoding.

D. Oracle-contract interaction strategies

The location oracle run by the mobile network operator is a regular blockchain participant that sends transactions to compatible smart contracts when set up accordingly. The oracle needs an initial configuration for the following:

- A mobile terminal that is to be tracked
- At least one smart contract address that is to be notified about the position of the mobile terminal

- The maximum cell level resolution of the geofence defined in the smart contract
- An interaction strategy describing when to submit new location information to the smart contract

The smallest resolution (maximum cell level) of the geofence needs to be defined upfront, as the oracle should not submit grid cells that are larger. If it were to submit these, it cannot be stated with certainty that the mobile terminal is within the geofence (cf. Figure 2). Turning to the interaction strategy, there are three approaches that can be taken. The terminal position can be submitted:

- *in regular intervals:* This has the benefit that a full position history will be created and can be traced at later points in time. If the terminal does not send location update messages to the network in periodic intervals, paging requests may be necessary that potentially create significant signaling overhead. At the same time, this approach may lead to a high number of blockchain transactions leading to increased storage costs.
- *on events triggered in the smart contract:* For example, when a shared car is returned, a smart contract may want to ensure that it is parked within a serviced area. During a conceivable deposit reclaim initiated by the driver, the location oracle could submit the location of the car.
- *on a distance moved:* To reduce the number of updates sent to the contract, the oracle could monitor the distance moved and only supply an updated position estimate if the terminal has moved a certain distance.
- *only on violation of the geofence:* To keep the number of transactions initiated by the oracle to a minimum, the oracle could monitor the terminal's position off-chain and only submit the location in case of a violation of the geofence.
- *on a combination of the above:* during the intial renting process a position update may be desirable, that is then followed by periodic updates.

E. Oracle position overlap with geofence

As visualized in Figure 3, the oracle provided position may consist of multiple grid cells. To determine whether the oracle-provided position overlaps with the geofence, each grid cell needs to be checked, potentially with multiple reduced cell levels. As long as one cell version is contained in the geofence, an overlap exists. The corresponding algorithm is listed in Algorithm 1. It would also be feasible to enforce full containment by requiring every cell to find a match in the geofence.

Examining the algorithm, we observe that the number of cells used in the geofence has no impact on the number of key value lookups that need to be performed.

If c is the number of grid cells to be checked, and l the number of levels the grid cell needs to be reduced by at most, then an overlap can be determined in at most $c \cdot l$ lookups. The lookup therefore only depends on the number of different resolution levels used in the geofence, and how many cells are supplied by the location oracle.

Fig. 4. An example car trace of mobile network position estimates (left), converted to S2 cell encodings (right).

F. Decentralized Car sharing

In contrast to the existing blockchain-based car sharing applications, a truly decentralized approach would allow users to set up their own smart contracts, so that they can specify their own requirements, without having to rely on a central contract or platform that can dictate terms and take comissions. To facilitate individual car sharing restrictions, the car owner can define a geofence for a city such as Berlin, perhaps including or removing certain regions. Further time and deposit constraints are conceivable. A potential renter would be able to search for all smart contracts that offer to share a car, browsing their current locations and renting conditions. Ideally, the smart contract supports utility functionality like inspecting the geofence, and renting it - whereby a deposit may be stored in the contract. To verify that the location oracle works properly, the user could request the cars position, confirming the result in the real world. During driving, the terminal built into the car enables periodic location updates via the mobile network, that are sent to the smart contract. If the user leaves the geofence, the deposit is retained.

Figure 4 illustrates a car trace obtained from simplified mobile network cell positions. In this scenario, a gap exists in the middle of the map that is not due to missing information. Instead, if a periodic location update has been chosen, the car may have passed several network cells before the corresponding smart contract is notified about a new position.

Algorithm 1: hasOverlap(locationList)

```
Predefined: geofenceMap, minLevel of geofenceMap
Input     : Key-value map of the geofence,
            list of geolocations to be checked
Output    : Boolean indicating whether the locationList
            is at least partially inside the geofence
for cell in locationList do
    while level(cell) ≤ minLevel do
        if cell in geofenceMap then
            return True
        else
            cell = reduceLevel(cell)
        end
    end
end
return False
```

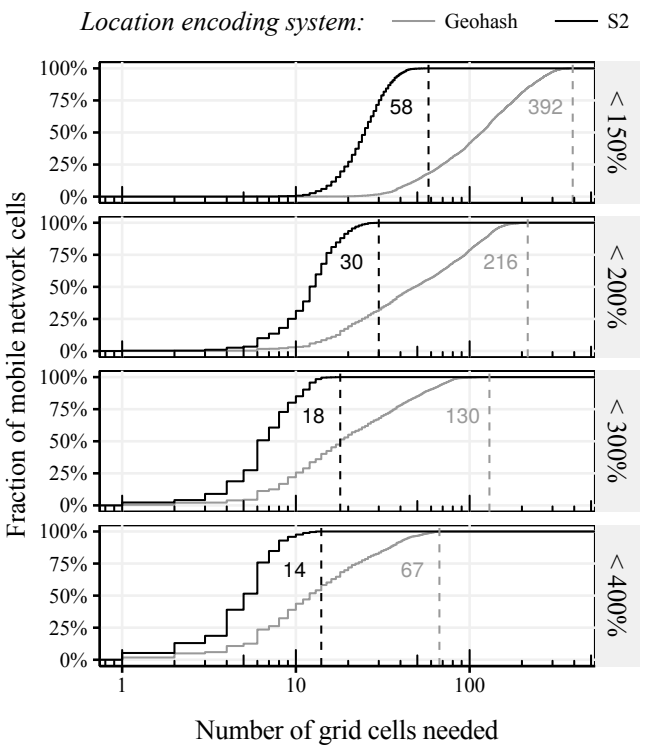

Fig. 5. Cumulative distribution of grid cells needed to model a coverage of less than the number indicated on the right. The vertical dashed lines indicate at which point 100% are reached. For example, the second plot indicates that at most 30 S2 grid cells are needed to encode any mobile network cell with less than twice (200%) of its original coverage area. For the same coverage constraint, 10 S2 grid cells are sufficient to encode 25% of the cells.

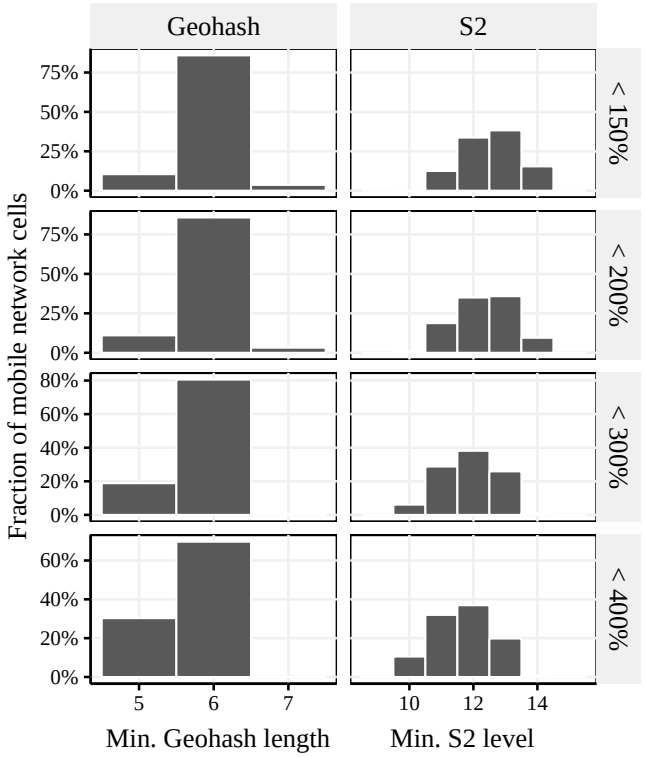

Fig. 6. Distribution of shortest Geohash lengths and minimum S2 levels for each cell tower area encoding, over all encodings. For example, roughly 80% of the Geohash encodings with a coverage of $< 150\%$ are of length 6 or longer. For the same coverage, about 10% of S2 encodings are of level 11 or higher.

IV. Evaluation

To evaluate the proposed solution, we compare the Geohash and S2 location encoding systems for modelling mobile network cell coverage areas and city-wide geofences. The aim is to encode areas with as few grid cells as possible, as they will lead to one-time storage and transaction costs for the blockchain. Only the initial setup can be costly, later data access is comparativly cheap. We identify the optimal grid cell sizes to configure a smart contract accordingly. Secondly, we measure actual execution costs based on an implementation for the EVM, which can also be translated to dollar costs based on current pricing circumstances of the public Ethereum blockchain.

A. Cell tower coverage area encoding

We model cell tower coverage areas obtained with an Android application [19], that collects cell tower information and resolves it using Google APIs. The location estimates consist of a latitude, longitude and an horizontal accuracy. The Android developer documentation defines this accuracy as the radius of 68% confidence[7]. In other words, there is a 68% likelihood that the device within a circle defined by the location and the radius. In reality such estimates aren't likely to be circles, but their accuracy will have the same order of magnitude.

We collected 4,505 of these cell towers position estimates in Germany, that have radius values ranging from 500m to 5000m. Figure 7 displays the distribution. The majority of mobile network cells has a horizontal accuracy between 500m and 2000m.

We encode all cell tower coverage areas using S2 and Geohash grid cells, while permitting a range of minimum and maximum Geohash lengths, and S2 levels. For each result we count how many grid cells are needed to represent the given polygon. We measure the coverage, that describes how much of the original polygon area has been covered. For example, a coverage of 130% implies an additional 30% of excess area is covered, that does not belong to the original polygon. The best possible value would be 100%.

Figure 5 illustrates the results. For four different coverage contstraints, a cumulative distribution of mobile network cells indicates how many Geohash or S2 grid cells are needed to encode them. By relaxing the coverage constraint, e.g., accepting a coverage of $< 400\%$, the number of grid cells

[7]https://developer.android.com/reference/android/location/Location

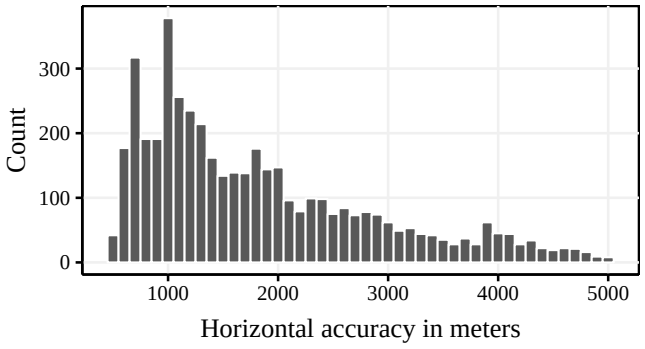

Fig. 7. Distribution of horizontal accuracy values (radius) obtained for cell tower position estimates from the Google location API.

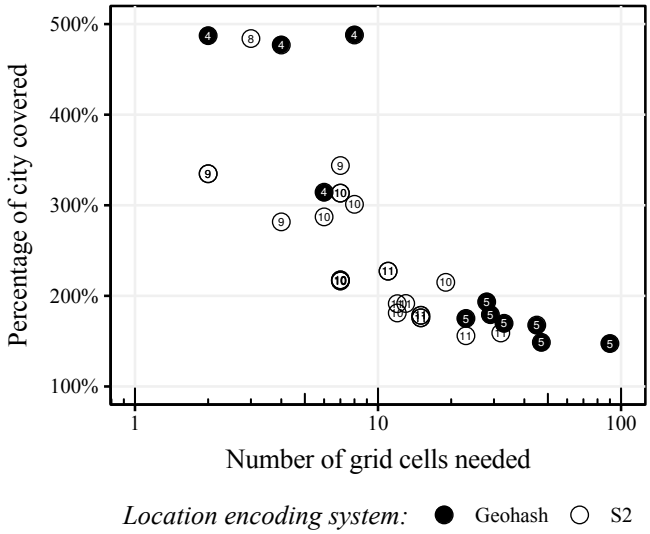

Fig. 8. Each circle represents an encoding of one of the top 7 most populated cities in Germany, where the number displays the the max. cell resolution. Indicated are the number of grid cells needed (Geohash and S2 encodings) to achieve a certain coverage (closer to 100% is better). A constraint is applied that the maximum Geohash length be 5 and the maximum S2 level be 11.

needed decreases. In all constraint scenarios, the S2 encodings perform better than Geohashes. This is likely due the more limited set of grid cells resolutions available for Geohashes. For example, reducing a length 6 Geohash to length 5, increases its area by a factor of 32, whereas reducing an S2 level 12 to level 11 leads to an area increase by a factor of 4.

Corresponding with Figure 5, Figure 6 shows the distribution of shortest Geohash lengths and minimum S2 levels (largest areas) over all encodings. We can observe that in S2 encodings, four different levels emerge, whereas Geohashes mostly consist of lengths 5 and 6 as the shortest lengths seen. By relaxing the coverage constraints, the distribution shifts towards shorter Geohashes and smaller S2 levels, meaning larger grid cells are observed more frequently.

For both coverage constraints $< 150\%$ and $< 200\%$, the maximum grid cell areas (Geohash length 5, S2 level 11) correspond to about 5×5km and 3×5km. This determines the choice for the lowest grid cell size that can be used for the geofences, as a larger oracle-provided grid cell cannot be inside a smaller geofence grid cell.

B. City geofence encoding

Following the results of the previous section, we continue by encoding the 7 largest cities by population in Germany: Berlin, Hamburg, Munich, Cologne, Frankfurt am Main, Stuttgart and Düsseldorf. We restrict the encoding to use at most a Geohash length of 5, and S2 level 11, to maintain compatibility with the mobile network cell area encoding. The results are shown in Figure 8. We observe that a Geohash encoding achieves a similar coverage compared to S2 encodings, while requiring more grid cells. Due to the minimum grid cell size constraint, no encoding can achieve a coverage below 150%. For example, using a maximum S2 level of 11 to cover Berlin leads to the best score of 160% coverage when using 30 S2 cells.

C. EVM gas costs

To determine smart contract execution costs, we implemented the system for the EVM, based on the S2 location encoding scheme. We measured the gas costs for storing geofences and the location evaluation algorithm introduced earlier. We observe that the gas costs for storing a geofence increase linearly with the number of grid cells defined. As a single mapping entry requires an `SSTORE` instruction, the cost for each grid cell corresponds to 20,000 gas. Combined with base transaction costs and some overhead costs for looping, a geofence consisting of 100 grid cells leads to a gas cost of 2,088,102 gas. As of August 16th, 2018, this would translate to $1.89 on the main Ethereum blockchain.

An oracle submitted location that is evaluated according to Algorithm 1, leads to multiple storage access instructions (`SLOAD`), that cost 200 gas each. Therefore, a transaction specifying a grid cell location that does not need to be reduced costs only 22,442 gas, of which 21,000 are the base transaction costs of the transaction. Additional reduction steps for the S2 implementation cost roughly 1,500 additional gas per step. Assuming the oracle supplies 15 positions, that each may have to be reduced by up to 3 levels, the maximum gas cost will be 88,500. For the main Ethereum blockchain, this comes down to at most $0.08 per update.

Depending on the strategy employed for providing the smart contract with location updates, these costs will add up. If 100 updates are sent to the smart contract, the corresponding cost would be $8 at current prices, but could be much higher when the public network is under heavy load. In general, the proposed solution can be implemented not only on other EVM-compatible blockchains, but also different smart contract platforms such as EOS[8] or NEO[9], where costs may differ.

[8]https://eos.io/
[9]https://neo.org/

V. Discussion

In a real world scenario, a mobile network operator can make use of multiple position estimation techniques that surpass the accuracy of what Google cell tower position estimates provide. If the accuracy is increased, the resolution of the grid cells can be improved, allowing for more accurate position claims and geofences. Transaction costs will slightly increase, as additional resolution levels need to be checked.

While our approach is also feasible on other smart-contract enabled blockchain platforms, we only implemented it for the Ethereum Virtual Machine. Transaction costs may vary depending on the platform used, and the real world costs for the main Ethereum blockchain could be much higher if such a system would be used on a larger scale on the main network. It may also be a good idea to use a private blockchain for this specific use case.

As transparency and traceability are inherent properties of most of todays blockchains that support smart contracts, location privacy is not maintained with our approach. Although the position updates would be coarse and not very specific, movements could be extracted from the blockchain to deanonymize users.

For the proposed use cases, a blockchain identity needs to be linked to a sim card. Therefore, a blockchain address is then associated to an identity in the real world. For scenarios that need location information that does not change frequently, it can be argued that an identity system is sufficient. For example, in supply chains, a factory location may be implicitly given, if the identity of the signing entity is known. This is because we know that the identity is stationary in the known location. Similarly, for use cases that depend on city-level location information such as residence, the information is implicitly known if they rarely change and are attached to the identity.

VI. Future work

Scalability efforts are an ongoing aspect in blockchain research. Our proposed solution could perhaps benefit from off-chain scaling solutions such as state channels. The state channel could be maintained while the geofence isn't violated, only to be settled on the main chain on violation.

As part of the question for incentives, economic aspects and threat modelling for blockchain-based proof of location systems are an open topic.

In order to improve privacy aspects, it may be of interest to determine whether zero knowledge proofs can be combined with geofences in smart contracts. For example, being able to prove that the terminal's current position is among the permitted positions (set membership), without revealing its true position could be of great benefit.

Finally, a location oracle does not need to be limited to providing position estimates. Depending on the location of the network cells, it may also be possible to provided enhanced information. For example if a terminal is moving between cells that only cover subway tunnels, the oracle may provide information on the mode of transport that is used.

VII. Conclusion

We have proposed an approach to enable LBS for smart contracts. It relies on a mobile network operator to act as an oracle, submitting location estimates of a terminal to a smart contract. The position can be evaluated to be inside or outside of a smart contract-defined geofence. We have evaluated Geohashes and S2 cells for their suitability to model position estimates and city wide geofences, with the finding that S2 cells perform better. After estimating transaction costs for the EVM, we conclude that LBS for smart contracts are not only feasible, but comparatively cheap for position evaluation.

References

[1] J. Hamari, M. Sjöklint, and A. Ukkonen, "The sharing economy: Why people participate in collaborative consumption," *Journal of the association for information science and technology*, vol. 67, no. 9, pp. 2047–2059, 2016.

[2] M. Cohen and A. Sundararajan, "Self-regulation and innovation in the peer-to-peer sharing economy," *U. Chi. L. Rev. Dialogue*, vol. 82, p. 116, 2015.

[3] S. P. Fraiberger and A. Sundararajan, "Peer-to-peer rental markets in the sharing economy," *NYU Stern School of Business Research Paper*, 2017, available at SSRN: https://ssrn.com/abstract=2574337.

[4] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun, "On the requirements for successful gps spoofing attacks," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 75–86.

[5] B. Nissen, L. Pschetz, D. Murray-Rust, H. Mehrpouya, S. Oosthuizen, and C. Speed, "Geocoin: Supporting ideation and collaborative design with smart contracts," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 2018, p. 163.

[6] N. Szabo, "The idea of smart contracts," *Nick Szabos Papers and Concise Tutorials*, vol. 6, 1997.

[7] V. Buterin, "A next-generation smart contract and decentralized application platform," *Ethereum Project White Paper*, 2014.

[8] A. Greenwald, G. Hampel, C. Phadke, and V. Poosala, "An economically viable solution to geofencing for mass-market applications," *Bell Labs Technical Journal*, vol. 16, no. 2, pp. 21–38, 2011.

[9] A. Küpper, U. Bareth, and B. Freese, "Geofencing and background tracking–the next features in lbss," in *Proceedings of the 41th Annual Conference of the Gesellschaft für Informatik eV*, 2011.

[10] S. Rodriguez Garzon and B. Deva, "Geofencing 2.0: taking location-based notifications to the next level," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2014, pp. 921–932.

[11] A. Küpper, "Location-based services: Fundamentals and applications," *John Willey & Sons, ISBN: 0-470-09231-9*, 2005.

[12] F. Gustafsson and F. Gunnarsson, "Mobile positioning using wireless networks: possibilities and fundamental limitations based on available wireless network measurements," *IEEE Signal processing magazine*, vol. 22, no. 4, pp. 41–53, 2005.

[13] S. Frattasi and F. Della Rosa, *Mobile positioning and tracking: from conventional to cooperative techniques*. John Wiley & Sons, 2017.

[14] H. Wymeersch, G. Seco-Granados, G. Destino, D. Dardari, and F. Tufvesson, "5g mmwave positioning for vehicular networks," *IEEE Wireless Communications*, vol. 24, no. 6, pp. 80–86, 2017.

[15] G. Brambilla, M. Amoretti, and F. Zanichelli, "Using blockchain for peer-to-peer proof-of-location," *arXiv preprint arXiv:1607.00174*, 2016.

[16] T. Dasu, Y. Kanza, and D. Srivastava, "Unchain your blockchain," in *Proc. Symposium on Foundations and Applications of Blockchain*, vol. 1, 2018, pp. 16–23.

[17] O. Survey, "A guide to coordinate systems in great britain: An introduction to mapping coordinate systems and the use of gps datasets with ordnance survey mapping," *Technical Rep. v2. 2*, 2013.

[18] S. Xu and K. Kamath, "Dynamic geohash-based geofencing," Aug. 23 2016, uS Patent 9,426,620.

[19] F. Victor, S. R. Garzon, and A. Küpper, "Smartphone-collected mobile network events for mobility modeling," in *Proceedings of the 14th IEEE International Conference on Ubiquitous Intelligence and Computing*. IEEE, 2017, pp. 871–878.