Distributed Privacy Preserving Platform for Ridesharing Services

Yevhenii Semenko, Damien Saucez

Distributed Privacy Preserving Platform for Ridesharing Services

Yevhenii Semenko and Damien Saucez

Université Côte d'Azur, Inria, France

Abstract. The sharing economy fundamentally changed business and social interactions. Interestingly, while in essence this form of collaborative economy allows people to directly interact with each other, it is also at the source of the advent of eminently centralized platforms and marketplaces, such as Uber and Airbnb. One may be concerned with the risk of giving the control of a market to a handful of actors that may unilaterally fix their own rules and threaten privacy. In this paper, we propose a decentralized ridesharing architecture which gives the opportunity to shift from centralized platforms to decentralized ones. Digital communications in our proposition are specifically designed to preserve data privacy and avoid any form of centralization. We integrate a blockchain in our proposition to guarantee the essential roles of a marketplace, but in a decentralized way. Our numerical evaluation quantifies the advantages and limits of decentralization and our Android implementation shows the feasibility of our proposition.

Keywords: Data Privacy; Decentralized; Sharing Economy; Blockchain; Peer-to-Peer

1 Introduction

Mass digitalization of our daily life and deep involvement of user data boosted the sharing economy by allowing the advent of massively used platforms, such as Airbnb and Uber, that make our day-to-day tasks much easier. However, such platforms raise concerns about the concentration of power and information in a handful of mercantile companies with the risk of inappropriate personal data usage [7], personal information leakage [16], or market control [9].

This paper proposes to tackle these issues for the case of ridesharing services by proposing a multilayered distributed architecture that removes the need of a centralize platform. The shift to a fully decentralized paradigm is not straightforward. First, with a centralized approach, personal data are concentrated and, unless security breaches or inadequate policies, the data can only be manipulated by well identified entities (i.e., the platform operator). In a fully decentralized entity data are scattered among the constituents of the decentralized entity platform with inherently more risks of information leakage. Second, decisions to authorize or not a given client or producer to be part of the platform or to define

a price are more complex in a decentralized platform as the system as a whole must answer these questions. Finally, responsiveness of the system can become a challenge as it is impossible to tune the resources as well as with centralized platforms.

Ridesharing services are particularly challenging as they are by nature highly dynamic (clients come and leave frequently), they involve users willing to minimize data communication as they are roaming, they require to exchange sensitive personal information, and they are subject to rules that can change from region to region while users can move between these regions.

Privacy preserving solutions for ridesharing have been proposed ([1–3, 6, 8, 10, 14]) but we are the first to take a holistic approach were we consider simultaneously privacy, accountability, business, communications, and scalability.

In Sec. 2 we define a decentralize privacy preserving platform for ridesharing where peers forms an overlay communication network and rely on a blockchain to organize their marketplace. In Sec. 3 we evaluate its scalability based on analytical bounds and on a trace-driven numerical evaluation. Sec. 4 discusses our Android prototype and the lesson learned while developping it. Finally, in Sec. 5 we conclude.

2 Decentralized Platform

Ridesharing platforms such as Uber or Lyft rely on centralized infrastructures involving third-parties, which raises privacy concerns. In this paper, we remove this dependency to avoid any lock-in with a third-party and to preserve privacy. To that aim, we conceive the platform as a fully decentralized peer-to-peer network [11] where each node is equally important and interchangeable and rely on a multilayer architecture.

2.1 Multilayered architecture

First, we build an overlay network including all the business actors to constitute the *network layer*. As ridesharing services can span over multiple regions with different constraints and can have several millions of users, the underlying peer-to-peer networking technology must provide a way to segregate traffic in arbitrary regions and it must be able to scale. For these reasons we must use a protocol where node *IDentifiers* (ID) can be structured and scoped, such as in Chord [18] or in Kademlia [12]. It is therefore straightforward to limit the scope of communications to particular regions. For example, if communications must remain within peers of a country, it is enough to partition the ID space with one partition per country and impose that every peer of a country uses an ID within the partition of the country. Fig. 1 shows an example of how to scope communications per region with Chord.

Second, as the platform does not involve a data storage third party, all actors involved in the platform must take part in the storage effort. Keeping in mind that data privacy is essential, the best approach is to build a Distributed Hash

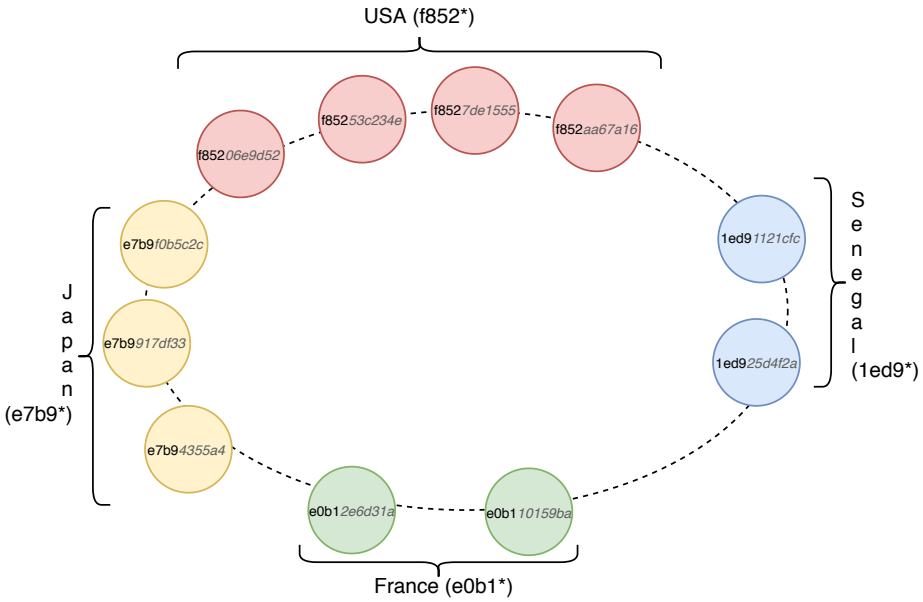

Fig. 1. Example of device grouping with Chord.

Table (DHT) to implement the *Data storage layer*. Even though the data storage layer by itself is not sufficient to guarantee data privacy, when it is combined with the other layers it enables data privacy as detailed in Sec. 2.2.

User Interface (UI) and user experience can't be altered by the decentralization even though the *User application layer* where the UI is implemented fundamentally changes. In centralized platforms, user applications interact with the rest of the platform via REST APIs. In a decentralized solution, the application must include the other layers of the platform and implement, at least partially, the logic of the platform.

Finally, the *Management layer* that is simple by nature in a centralized approach is complex to implement in a fully distributed system with no trusted parties. Particularly complex issues to solve in such an environment are how to control business processes, manage data access, and provide accountability to solve potential disputable situations. These issues arise as there is no central to solve debatable issues, to control or to set the required system parameters.

To implement the *Management layer* we use a blockchain [19] which plays the role of an intermediary for controlling data access, guarantee that parties act in conformance with the rules of the platform, and log events. The business logic is implemented with smart contracts and privacy is ensured.

Alone blockchains do not provide data privacy protection. Instead they tend to reduce privacy as they spread data to open locations. However, thanks to smart contracts data owners can specify who and how their data can be ac-

cessed. In Sec. 2.2 we detail how to guarantee data privacy with our multilayered distributed platform.

2.2 Data Privacy

The first condition to preserve privacy is to minimize the amount of data that is spread in the system. That is, for each data, the designer of the service must determine whether or not the data must be shared, who can access it, and under which conditions. Consumers of the data are either identified at the time of the publication or they are not.

The first case is the simplest, the data is encrypted with the public key of the target such that only the target can read the data. In this case, the data can be stored anywhere in the DHT, multiple targets means multiple copies of the data. As our architecture does not rely on trusted third parties, there is no trusted *Public Key Infrastructure* (PKI) on which we can rely to obtain the public key. However, if like in Bitcoin the ID is made such that it is unambiguously linked with the public key of the party then it is possible to safely retrieve the public key for that node [13]: 1) the producer of data determines the ID of its target, 2) it requests the system (or the target directly) to provide its public key, and 3) it verifies that the ID and the public key are linked (e.g., the ID contains a hash of the public key).

In the second case where the target of the data is not known while publishing the data, it is not possible to use encryption directly. Instead, the producer uses the Shamir's Threshold Scheme [17]. The producer generates as many parts of its secret as needed and determines a DHT key for each part such that they can be published in the DHT according to its requirements. Here the choice of the key for each part to be published in the DHT is important as it determines the nodes that will store the data. In parallel to publishing the data in the DHT, the producer also publishes a smart contract in the blockchain that specifies the policy of accessing the data. As a result, when a party wants to consume a data, it sends a request to the overlay and if the smart contract is fulfilled then the storing nodes provide their parts to the requester that will be able to re-construct the data. The threshold value and where to store the parts (and their redundancy level) depends on the service's threat model.

2.3 Protocol

Road transportation clients and drivers are by nature on the move and thus have poor, expensive, and unreliable connectivity. For that reason they do not directly take place in the peer-to-peer network. Instead the peer-to-peer network is run by arbitrary machines willing to share their resources for the community.The nodes taking part in the peer-to-peer network then act as service nodes to access the network for the clients and drivers by the means of an HTTPS REST API.

Client and driver applications select a node from the peer-to-peer network as service node according to their own preferences (e.g., randomly or topologically close to them, or a trusted service node).

Any ride can be decomposed in three phases. First, the client hails for a ride. During that phase, the client and a driver mutually agree on making the ride together. Second, the ride starts and the client is dropped-off at its desired location. Finally, the client and driver finish the transaction by paying and they comment the service if they want.

Below we detail the protocol that we designed to implement these three phases in our distributed platform. Fig. 2 and Fig. 3 depict the exchanges and the messages, respectively.

Requesting phase When a client wants to ride from point A to point B, it sends a `Request` message to its service node. The message contains a nonce, a timestamp, and information on the desired types of car. The nonce is made to avoid leaking the identity of the client in the network and is the HMAC of a random number generated by the client. The information about the ride are the anonymized origin and destination location of the ride, the exact distance for the ride as computed with a traffic route planner API, the price per kilometer that the client is ready to pay, and the categories of cars the client is willing to use. The client does not provide its actual origin and destination coordinates to avoid leaking sensitive information to the system. Instead it provides approximate locations. The price per kilometer is computed according to an auction mechanism. The location anonymization and price computation mechanisms are out of the scope of this paper.

When a service node receives a `Request`, it broadcasts it in the peer-to-peer network and tags the message with its own IP address and port number. In the meanwhile, the service node locally stores the request in its *pending ride requests queue*. When a node in the peer-to-peer network receives a broadcasted `Request` message, it locally stores it in its pending ride requests queue.

Every driver periodically polls its service node with a `Ping` message to know the pending ride requests. The `Ping` message contains the anonymized location of the driver, its type of car, and a timestamp. If the driver is in a location close to a request in the pending request queue of the service node, the service node replies to the driver by forwarding it the oldest compatible "close enough" `Request` message in its pending ride requests queue. If the driver doesn't want to accept this ride, it silently ignores it. Otherwise, it sends an `Acknowledgement` message directly to the service node of the client, without going through the peer-to-peer network as it knows the IP and port that can be used for direct communications with the service node. It is worth to notice that at this stage the driver leaked potential private information as it publicly reveals its ID, current approximate location, and will to achieve a ride that is uniquely identified by a nonce. Unfortunately, the only solution to avoid this would be to have a mechanism to encrypt the acknowledgment such that only the client can decrypt it. But as we are in an untrusted environment, that would imply to leak the client ID as drivers would have to verify that the request was not forged by a service node. In the meanwhile, only the ID and an approximate location is leaked, which is not a severe threat for drivers as they are anyway clearly publicly identified in

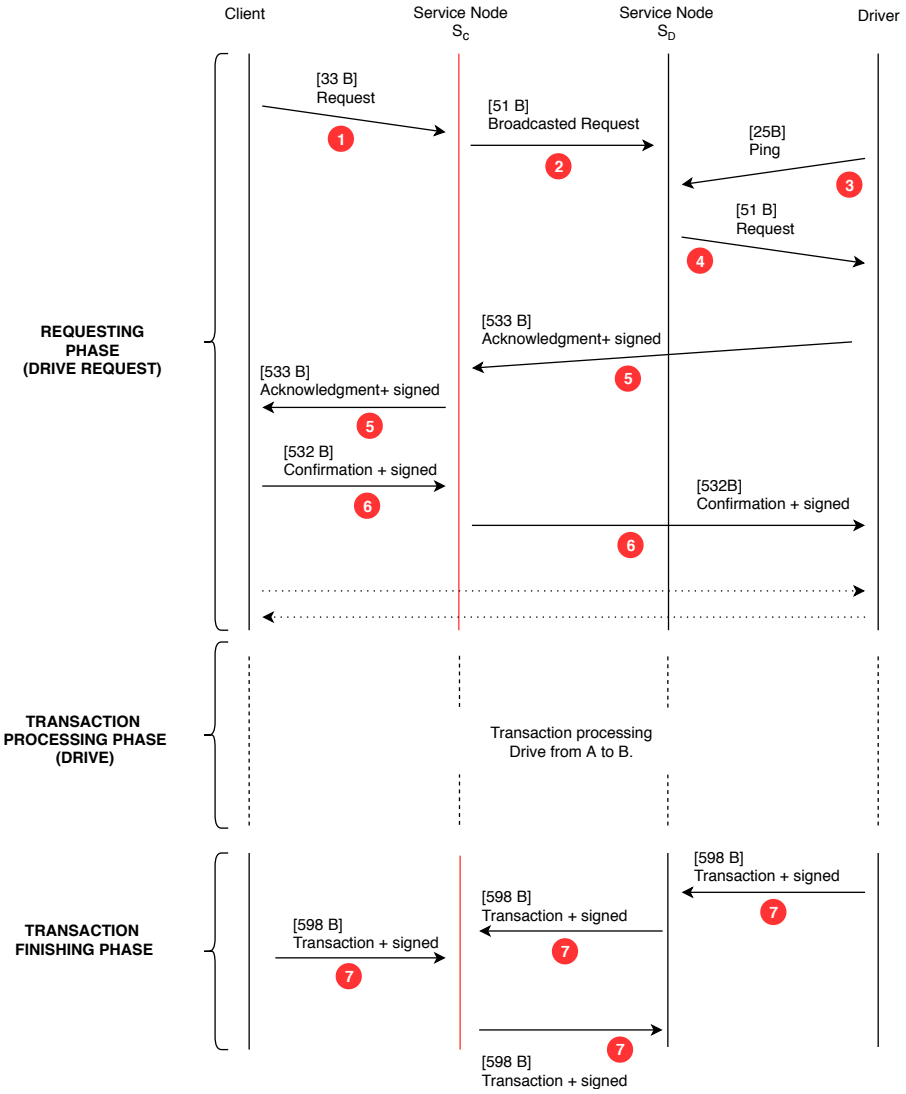

Fig. 2. Ridesharing service protocol workflow.

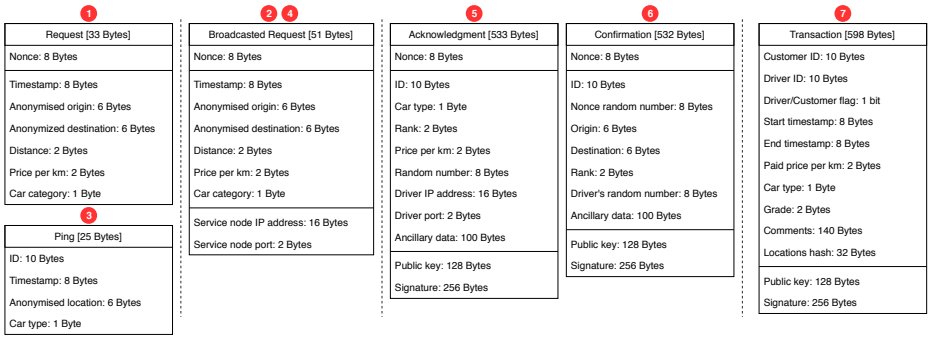

Fig. 3. Ridesharing service messages.

real life for legal reasons (e.g., with a specific sticker on their cab), while it could be a sever threat for clients. This is the reason while we designed our protocol in that way.

The `Acknowledgement` message contains the nonce taken from the `Request`, the driver's ID and public key, he's car type, his rank, a price per kilometer, a random number, and ancillary data that depend on the local regulations and habits (e.g., driver name, car make and model...). In addition, it provides the IP address and port number to use to have a direct communication with it.[1] The message is signed by the driver. When the client service node receives this message, it forwards the acknowledgment to the client and removes the request from its pending requests queue such that any further `Acknowledgement` message will be silently dropped.

When a client receives an `Acknowledgement` message, he decides whether or not he agrees to make the ride with that particular driver (e.g., based on the driver's rank). If he doesn't accept the driver proposition, he silently ignores the proposition in order to avoid leaking personal information. If he accepts the proposition, he sends a `Confirmation` message directly to the driver's device (for which he knows the IP address and port number to use). The message contains the nonce and the random number used by the client to generate the nonce, the client's ID, public key, and rank, the exact origin and destination points of the ride, the random number sent by the driver, and ancillary data that depend on the local regulations and habits (e.g., client's surname). The message is signed by the client and is fully encrypted with the public key of the driver. With the nonce and its associated random number, the client's public key, and the signature, the driver can verify that the client acknowledgment is generated by the owner of the ID that initiated the `Request` message and with the echo of the random number that it provided in the `Acknowledgment` message, this prevents replay attacks.

[1] We assume that drivers can set in place hole-punching mechanisms, e.g., via their home box, to allow direct connection to their device while working.

At this stage, both the client and the driver have fully disclosed their personal information, but this information is known only by them. The client and driver start exchanging their actual GPS coordinates until the client is picked up. To minimize the number of messages exchanged they rely on a traffic route planner API and only exchange information if they deviate from the plan. These communications are encrypted and directly sent between the client and the driver.

It is worth to mention that even though the client's service node doesn't know the ID or exact coordinates of the client, it knows its IP address (as it has direct communication with it) and knows the ID of the driver. There are no practical solutions to avoid such leakage of information but the client can tackle this issue by using a trusted service node.

Driving phase The drive starts when the client is picked up by the driver. At this time, devices can collect GPS exchange format (GPX) data independently from each other. During the drive there are no messages exchanged between the devices or with the rest of the network. Due to this, a third party will never be able to determine the time of making the drive or intercept any data concerning the transaction participants GPS positions. This information is not published anywhere and is for strict personal usage. It can, for example, be used during a litigation where parties can compare their traces to justify their disagreement.

Cloture phase When the ride is finished, participants give a grade (e.g., stars) and a comment on the drive. The driver and the client independently advertise their ride to the network with a `Transaction` message. This message contains the ID of the client and of the driver, a start and an end timestamp, a flag indicating whether the message is from a client or a driver, the proposed grade, a comment, the price paid per kilometer, and a hash of the concatenation of the origin and destination of the ride. The message is signed by the emitter of the message (i.e., the driver or the client) and contains its public key.

The only personal information that can be inferred from a `Transaction` message is that a given client did a ride with a given driver at a given time. Other information, such as location, are never disclosed.

Transaction Data are eventually validated in the blockchain. Each node in the blockchain checks the values of both transaction data. If both are similar and correct, i.e., timestamps are reasonably close in both transaction messages, the hash of the locations is the same, and the price per kilometer is the same, then they are considered as validated transactions and added to the new next block. In the case transactions cannot be validated (e.g., only one of the two has been received, or the transactions are incompatible), they are still eventually added to the blockchain but they are marked as being in conflict.

Periodically, a *summary block* is published in the blockchain. The new summary block contains the list of observed IDs seen since the last summary block and their new rank. The rank is computed based on the last published rank for this ID and the grades that appear in the transactions with this ID. The function used to re-compute the rank must be robust to malicious users.

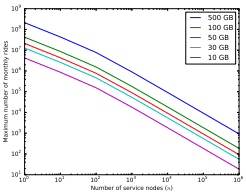

Fig. 4. Total maximum number of monthly rides supported by the platform, given a maximum monthly data volume budget per service node.

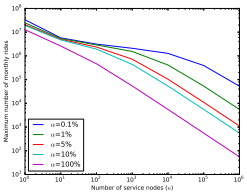

Fig. 5. Total maximum number of monthly rides supported by the platform when service nodes are grouped per zone for a maximum monthly data volume budget of 30 GB per service node.

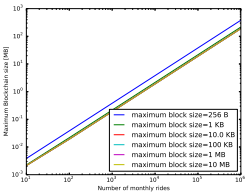

Fig. 6. Maximum increase of blockchain size per month as a function of the number of monthly rides.

3 Evaluation

In the following we evaluate the scalability of our proposition when it is implemented with Chord. We first perform an analytical study that we comment with data coming from on a real ride trace.

3.1 Platform dimensioning

Traffic load With the protocol described above, we can precisely determine the data traffic generated by each ride for a network composed of n service nodes. We compute the maximum amount of traffic for the worst case scenario (i.e., using IPv6). To determine the amount of traffic supported by service nodes for one ride, we have to identify the three different types of service nodes.

(i) The *client's service node* receives the client's `Request` that it broadcasts to its neighbors (i.e., $log_2(n)$ service nodes). It also receives the driver's `Acknowledgment` messages from the d drivers willing to take the ride and forwards the first one to the client. At the end of a ride, it receives the `Transaction` message from the its client and forwards it to its neighbors (i.e., $log_2(n)$ service nodes) but also the one from the driver that it also broadcasts.

(ii) The *driver's service node* receives one broadcasted `Request` message and forwards it to its neighbors (i.e., $log_2(n)$ service nodes) and the driver's `Ping` message to which it answers by forwarding the broadcasted `Request` message. Similar to the client's node, this node will receive and broadcast transactions.

(iii) The *relay service nodes* have neither the client nor a driver. Such relay nodes receive one broadcasted `Request` message and two `Transaction` messages that they have to broadcast it to their neighbors (i.e., $log_2(n)$ nodes).

The worst case is if the client requests are always issued from clients connected to the same client's service node and when one driver of each service node (including the client's service node) answers to the request.

Fig. 4 shows the evolution of the maximum number of rides that the whole platform can handle every month for a given monthly data volume quota. The x-axis gives the number of service nodes and the y-axis gives the maximum number of rides per month. Both axes are in log-scale. As shown in Fig. 4, the maximum number of rides that can be handled by the system exponentially decreases with the number of service nodes because every request has to be broadcasted in the network. For example, for 10,000 service nodes having a budget of 30 GB per month, the platform will handle at least 5,524 rides per month. This is three orders of magnitude lower than the number of requests currently seen by Uber or Lyft in a city like San Francisco [5]. Nevertheless, it is important to remember that this value represents the minimum number of rides that the platform can support. Based on a trace study in Sec. 3.2 we will see that in practice our platform can handle much more rides, and be used for current taxi services.

To alleviate this scalability issue, we can leverage the possibility to limit the scope of the messages in our Chord network thanks to a clever construction of IDs and identify two types of communications. On the one hand the `Transaction` messages must be broadcasted to all the service nodes in order to be treated safely by the blockchain. All the other messages can remain in the scope of the region where the number of service nodes can be small (e.g., a dozen) without impairing the security of the system if taken randomly in a large pool of supposedly independent nodes. If α is the fraction of service nodes that compose a region the traffic volume at a service node is in $\mathcal{O}(\alpha \cdot n + log(n))$ instead of $\mathcal{O}(n + log(n))$ which substantially decreases the service node load for small values of α. Similarly to Fig. 4, Fig. 5 shows the maximum number of rides that the platform can support for various α and a monthly budget of 30 GB. We can see an important improvement as regions decrease in size. For example, with 10,000 nodes, if each region is limited to 10 service nodes (i.e., $\alpha = 0.1\%$), the platform will support at least 1,275,197 rides per month, which is of the same order of magnitude as what ride sharing services are experiencing today in large cities like San Francisco [5].

Blockchain bloat Blockchains are strictly growing structures and we can compute the worst case increase of the blockchain size as that is directly linked to the number of rides supported by the platform: for every ride, at most two transactions will be published in a block of the blockchain. Each transactions in a block contains the IDs of the driver and client, a flag to indicate if the emitter was the driver or the client, a rank, the price per kilometer and the pointer to the comment stored in the DHT. A block contains a list of transactions, a timestamp, the hash of the block, and the hash of its previous block. Summary blocks are also created with the list of IDs and their rank for IDs with a modified rank since the emission of the last summary block. In the worst case, if the IDs are observed only once between two summary blocks and if their rank changed, the summary block will contain all the IDs.

Fig. 6 shows the maximum monthly increase of the blockchain size as a function of the number of monthly rides. Both axes are in log scale. As long as

the maximum block size allows to store a large enough number of transactions, the size of the block has no particular impact on the blockchain size. On the contrary, when the block size is too small, it cannot contain many transactions and the block overhead (i.e., its timestamp, hash value, and predecessor pointer) is not negligible anymore. Nevertheless, for reasonable block size, increasing the maximum block size only marginally influences the block chain size but is makes it less reactive as the time span between two block creations increases, a linear function of the transaction rate.

Fig. 6 shows that for realistic parameters of the system, the monthly increase of blockchain size is reasonable. For example, for a monthly budget of 30 GB, $\alpha = 0.1\%$, and 10,000 service nodes, the number of rides would be 1,275,197. With a maximum block size of 100 KB, a new block would be created about every 20.57 minutes and a new summary block about every 2.48 hours for a total of 241.10 MB. If the system can be less reactive, the maximum block size can be increased to 1 MB. In this case, blocks are created every 3.50 hours and summary blocks are created every 1.03 days and the blockchain size monthly increase is 240.84 MB.

3.2 San-Francisco Taxi trace evaluation

In Sec. 3.1 we show the scalability limits of our distributed platform in the worst possible case. In this section, we re-evaluate the scalability of our platform in a realistic scenario. To that aim, we use the San Francisco taxi cabs mobility trace from Piorkowski et al. [15]. This dataset tracked GPS coordinates and status of 536 taxi cabs for one month in the Bay area, between May 17^{th} 2008 and June 9^{th} 2008. 2008. In total, this data set logs 437,377 rides over a month.

If the platform is composed of 10,000 service nodes, in the worst case where all available drivers acknowledge ride requests and $\alpha = 1$, the maximum monthly traffic at a service node would only be 45.96 GB while the theoretical worst case is higher than 500 GB for the same amount of rides, as depicted by Fig. 4. These results show that despite in theory our distributed platform does not scale as well as a centralized platform, in practice the distributed scheme is perfectly usable. The reason is that in practice the offer and the demand are aligned and only few resources remain unused. In practice, $d \ll n$ for large enough number of service nodes.

Finally, the blockchain size as computed from the trace, shows that in practice the blockchain bloat is limited. For instance, with maximum block size values between 1 KB and 10 MB, the total blockhain size to store informations from the trace stays between 78 MB and 79 MB.

4 Proof of concept implementation

The architecture and protocol presented above have been mostly driven by our prototype implementation as described in this section. We decoupled the application layer from the rest of the platform and used a REST API. The user

application layer is implemented in Android while the rest is developped with the Spring Framework [4] with which we implemented the REST API, the Chord network, and a DHT.

Smartphones are always "on the move" and because of their intermittent connectivity it is not straightforward to allow service nodes to spontaneously send data to the mobile device but two alternatives exist:

- to rely on external services provided by the mobile application system (e.g., Google Firebase);
- to regularly poll service nodes from mobile applications.

In the usual case, this question is not of particular importance and many would rely on the first solution. However, for propositions with a particular focus on privacy, like ours, this solution cannot be considered as a suitable one as it would infringe user privacy. For that reason, we have chosen the second approach, which explains why we introduce the `Ping` message from the driver in the requesting phase of the protocol. The polling period must be chosen carefully to offer at the same time good reactivity and low network usage to reduce operational costs. Nevertheless, we assume that drivers have local data plans (as opposed to clients that might be have expensive roaming fees) and we do not foresee particular cost issues as long as polling is done only by drivers.

In mobile environments it is usually not allowed for a terminal equipment (e.g., a smartphone) to open ports for listening. This reason combined to privacy reasons explains why the acknowledgement and the confirmation messages (5 and 6) are transiting through the client service node. Once confirmed, our protocol assumes that direct communication between the driver and the client is possible, which in practice requires complex NAT/firewall traversal mechanisms that that could be implemented by the service nodes. Our prototype does not implement traversal techniques but we anticipate that this particular point is technically complex given the privacy requirements we have.

We only have implemented payment in cash in our prototype. However, to be adopted by user it should support dematerialized payment solutions. Usual electronic payment techniques such as Visa are easy to put in place but they require trusted third parties, which goes against the general idea of our proposition. Blockchain-based cryptocurrencies are more adapted for that usage. One may propose to integrate the currency directly in our system. However, we believe that for adoption by many it is better to connect to already existing cryptocurrency markets, the issue being then to guarantee the privacy while using these platforms.

Users expect to be able to seamlessly use their services on any device. When the service relies on a centralized infrastructure managed by a third party this ubiquity is simple to implement. However in a fully distributed system with privacy preserving properties it is harder to implement. The main issue is to to store the profile in a distributed way such that only the real owner can access private information. A simple solution would consist in protecting the information with encryption and give the key only to the owner. Unfortunately this method is not

adapted for real world usage where users regularly lose their credentials or keys and where recovery mechanisms are used daily by the users (e.g., email recovery). For now our implementation keeps the profile encrypted and its private key on the mobile device as we have not found yet a mechanism that is at the same time truly secured and guarantees privacy and data recovery in any situation.

Finally, an important feature of ride sharing solutions is the trip planner to estimate costs. In theory the client and the driver only need a road map covering the trip region and they can compute the route locally. A solution would be to store worldwide road maps on the devices, but that would require several tens of gigabytes on the devices. This approach minimizes the data roaming costs but is not practical, unless people plan their trip and download the desired locations. An alternative, used by virtually all route planing apps today without causing any particular problem, is to download maps only when needed. The latter solution is acceptable as long as maps and live traffic information can be retrieved by regions to avoid to disclose exact location to the map provider. In parallel the users of our solution must be able to find an exact location. When the address is known exactly the problem is rather easy if downloaded maps are well annotated. However, the trend is to use natural language search for addresses and current services offering such solution rely on a centralized approach meaning that the users must disclose critical information to third parties.

5 Conclusion

The sharing economy relies on centralized platforms causing serious threats on security, privacy, and concentration of power. To tackle these issues we present a fully decentralized and privacy preserving solution. Communication between clients and providers is ensured by a peer-to-peer network and distributed storage. A blockchain plays the role of the marketplace to compute the prices and provide proven trust between the clients and the providers. We carefully designed our communication protocol and data manipulations to ensure data privacy.

We have specifically designed our solution for ridesharing business and our analytical study shows that scalability of a privacy-preserving distributed platform remains a challenge in theory. However, in practice, when applied to real taxi services, we can see that it scales enough to enable new respectful sharing economy businesses.

Acknowledgments

This work has been supported by the project *Data Privacy* funded by the IDEX of Université Côte d'Azur.

References

1. Arcade City: The Future of Ridesharing Is Decentralized. https://fee.org/articles/arcade-city-the-future-of-ridesharing-is-decentralized/

2. Hailing Rides Down Crypto Lane: The Future Of Ridesharing. https://www.forbes.com/sites/andrewrossow/2018/07/18/hailing-rides-down-crypto-lane-the-future-of-ridesharing/, accessed: 2018-12-11
3. LaZooz. http://lazooz.org, accessed: 2018-12-11
4. Spring Framework. https://spring.io, accessed: 2019-02-14
5. TNCs TODAY. http://tncstoday.sfcta.orghttp://tncstoday.sfcta.org, accessed: 2018-08-17
6. Aïvodji, U.M., Gambs, S., Huguet, M.J., Killijian, M.O.: Meeting points in ridesharing: A privacy-preserving approach. Transportation Research Part C: Emerging Technologies **72**, 239–253 (2016)
7. Cadwalladr, C., Graham-Harrison, E.: Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. The Guardian **17** (2018)
8. Dai, C., Yuan, X., Wang, C.: Privacy-preserving ridesharing recommendation in geosocial networks. In: International Conference on Computational Social Networks. pp. 193–205. Springer (2016)
9. Fletcher, D.: How facebook is redefining privacy (2010)
10. Goel, P., Kulik, L., Ramamohanarao, K.: Privacy-aware dynamic ride sharing. ACM Transactions on Spatial Algorithms and Systems (TSAS) **2**(1), 4 (2016)
11. Lua, E.K., Crowcroft, J., Pias, M., Sharma, R., Lim, S.: A survey and comparison of peer-to-peer overlay network schemes. IEEE Communications Surveys & Tutorials **7**(2), 72–93 (2005)
12. Maymounkov, P., Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: International Workshop on Peer-to-Peer Systems. pp. 53–65. Springer (2002)
13. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
14. Pham, T.V.A., Dacosta Petrocelli, I.I., Endignoux, G.F.M., Troncoso-Pastoriza, J.R., Huguenin, K., Hubaux, J.P.: Oride: A privacy-preserving yet accountable ride-hailing service. Proceedings of the 26th USENIX Security Symposium (2017), http://infoscience.epfl.ch/record/228219
15. Piorkowski, M., Sarafijanovoc-Djukic, N., Grossglauser, M.: A Parsimonious Model of Mobile Partitioned Networks with Clustering. In: The First International Conference on COMmunication Systems and NETworkS (COMSNETS) (January 2009), http://www.comsnets.org
16. Robbins, J.M., Sechooler, A.M.: Once more unto the breach: What the equifax and uber data breaches reveal about the intersection of information security and the enforecement of securities laws. Criminal Justice **33**(1), 4–7 (2018)
17. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
18. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Computer Communication Review **31**(4), 149–160 (2001)
19. Tapscott, D., Tapscott, A.: Blockchain revolution: how the technology behind bitcoin is changing money, business, and the world. Penguin (2016)