Oracleboros: Reusing Hyperledger Fabric Mechanisms to Provide Oracle Functionality

Michal Koutenský
Faculty of Information Technology
Brno University of Technology
Brno, Czechia
0000-0001-6912-8713

Vladimír Veselý
Faculty of Information Technology
Brno University of Technology
Brno, Czechia
0000-0002-6346-2152

***Abstract*—**
***Index Terms*—hyperledger fabric, blockchain, decentralized oracle network, periodic transactions**

I. Introduction

Blockchain technologies provide a novel approach to building distributed applications. By combining peer-to-peer networking, tamper-proof storage, and distributed computation, they provide all the necessary building blocks for decentralized applications with low level of trust between actors.

The rise of popularity of blockchain technologies has been lead by Bitcoin. Created as an pseudonymous currency that is resistant to government censorship, Bitcoin utilizes a distributed, append-only ledger of transactions grouped into blocks forming a cryptographically verifiable chain; hence the term blockchain. The network is governed by a set of rules on how new blocks are added, thus maintaining consensus on the ledger state.

While Bitcoin remains the most popular and well-known blockchain technology to this day, it has spawned numerous successors, trying to fix some of Bitcoin's perceived shortcomings or innovate on its design. Notable projects belonging in the first group are Monero, Dash, PIVX, Firo @todospytat honzu, cryptocurrencies developed with the goal of enhancing user privacy. These projects typically use various transaction obfuscation techniques (e.g., tokenization, coinjoining) or employ zero-knowledge cryptography. For the latter, it is important to mention Ethereum. Ethereum, like Bitcoin, is a blockchain-based cryptocurrency. Unlike Bitcoin, whose sole feature is the exchange of assets (and value), Ethereum introduces the concept of smart contracts.

A smart contract is a piece of user code that is executed within the network as part of a transaction. This places some limitations on what the code can do, as it has to produce the same result on all the nodes in the network that execute it. Generally speaking, this means that it cannot interact with any resources outside the blockchain, as such interactions are, by their nature, non-deterministic and do not guarantee availability. On the upside, the inputs and the result — and even the code [1] — of such computation can be independently verified. This enables parties that do not necessarily trust each other to engage and participate in interactions more complex than a simple asset transfer, since all parties can rely on the blockchain network to enforce the agreed-upon rules of interaction: the contract.

This programmability enables blockchain technologies to serve a much wider set of use-cases. Smart contracts now power a diverse ecosystem of decentralized applications including Decentralized Finance (DeFi), gaming, supply chain management, governance, and even healthcare.

Other use-cases, however, still require interaction with the "outside world". Examples might be querying the price of some external asset to calculate the value for a transaction, or checking the outcome of a sports match to reward the person who made the closest guess. This problem is being tackled by *oracles* - nodes or networks of nodes acting as a bridge between a blockchain and external systems. Oracles monitor blockchain activity for events which require their collaboration, and then provide an external system with blockchain information, or query an external system and write the received information back into the blockchain, thus making it available to smart contracts.

So far, all the blockchain technologies mentioned have been permissionless, i.e., public networks anyone can participate in. While having a low barrier to entry is desirable, it also comes with some drawbacks. Most notably, the consensus mechanism used by Bitcoin, Proof-of-Work, has been shown to consume massive amounts of resources and have a severe destructive impact on the Earth's natural environment. In other cases, mainly for use within various industries, e.g., supply chain management, this lack of access control is undesirable.

There now exist a number of blockchain technologies attempting to target these non-cryptocurrency, industrial use-cases by being permissioned, or having the option to run as permissioned, by design. Most notably, these exist under the Hyperledger project umbrella, which itself is under the Linux foundation. It contains industry-leading members such as American Express, IBM, MasterCard, Oracle, T-Mobile, and many others.

Hyperledger Fabric is one of these permissioned blockchains. In addition to having an identity layer based on TLS certificates, its most notable feature is the built-in

Identify applicable funding agency here. If none, delete this.

[1] bytecode

support of private data. This is data that is meant to be disseminated only to a subset of a network's peers (which rules out writing it to the ledger), while preserving the properties that make blockchains an attractive vehicle for decentralized applications in the first place. It is effectively an off-chain database whose state is determined by transactions in combination with a gossip protocol between the peers. Optionally, this data can have a lifetime, after which it is deleted. Currently, the lifetime is measured in blocks. This is unsuitable for scenarios with data retention policies which require data to be deleted after a certain time period (days, weeks, months...), be it a corporate policy or a government regulation. Adapting the mechanism to work with regular SI time units is an open research problem.

We aim to tackle this problem of timely data deletion in Hyperledger Fabric in this paper. Section II provides an overview of prior art, focusing on the use of oracles in permissioned blockchains. In Section III we describe the problem in more detail, and outline how it could be solved with the use of oracles. In Section IV we showcase how all the necessary building blocks are already present in Hyperledger Fabric and propose a theoretical extension to provide this functionality without relying on external oracle services, in keeping with the batteries-included nature of Hyperledger Fabric. Section V contains a discussion about the limitations and possible extensions of our work.

II. Related Work

[1]

TODOVladmir - encrypted data v smart contractoch - oracle a ich implementacia - oracle v permissioned sietach - secure offchain storage - usecase s periodickymi transakciami?

III. Problem Description

One of the unique features provided by Hyperledger Fabric are *Private Collections*, a confidential, mutable access storage. Other popular blockchain technologies which support user programmability (smart contracts) usually only provide storage in the form of a public (with regards to network membership) immutable ledger. For scenarios where data confidentiality is required, the ledger contents can be encrypted to restrict access. Alternatively, an off-chain storage can be used, with varying levels of integration, possibly using the smart contract to exercise access control.

All these solutions suffer from requiring additional work to set up and maintain the supporting services. Whether it is an external storage platform or a management scheme for encryption secrets, these supporting services are needed for the proper functioning of the decentralized application and thus shape its properties. The governance, integration, and failure modes are just some of the things that need to be considered when building such an application. Off-chain storage can be susceptible to tampering unless there are mechanisms in place to prevent or detect such behavior. In the case of storing encrypted data in the ledger, the data will stay there for the whole lifetime of the ledger. If the decryption secrets are leaked, or an adversary manages to break the encryption mechanism, data confidentiality is lost.

Private Collections, by contrast, provide a confidential storage that is well-integrated with the blockchain. It is deployed as part of regular blockhain node deployment; the smart contract API has methods to interact with it; access control rules are analogous to those for the blockchain itself; any activity (as it is done through smart contracts only) is logged in the ledger by the smart contract execution environment; and data, including its history, can be deleted while preserving the necessary audit trail. This greatly simplifies the design and development of applications which need this kind of confidential data storage.

One missing feature is the ability to restrict the lifetime of data in the system. In certain scenarios, there are requirements on how long data can be stored, which necessitates their timely deletion. These requirements may be regulatory and therefore cannot be simply ignored. As a practical example, let us briefly describe the blockchain system for exchange of Passenger Name Records between European Passenger Information Units that we have developed within the TENACITy project, as our experience there directly motivated this work.

For flights which do not originate or land in an EU member state, air carriers are required to collect Passenger Name Records (PNR). These PNRs contain sensitive information about the passenger, such as their personal information, booking information, etc. The air carriers collect this information and forward it to a dedicated Passenger Information Unit (PIU) in the corresponding member state. Each member state has one PIU which collects data for flights originating or ending in their state. In certain cases, a PIU of one member state might need to request some PNR data from another PIU to aid in their investigation. This would be potentially done using our blockchain system, which uses Private Collections to store this data and exercise access control. However, the PNR directive requires the data to be deleted after five years.

Within Hyperledger Fabric, the only automated possibility right now is to set the lifetime in a number of blocks. As blocks are generated on demand, this approach is unreliable and therefore unsuitable for the strict requirements of handling citizen's personal information. The deletion can be triggered externally, which is the approach we have chosen for our system, but this decision comes with its own set of problems. It means that the external service must be reliable and available at all times, else data does not get deleted. A malicious actor could turn it off without compromising their interaction with the system. Even in situations where the unavailability is unintentional and not malicious, leaving data in the system is a liability. A system cannot leak data it does not have.

In light of European Union's actions on protecting citizen privacy, such as the General Data Protection Regulation and the aforementioned PNR directive, it is likely that any other systems attempting to transfer sensitive personal information (e.g., finance, travel, healthcare, administration) will face similar challenges. As such, we believe it would be valuable for Hyperledger Fabric, a popular blockchain technology targeting

industry use-cases, to support such feature natively.

IV. Proposed Hyperledger Fabric Extension

To automatically delete data after a certain time period, we need three things: a) some kind of time information related to the data's lifetime (i.e., the creation timestamp and lifetime duration, or the expiration timestamp); b) a function which takes the current time and a set of data (together with the lifetime information) and produces a subset of the data that is meant to be deleted; and c) a way to periodically run the function from b). We will refer to a) as *cleanup metadata*, to b) as *cleanup function*, and to c) as *cleanup trigger*.

As Hyperledger Fabric if quite flexible with regards to how data is stored and accessed, we will consider the following setup when designing our solution. A Fabric Channel contains n participating organizations. Each participating organization has an m number of peers, where m can be different for each organization. Each organization has a single corresponding Private Collection which holds its data. These collections use the key-value store provided by LevelDB. The organization is responsible only for the timely deletion of data in its Private Collection.

The capabilities provided by Fabric are very well suited for handling both the *cleanup metadata* and *cleanup function*. A smart contract, which we will call the *cleanup contract*, can be deployed within the channel to provide this functionality. It would have the following methods:

- `AddData(collection, id, timeInfo)`
- `RemoveExpiredData(currentTime)`
- `RenameData(collection, oldId, newId)`

`AddData` allows users to inform the contract about the existence of new data which needs to be cleaned up. In cases where the key associated with some data needs to be changed (e.g., for more efficient access when using composite keys), the cleanup contract can be informed of this change by the `RenameData` method. Crucially, this preserves the associated cleanup metadata, when compared to an otherwise equivalent delete/add sequence. Both of these methods can be called by other smart contracts when manipulating data as part of their normal operation, to keep the cleanup metadata consistent. The mechanism of storing the cleanup metadata can be implemented in various ways (ledger, single private collection, multiple private collections) depending on the confidentiality requirements of the channel.

`RemoveExpiredData` is the core of the cleanup contract. It implements the cleanup function and uses the output to execute the side effect of actually deleting the data from the corresponding private collection. This coupling is intentional; leaving the responsibility of deleting the data to the caller would weaken the guarantees we are trying to achieve.

Such a smart contract can be implemented fairly easily in the current version of Fabric. What is missing for a fully functioning, self-contained system, as discussed in Section III, is the cleanup trigger.

Unlike `AddData` and `RenameData`, `RemoveExpiredData` cannot be called by a user smart contract which implements the trigger logic, as that smart contract cannot execute its methods without having been invoked in turn by something else, effectively only shifting the problem around. Indeed, were it possible, the trigger could be contained in the cleanup contract itself, avoiding this unnecessary indirection.

This suggests that the trigger must be external, i.e., a user manually creating a transaction, or an external application creating a transaction on a user's behalf (possibly an oracle). This is how transactions are commonly created. However, this creates a dependency on such an external service, or, in the case of explicit user activity, is not automatic.

Due to the permissioned nature of Fabric, all the participating peer nodes have identities with corresponding organization membership. This means they can sign transactions. A peer could, therefore, periodically submit a transaction invoking `RemoveExpiredData` on the cleanup contract. The period can be part of the channel configuration so that it can be adjusted for different scenarios, even during a channel's lifetime; or, it can be part of the peer node configuration, set by the organization as deemed necessary. As the presence of a peer node is required for the organization's participation in the channel, this does not introduce any new dependency or failure point in the system.

An organization can, however, have more than one peer. If each peer submits such a transaction, only the first one will be valid and the following ones will fail due to a conflict. Such conflicts pose a scalability issue, negatively impacting network throughput and latency [2]. Ideally, we only want one transaction per organization per cleanup time period to be submitted.

To achieve that, we need to select a single available peer to act on behalf of the organization. More precisely, it is the organization's peers themselves which need to agree on this. This is a familiar consensus problem common to all sorts of distributed systems, blockchains included. In fact, it is analogous to the problem faced by orderer nodes within a channel, which need to agree on how to order the transactions in a block. One of the mechanisms used by Fabric is the Raft consensus algorithm, where the orderer nodes elect a leader to do the actual ordering. We can apply that solution here as well — the organization's peer nodes will elect a leader using Raft to submit the transaction.

V. Discussion

Our proposed extension manages to provide a subset of functionality usually provided by oracles — access to external information, i.e., current time — without introducing additional actors and services into the system. However, it does so under very constrained conditions.

The proposal is limited to one "cleanup period". If the system uses multiple kinds of data with different cleanup periods, the period must be set to the greatest common divisor of all the periods. This can result in a significant increase in the number of transactions generated in certain scenarios.

Consider two periods[2], 2 and 3, which share no common divisor except 1.

TABLE I
A VISUALIZATION OF TRANSACTION GENERATION FOR PERIODS 2 AND 3 AND THEIR GREATEST COMMON DIVISOR.

Period	T_1	T_2	T_3	T_4	T_5	T_6	Txs
2		•		•		•	4
3			•			•	
1	•	•	•	•	•	•	6

Table I visualizes the transaction generation over a period of their lowest common multiple. Triggered independently, this would create 4 transactions; when merged, it creates 6, a 33% increase in the number of transactions. For periods 4 and 7, the results are much worse: 10 and 28 respectively, resulting in a 65% increase — over half of the generated transactions are not required and do no useful work.

A possible solution would be to run each period trigger independently, with its own leader election and transaction generation cycle. This suffers from a complementary problem — generating duplicate transactions when the periods align (as in T_6 in Table I). As the number of period triggers increases, such alignments will happen more often, possibly generating even more than just two transactions at a time. Therefore, the scalability of this approach requires further evaluation. Even if the cleanup contract(s) were designed in a way to prevent these transactions from creating collisions, the extra transactions are redundant, as they carry the same information. Ideally, the system would allow specifying these complex requirements and trigger transactions accordingly, possibly through a powerful programmable interface.

The proposal assumes that the peer nodes have correctly synchronized time and do not behave in a malicious way. While not completely unwarranted, as the peer nodes are in control of the organization, we should still consider the case where some nodes do not report the correct time. This problem occurs in oracle networks, where even honest nodes might not agree on the value of some data due to the natural variability of the external data source they query [3]. Chainlink solves this through the use of an aggregation function, which takes the collected inputs and produces a single value, accounting for the variations. A similar approach could be used here. This would, in addition, allow the trigger to not only supply the current time, but other external information obtained through calling an external data source. For auditing purposes, the values provided by each node could even be recorded in some kind of immutable decentralized storage.

The proposal assumes a specific kind of channel configuration, especially with regards to how data is stored in private collections, the rules for accessing it, and the responsibility for timely deletion. The scenario has been selected for its simplicity and ease of understanding, as well as being closely resembling our practical problem in the TENACITy project. To support a wider variety of configurations, an additional mechanism for managing consensus groups would need to be introduced, as it is no longer possible to use an implicit membership (organizational association). This is similar to the need to define channel membership.

Observant readers might have noticed that the modifications suggested to make our proposal more general resemble features found in blockchains and/or decentralized oracle networks. This connection is briefly mentioned in the Chainlink 2.0 whitepaper [1] on page 21:

> A DON could in principle alternatively use a highly performant permissionless blockchain for its ledger in its role supporting an equally scalable layer-2 or blockchain system. Similarly, hybridization is also possible: The DON could in principle be composed of nodes that are validators in an existing blockchain, e.g., in Proof-of-Stake systems in which committees are selected to execute transactions. This particular mode of operation requires that nodes operate in a dual-use manner, i.e., operate both as blockchain nodes and DON nodes.

The whitepaper discusses Chainlink's vision for the next generation of their oracle service, focusing on novel features such as off-chain data aggregation, fair transaction sequencing, and data source authentication. It does not elaborate further on the consequences of this hybridization.

We consider it an interesting development to have arrived at a similar conclusion from "the other end" — instead of having a vision of how DONs should work and considering whether existing blockchain technologies could be utilized to support some of its functions, we took an existing blockchain technology and tried to adapt its mechanisms to enable additional usescases, reaching something resembling a DON.

To the best of our knowledge, there is no research studying this duality. Hyperledger Fabric, somewhat unusually in the blockchain landscape, already natively supports both on-ledger and off-ledger (private collections) storage to be used by application developers; a feature envisioned to be provided by DONs [1]. This means that hybridization has already been happening in practice, possibly by chance. Intentionally designing and implementing such complete "oraclechain", able to provide both blockchain and oracle functionality — and thus able to act as a supporting service for itself — could provide valuable insight into the future of distributed applications as envisioned by Web 3.0.

VI. CONCLUSION

ACKNOWLEDGMENT

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

[2]Of your favorite unit of time.

REFERENCES

[1] Lorenz Breidenbach, Christian Cachin, Benedict Chan, *et al.*, *Chainlink 2.0: Next steps in the evolution of decentralized oracle networks*, Apr. 15, 2021. [Online]. Available: https://research.chain.link/whitepaper-v2.pdf.

[2] A. Stoltidis, K. Choumas, and T. Korakis, "Performance optimization of high-conflict transactions within the hyperledger fabric blockchain," in *2024 6th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, Berlin, Germany: IEEE, Oct. 9, 2024, pp. 1–4, ISBN: 979-8-3503-6784-3. DOI: 10.1109/BRAINS63024.2024.10732190. [Online]. Available: https://ieeexplore.ieee.org/document/10732190/ (visited on 03/08/2025).

[3] S. Ellis, A. Juels, and S. Nazarov, *ChainLink: A decentralized oracle network*, Sep. 4, 2017. [Online]. Available: https://research.chain.link/whitepaper-v1.pdf.