Foundational Oracle Patterns: Connecting Blockchain to the Off-chain World

Roman Mühlberger[1], Stefan Bachhofner[1], Eduardo Castelló Ferrer[2], Claudio Di Ciccio[3], Ingo Weber[4], Maximilian Wöhrer[5], and Uwe Zdun[5]

[1] Vienna University of Economics and Business, Vienna, Austria
[2] Massachusetts Institute of Technology, Cambridge, United States
[3] Sapienza University of Rome, Rome, Italy
[4] Technische Universitaet Berlin, Berlin, Germany
[5] University of Vienna, Vienna, Austria

Abstract. Blockchain has evolved into a platform for decentralized applications, with beneficial properties like high integrity, transparency, and resilience against censorship and tampering. However, blockchains are closed-world systems which do not have access to external state. To overcome this limitation, oracles have been introduced in various forms and for different purposes. However so far common oracle best practices have not been dissected, classified, and studied in their fundamental aspects. In this paper, we address this gap by studying foundational blockchain oracle patterns in two foundational dimensions characterising the oracles: (i) the data flow direction, i.e., inbound and outbound data flow, from the viewpoint of the blockchain; and (ii) the initiator of the data flow, i.e., whether it is push or pull-based communication. We provide a structured description of the four patterns in detail, and discuss an implementation of these patterns based on use cases. On this basis we conduct a quantitative analysis, which results in the insight that the four different patterns are characterized by distinct performance and costs profiles.

Keywords: Blockchain · Design patterns · Software patterns · Oracles

1 Introduction

Conceptually, a blockchain is an append-only store for transactions, which is distributed across many machines and structured into a linked list of blocks [26]. Based on its decentralized nature, structure, and use of cryptographic protocols, blockchain technology provides a modern platform for distributed applications with properties like high integrity, transparency, and resilience against censorship and tampering. This creates, among others, new opportunities and challenges for inter-organizational business processes [16]. These inherent properties make blockchain technology a good fit for use cases where data integrity is of crucial importance, e.g. clinical trials [22,12], food security [3], or financial risk when dealing with business partners [26, Ch.12]. Consequently, organizations realize efficiency and effectiveness gains with blockchain technology as business processes can have a higher degree of automation, e.g., by running business processes on the blockchain [21] or by automating information

Table 1. An overview of the four oracle types.

	Pull	Push
Inbound	The **on-chain** component **requests** the off-chain state from an **off-chain** component	The **off-chain** component **sends** the off-chain state to the **on-chain** component
Outbound	The **off-chain** component **retrieves** the on-chain state from an **on-chain** component	The **on-chain** component **sends** the off-chain state to an **off-chain** component

exchange between mutually untrusting parties. Many such applications are made possible by a feature of second-generation blockchains, *smart contracts*, which "are programs deployed as data in the blockchain ledger, and executed in transactions on the blockchain" [26]. With smart contracts, blockchains become decentralized, neutral execution platforms for user code.

Regardless of the generation, blockchains are closed-world systems: from inside, one can only access data that is on the blockchain already. Oracles have been proposed to mitigate that limitation. In the context of blockchains, an oracle is a component that can transfer data between the outside world and the blockchain. However, the implementation of oracles provides considerable conceptual challenges as they can be regarded as a centralized point of failure or may introduce security and trust concerns [16]. Consequently, much of the research regarding oracles focuses on how to address these security and trust concerns, e.g., by using multiple independent oracle instances to form a decentralized oracle [25], extending trust properties to off-chain computation [10], or strengthening trust in incoming data [13]. However, foundational aspects of blockchain oracles that allow for their categorization and abstraction have not been subject to close investigation yet.

In this paper, we address this gap by examining two core dimensions of oracles: (i) the *direction*, i.e., whether the data flow is *inbound* or *outbound* from the viewpoint of the blockchain; and (ii) the *initiator* of the data flow, i.e., whether it is *push* or *pull*-based communication. There are four combinations of these options, an overview of which is shown in Table 1. We describe each of these as a pattern, and examine its characteristics. Note that, on this level, the four patterns can be implemented without relying on smart contracts, i.e., even on first-generation blockchains like Bitcoin. Each of the patterns can also be suitably combined with other, higher-level patterns from the literature, like decentralization or provable computation.

To characterise the different patterns, we implemented them in the context of two use cases, and use these implementations for the purpose of obtaining measurements. To this end, the implementations are based on Ethereum, and we sent over 2,500 transactions to the Ethereum test network to obtain concrete data. This allows us to quantitatively study the characteristic differences between the four oracle patterns. In particular, we focus on time (latency) and cost.

The remainder of the paper is structured as follows. Section 2 introduces background literature and related work. The patterns are described and contrasted in Section 3. The use cases for the implementation are described in Section 4. On the basis of the implementation, we analyze the four patterns with respect to time and costs in Section 5.[1] Next, we discuss our results and threats to validity in Section 6. Finally, the paper concludes in Section 7.

[1] The source code can be found at https://github.com/MacOS/blockchain-oracles-data-collection

2 Background and state of the art

In a significant number of times, applications built on blockchain infrastructure require data from real world states and events [4,9]. Examples include financial data, weather-related information, random number generations or arbitrary data from off-chain devices and web services accessible via Application Programming Interfaces (APIs). Blockchain oracles provide a way to interact with the off-chain world [26]. Oracles can be implemented as software (interacting with online sources) or hardware (interacting with the physical world), human (interacting with individuals) or computation-based oracles (performing off-chain calculations), single-source (centralized) or consensus-based oracles (decentralized, using a multitude of sources) [4]. In this paper, we abstract from the way in which oracles are implemented and focus on the foundational patterns they realize. Next, we discuss the basic notions behind blockchains and elaborate on state-of-the-art solutions adopted thus far for the realization of oracles.

Blockchain. At the core of a blockchain lies the transaction, that is, the transfer of value between accounts. Transactions are temporally ordered and stored in a sequential structure named ledger. Every participating full node in the blockchain network keeps a local copy of the ledger. Updates in the network are communicated via blocks, each collating the transactions to be appended to the ledger. To generate and broadcast new blocks, the so-called mining nodes can be required to prove their trustworthiness, e.g., by solving computationally hard problems (Proof of Work). A consensus algorithm allows for the eventual consistency of the distributed ledger. Every block is linked to the previous one via hashing, thus forming a chain – hence the name, blockchain. Smart contracts turn blockchains such as Ethereum [23], Hyperledger Fabric [7] and Algorand [5] into programmable infrastructures. Developers can encode smart contracts with a programming language and compile them to bytecode. Upon deployment, smart contracts are associated with a unique address. They are executed and saved across all connected nodes of the network. The invocations have a computation price expressed in terms of gas. In order to store information, e.g., on the Ethereum blockchain, it can be placed into a transaction payload and possibly added to the contract storage, contract logs, or kept in the transaction payload [24]. After the transaction is included into a block, the information is publicly accessible within the network.

Blockchain oracles. A plethora of commercial and open-source tools have emerged that implement inbound oracles. *Orisi*[2] is a solution for a distributed set of inbound oracles for Bitcoin, which are executed by independent and trustworthy third parties. The majority of all oracles has to agree on the outcome from external data. To fulfill this purpose, money from senders and receivers is parked into a multiple-signature address, including their signatures as well as the signature address of the majority of the oracles result. In our framework, Orisi is categorized as a pull-based inbound oracle. *Oraclize*, recently rebranded as *Provable Things*[2]is a popular service for inbound oracles that works with multiple smart-contract-enabled blockchain platforms. The service acts like a trusted intermediary between blockchains and a variety of independent data sources. It also provides a mechanism to mitigate *corrupt* oracles [17]. Its Provable Engine executes a set of instructions to react as certain

[2]Orisi: https://orisi.org/. Provable Things: https://provable.xyz. TinyOracle: https://github.com/axic/tinyoracle. ChainLink: https://chain.link/. All links accessed on June 7, 2020.

conditions are met, thus making it classifiable both as a push-based and a pull-based inbound oracle. Other services which are natively classifiable as pull-based follow. In the Ethereum-specific *TinyOracle*[3]an intermediary contract acts as a receiver for the actual contract and simultaneously emits an event to the subscribing RPC client. The *lookup* contract stores both query and respondent addresses, while the *sample client* contract calls the oracle service of TinyOracle. *Reality Keys* provides a combination of both automated and human-driven pull-based inbound oracles [17]. *Chainlink*[4]offers a general-purpose framework for building decentralized inbound oracles, providing decentralization on both oracle and data-source levels. A Chainlink node can have multiple external adapters for different data sources. *Witnet* [18] provides a decentralized oracle network protocol based on Ethereum. It also enables miners to earn tokens. An Ethereum bridge is implemented, providing Witnet nodes to run Ethereum nodes with the option to operate with Ether and make contract calls.

Blockchain inbound oracles have also been considered in a number of research works. Xu et al. [24] introduce the concept of *validation oracles*, namely trusted third-party operators (either automatic or human) that act as inbound oracles. The authors distinguish between *internal* ones, periodically transmitting external verified data to the blockchain, and *external* ones, operating as trusted external validators of transactions based on information that is external to the blockchain. According to our scheme, we see that the former is push-based and the latter is pull-based. Adler et al. [2] introduce a decentralized pull-based inbound oracle service. The implementation provides a voting game, which decides the truth or inaccuracy of propositions. Players can be *voters* or *certifiers*. While certifiers play a role in cases with the requirement for high accuracy, voters are utilized for low-risk/low-reward roles. Due to the random selections of propositions, a level of security is provided against manipulation. We remark that the successful implementation of random generators is also part of the realization of oracles. Zhang et al. [27] present *Town Crier*, a push-based inbound oracle that acts like a data-feed system connecting a blockchain with a back-end that scrapes HTTPS websites.

We can observe that, thus far, the vast majority of the efforts has been devoted to the design and implementation of inbound oracles. Indeed, a recent technical report of ISO/TC 307 describes oracles for their sole task of providing off-chain information to the blockchain [14]. In this paper, however, we also investigate and specify the patterns behind the opposite information flow, namely that of outbound oracles, also known as *reverse* oracles [25].

3 Patterns

In this section, we describe in detail basic oracle patterns resulting from the partitioning of the direction (inbound/outbound) and initiation of data flow (pull/push) between on-chain and off-chain components. Figure 1 shows the data flow along the fundamental dimensions outlined above. When applying this partitioning, a basic distinction can be made between inbound oracles and outbound oracles, each of which can be further refined according to data pull and push strategies.

Before discussing each pattern in more detail, we first give a general overview of the patterns and their respective conceptual structural components (also called “pattern participant”) in Fig. 2. The blockchain is considered to be part of a larger software system, with software

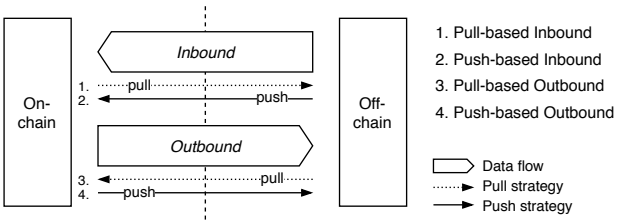

Fig. 1. Conceptual overview of the oracle data flow partitioning.

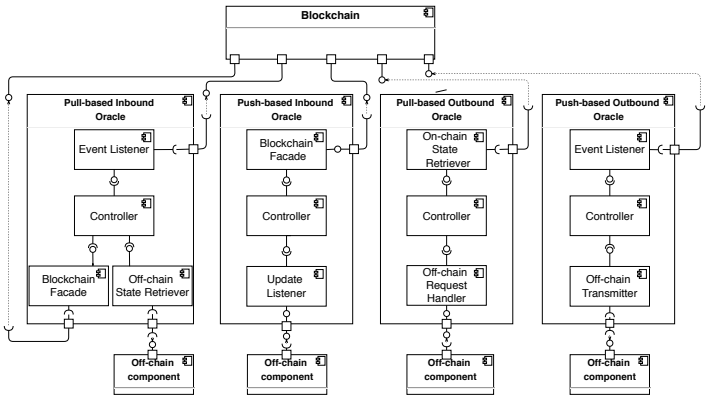

Fig. 2. An overview of the oracle types and conceptual structural components.

components being located on and off-chain. In such an environment, it is often necessary to be able to communicate across system boundaries in both directions to exchange information. For example, components on the blockchain (such as smart contracts) may require knowledge from software components outside the blockchain, and vice versa. The outside world requires knowledge from the blockchain, too. Regarding the terminology used throughout this paper, note that the term "event" in relation to the blockchain refers to any activity that can take place on the blockchain (e.g., data is persisted, a transaction occurs, a block is added, etc.).

3.1 Inbound Oracle

An inbound oracle transmits information from the outside world to the blockchain. As a blockchain cannot directly acquire information from the outside world, it relies on the outside world pushing information into the network. Given this fact, the most obvious approach to obtaining external information on the blockchain is to alert the outside world about the need to push required information into the network. This approach is described in the *pull-based inbound oracle* pattern and is characterized by the fact that the exchange of information is initiated on-chain.

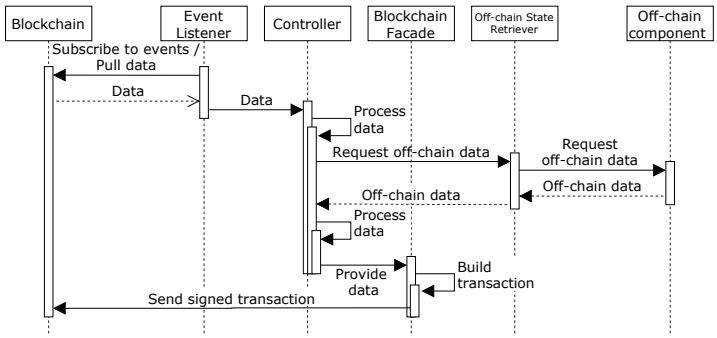

Fig. 3. Sequence diagram showing the component interactions for the *pull-based inbound oracle*.

PATTERN: Pull-based inbound oracle

Problem A blockchain application requires knowledge contained outside of the blockchain, but since blockchains are closed systems, applications cannot directly acquire information from the outside world.
Solution A *pull-based inbound oracle* allows blockchain applications to request states from off-chain components. When a blockchain application requests an off-chain state, the *pull-based inbound oracle* receives this request, gathers the state from off-chain components, and sends the result back to the blockchain (via a transaction).
Benefits State requests are initiated in the blockchain. Thus the whole process is transparent. It can be traced whether off-chain data was successfully provided (in time) or not.
Drawbacks State requests have to be initiated from the blockchain, this induces a passive character. Further, the *pull-based inbound oracle* response time depends on the speed of the blockchain network, which may lead to a bottleneck. Network congestion may result in delayed or missed off-chain state retrieval, as the oracle only starts working after it registers requests from the blockchain.

The conceptual interaction of the pattern participants is shown in Fig. 3: An *Event Listener* subscribes to relevant events on the blockchain, which forwards event data to a *Controller*. The *Controller* gathers required data from an off-chain component via an *Off-chain State Retriever*. The gathered data may be further processed by the *Controller* before it is returned to the blockchain via a *Blockchain Facade*.

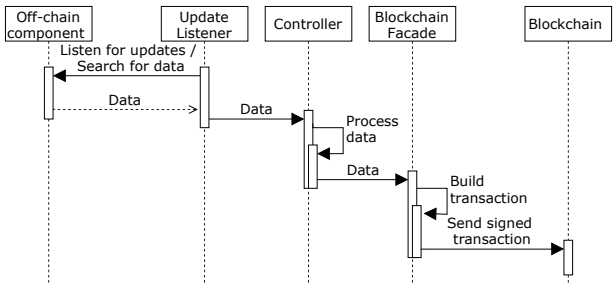

Fig. 4. Sequence diagram showing the component interactions for the *push-based inbound oracle*.

Another approach to transferring external knowledge to the blockchain is to monitor changes in the off-chain world that are relevant to the blockchain and to transfer these changes to the network. This approach is described by the *push-based inbound oracle* pattern and is characterized by the fact that the exchange of information is initiated off-chain.

PATTERN: Push-based inbound oracle
Problem A blockchain application must be supplied with knowledge outside the blockchain, but since blockchains are closed systems, this knowledge cannot be directly communicated.
Solution A *push-based inbound oracle* allows off-chain information to be propagated to the blockchain by monitoring off-chain state changes and forwarding them to the blockchain.
Benefits Scattered or irregularly updated data outside the blockchain is proactively pushed to the blockchain application. Therefore, the application does not require capabilities to search and query off-chain data. In addition, data can be checked more easily by the *push-based inbound oracle*, considering the limited functionality of blockchain environments.
Drawbacks The *push-based inbound oracle* is not deployed or triggered on the blockchain, making data provision entirely dependent from (non-distributed) applications running off-chain. To manipulate blockchains with incorrect information, an adversary only needs to compromise the off-chain component(s) from which the oracle receives the data.

The *push-based inbound oracle*, as conceptually illustrated in Fig. 4, listens to relevant off-chain component updates via an *Update Listener* and forwards the data to the *Controller*. The *Controller* may process (e.g., filter, verify, etc.) the data before it is sent to the blockchain via a *Blockchain Facade*.

3.2 Outbound Oracle

An outbound oracle transmits information from the blockchain to the outside world. Due to its underlying properties, a blockchain can store state information in the form of transactions, but it cannot actively communicate that state to the off-chain world. In light of this, the most obvious path to obtaining data from the blockchain is to fetch it. This approach is described by the *pull-based outbound oracle* pattern and is characterized by the fact that the exchange of information is initiated off-chain.

PATTERN: Pull-based outbound oracle
Problem Knowledge contained on the blockchain is needed outside the blockchain, but since blockchains are closed systems, the outside world cannot directly request information.
Solution A *pull-based outbound oracle* allows blockchain data to be queried and filtered to make it available to the outside world. It can be called from (off-chain) components to pull (all) blockchain data and query relevant information.
Benefits The *pull-based outbound oracle* allows to decouple external status requests from the actual status retrieval. Thus, the pattern offers the possibility of uniformly accessing and querying relevant information on the blockchain.
Drawbacks Depending on the size of the blockchain and the knowledge of the location of the requested information, the provision of the data may take some time.

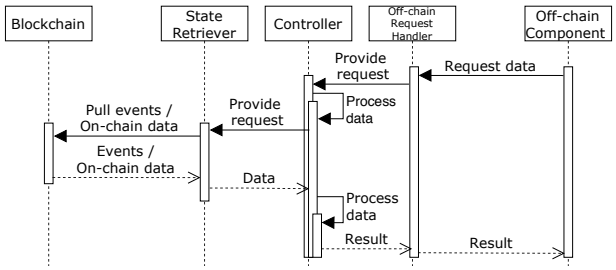

Fig. 5. Sequence diagram showing the component interactions for the *pull-based outbound oracle*.

The *pull-based outbound oracle*, as conceptually outlined in Fig. 5, receives off-chain data requests via an *Off-chain Request Handler* and forwards the requests to the *Controller* to process the request before forwarding it to the *State Retriever*, which is responsible for retrieving data from the blockchain. The result is returned to the *Controller*, which may process the data before it is sent to the off-chain requester via the *Off-chain Request Handler*.

Another approach to transferring internal information from the blockchain is to observe changes on the blockchain that are relevant to the outside world and to transfer these changes off-chain. This approach is described by the *push-based outbound oracle* and is characterized by the fact that the exchange of information is initiated on-chain.

PATTERN: Push-based outbound oracle
Problem Knowledge contained on the blockchain must be available outside the blockchain, but since blockchains are closed systems, applications cannot directly propagate information to the outside world.
Solution A *push-based outbound oracle* monitors the blockchain for relevant changes to subsequently trigger or perform activities outside the blockchain.
Benefits The *push-based outbound oracle* constantly monitors the blockchain. Thus, it is possible to (partially) automate blockchain related tasks by taking action when a blockchain state is updated.
Drawbacks The *push-based outbound oracle* is required to run continuously in order to monitor all events (on time) on the blockchain. In case the oracle unexpectedly stops, updates (depending on the implementation) may be missed. In addition, depending on the speed of the blockchain network, delays can occur, which can lead to unwanted delays in time-sensitive interactions.

The *push-based outbound oracle*, as shown in Fig. 6, subscribes to relevant events on the blockchain via an *Event Listener* and forwards event data to the *Controller*, which may process the data before it is sent via the *Off-chain Transmitter* to an off-chain component.

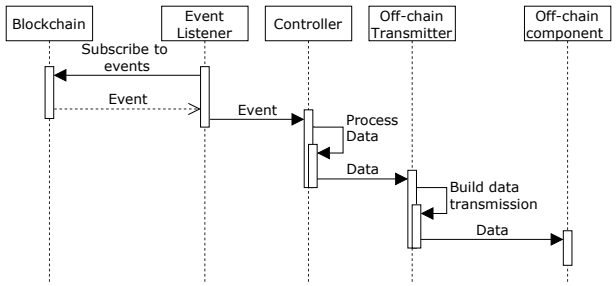

Fig. 6. Sequence diagram showing the component interactions for the *push-based outbound oracle*.

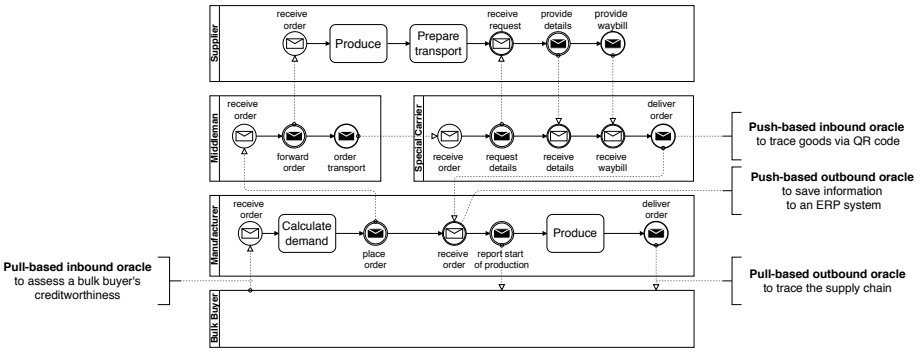

Fig. 7. A supply chain process (in BPMN, from [21]), showing where oracles are employed.

4 Use cases

Among other successful use cases, the blockchain has been adopted as a backbone for the execution of multi-party business processes [8]. This section describes some use cases in that domain we considered to implement the oracle patterns.

Figure 7 illustrates a simplified model of a supply-chain process inspired by [21]. The initiator of the process is a bulk buyer who places an order. The order is then forwarded to a manufacturer. The manufacturer, in turn, calculates the needed material and delegates a middleman to forward the order to a supplier and to book the transportation by a special carrier. When materials are ready, the carrier takes care of the transport from the supplier site to the manufacturer's. Finally, the manufacturer produces the goods and delivers them to the bulk buyer.

The execution of the process is tightly bound at many stages to data flows from and toward the blockchain system. The transfer of information from the off-chain world to the on-chain environment and vice-versa is carried out by the oracles. We focus in particular on four oracles – one for each pattern. They are highlighted with textual comments in Fig. 7 and detailed next. Our implementations of those oracles are based on the Ethereum blockchain, *Web3* library and *Python*. Our additional modules for QR scans are based on *QR-Code-Scanner*. [5] The source code is available, see Footnote 1.

Figure 8 depicts the oracle-based interaction between a bulk buyer and the manufacturer. The bulk buyer places an order over a web application (1). The order is forwarded to the manufacturer if the creditworthiness of the buyer is verified. The order details including the order ID and information on the customer and bulk buyer are forwarded via a transaction to a smart contract (2). The smart contract publishes an event containing information on the bulk buyer such as name and Tax ID. The Event Listener of a *pull-based inbound oracle* subscribes to updates on such events. To retrieve information on the buyer's creditworthiness, the oracle calls the API of an external credit assessment service upon request via its Off-chain State Retriever (4). As the oracle processes the response (5) with the Controller, it returns this

[5] Web3: https://github.com/ethereum/web3.py. Python: https://www.python.org/. QR-Code-Scanner: https://github.com/code-kotis/qr-code-scanner. All links accessed on June 7, 2020.

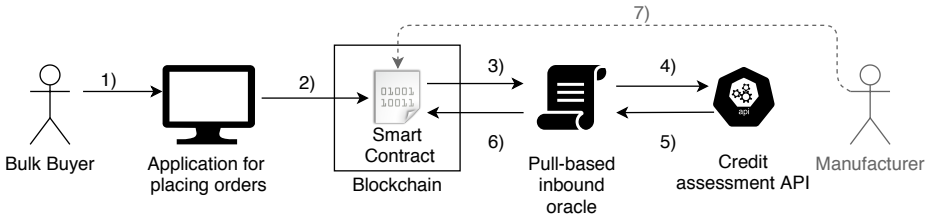

Fig. 8. Oracle-based creditworthiness verification of actors in the supply chain process of Fig. 7.

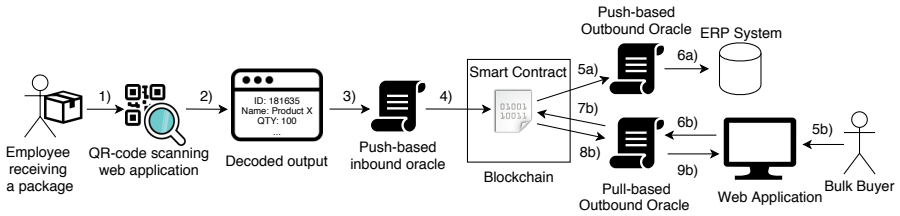

Fig. 9. Oracle-based tracing of goods via QR Code scanning in the supply chain process of Fig. 7.

information as transaction data to the smart contract (6) with its Blockchain Facade. Finally, the manufacturer accesses the order after the verification (7).

Figure 9 illustrates a blockchain-based use-case for the tracing of goods in a supply chain via QR-code scanning. It involves three oracle patterns. The use case starts with an employee registering the delivery of a package. To certify the sending of the package, the employee uses a device with a QR-code scanning application (1). The information from the QR code includes the order ID, the name and the quantity of items (2). Thereafter, the *push-based inbound oracle* receives the data from the scan (3) via its Update Listener. The Controller of the oracle encodes the data into a blockchain transaction, enriching it with the location and current timestamp. Its Blockchain Facade transmits the data to a smart contract (4). The smart contract, in turn, publishes an event that is parsed by the Event Listener of a *push-based outbound oracle* (5a). The Controller of the latter decodes the event data and further passes it along to an ERP system via an Off-chain Transmitter (6a). The bulk buyer traces the production of the items identified by the order ID over the blockchain via a web application (5b). Upon request, the web application calls the Off-chain Request Handler of a *pull-based outbound oracle* (6b). The oracle Controller turns the request into a query for the On-chain State Retriever. As the requested information is found (8b), the application provides the entire data record on the product(s) back (9b). We implemented these use cases to serve as a basis of the analysis described next.

5 Analysis of Performance and Transaction Fees

This section describes our findings from a quantitative analysis on proof-of-concept implementations of the four oracles, based on the use cases presented above.

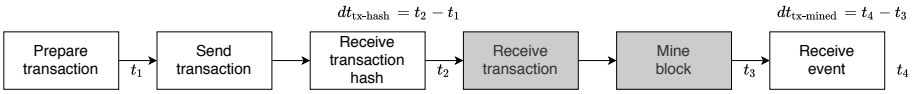

Fig. 10. Schematic process for measuring latency, with off-chain (white) and on-chain (grey) tasks.

Setup. We focus on the time and costs dimensions. Regarding time, specifically latency, we are interested in answering two questions. The first question is whether we observe differences in time among the different implemented patterns. This might indicate that dissecting oracles the way we propose in this paper is not only important from a software engineering perspective, but also with respect to the range of use cases they cater for. The second question is whether the observed timings are caused by our experimental settings. We perform all experiments on Ropsten, a test network for Ethereum. We choose Ropsten as it is accepted in the scientific literature for testing purposes [1,6,15]. The test code and the code used for the quantitative analysis are available, see Footnote 1. The smart contract *arrival.sol* mimics the use case from Fig. 9, which we use to evaluate the *push-based inbound oracle*, the *pull-based outbound oracle* and the *push-based outbound oracle*. It is deployed at address `0x1186aEDAb8f37C08CC00a887dBb119787cfE6AAf`. The smart contract *customer.sol* mimics the use case from Fig. 8, which we use to evaluate the *pull-based inbound oracle*. It is deployed at address `0x9c2306eccc5afa6ee0c1eca6deab66cc336c3b3d`.

To assess the costs of inbound oracles, we measure the consumed gas. Note that gas costs also captures the computational and storage effort. We convert Ether to Euros by using the mean exchange price for Ether over the evaluation period (144.86 €/Ether), and gas usage converts to Ether using the gas price of the transactions (on average 7.45×10^{-10} Ether / gas).

The outbound oracles read from the blockchain and we thus focus on the time dimension. Note that we keep the retrieval state of the *pull-based outbound oracle* constant to eliminate this as a varying factor. Furthermore, in the implementation of the *pull-based inbound oracle* we do not store any states in the receiving smart contract, because the transaction invokes the client smart contract directly and we exclude its handling of the data in the experiment. In contrast, the *push-based inbound oracle* stores the state and emits an event; this is necessary so that the client smart contract can retrieve the state.

To measure latency (see also Fig. 10) we capture the time between a transaction being sent to the blockchain node (t_1) and the time when we receive the transaction hash (t_2). We indicate the difference as $dt_{\text{tx-hash}}$. For the *push-based outbound oracle*, we measure the period between the timestamp of the block that included the transaction (i.e., the timestamp when the miner started mining that block, t_3), and the time in which we receive the event (t_4). We name the difference as $dt_{\text{tx-mined}}$. When clear from the context, we will refer to both measures as dt. It is debatable whether the mining time should be part of the latency measurement. Note that the time between the submission of a transaction and its inclusion / commitment on the ledger varies drastically between blockchain platforms. Additionally, various other factors need to be taken into account, such as network congestion and, for commit time on Proof-of-Work blockchains, the number of confirmation blocks which is a user-defined parameter – see

Table 6. Summary statistics of time and costs for oracle invocations (on the Ropsten Ethereum test-net).

	n	**mean**	**std**	**min**	$x_{0.25}$	$x_{0.50}$	$x_{0.75}$	**max**
Push-based inbound oracle	2437							
$dt_{\text{tx-hash}}$ [seconds]		0.53	0.08	0.46	0.49	0.50	0.54	2.14
Transaction cost [Gas]		44,827	1,265	36,739	45,139	45,235	45,259	45,319
Transaction cost [€]		4.96×10^{-3}	5.78×10^{-3}	2.96×10^{-11}	6.55×10^{-5}	6.53×10^{-3}	6.55×10^{-3}	1.37×10^{-1}
Push-based outbound oracle	438							
$dt_{\text{tx-mined}}$ [seconds]		16.20	15.95	0.53	5.41	10.71	21.44	129.95
Pull-based inbound oracle	126							
$dt_{\text{tx-hash}}$ [seconds]		0.52	0.05	0.46	0.48	0.50	0.52	0.78
Transaction cost [Gas]		22,770	0	22,770	22,770	22,770	22,770	22,770
Transaction cost [€]		8.91×10^{-5}	3.96×10^{-4}	7.91×10^{-7}	7.91×10^{-7}	7.91×10^{-7}	7.91×10^{-7}	1.85×10^{-3}
Pull-based outbound oracle	2611							
$dt_{\text{tx-hash}}$ [seconds]		0.13	0.03	0.11	0.11	0.12	0.12	0.45

e.g. [20] for details and measurements. Here, we measured simple inclusion time without additional confirmation blocks, as a placeholder and to highlight this underlying issue.

Results. Figure 11 and Table 6 show the results of our experiments. The *pull-based outbound oracle* is the fastest of the four oracles with a mean dt of 0.13 ± 0.03 seconds, while the *push-based outbound oracle* is the slowest with a mean dt of 16.20 ± 15.95 seconds. This difference stems from the fact that the *pull-based outbound oracle* reads historical states from the blockchain, whereas the *push-based outbound oracle* requires a transaction to be included – which is subject to high variance and an average delay of roughly 1.5 inter-block times [26]. This transaction triggers the event that is picked up by the *push-based outbound oracle*. We received 75% of the *pull-based outbound oracle* transactions within 0.12 seconds. For the *push-based outbound oracle*, instead, the third quartile amounts to 21.44 seconds. From the box plots in Fig. 11, we can observe that the dt measurements of the *pull-based outbound oracle* and the *push-based inbound oracle* have a significant number of outliers and follow a long-tail distribution. This is less pronounced for the other two oracles. Discounting outliers, the dt distribution for the *pull-based inbound oracle* is similar to *push-based inbound oracle*, with mean dts of 0.52 ± 0.05 and 0.53 ± 0.08, respectively, and the same minimum (0.46) and median (0.50) values. They differ slightly in their 25th (0.48 vs. 0.49) and 75th (0.52 vs 0.54) percentiles.

For *push-based inbound oracle* and *pull-based inbound oracle* we measured the transaction costs in Ether, and converted them to Euros with the above-mentioned exchange rate. The results are reported in Table 6. The gas price setting in our setup relied on the current market price – which turned out to be highly variable on Ropsten, and not representative of the Ethereum mainnet. To give an indication of the cost we would have incurred on the mainnet, we retrieved the approximate median gas price from the Google BigQuery public database of Ethereum for the period in question, which was 8.5 Gwei (averaged over 3.15 million transactions). If we multiply this with our mean gas consumption and the exchange rate, we get a median transaction cost of 0.028 € for *push-based inbound oracle* and 0.056 € for *pull-based inbound oracle*.

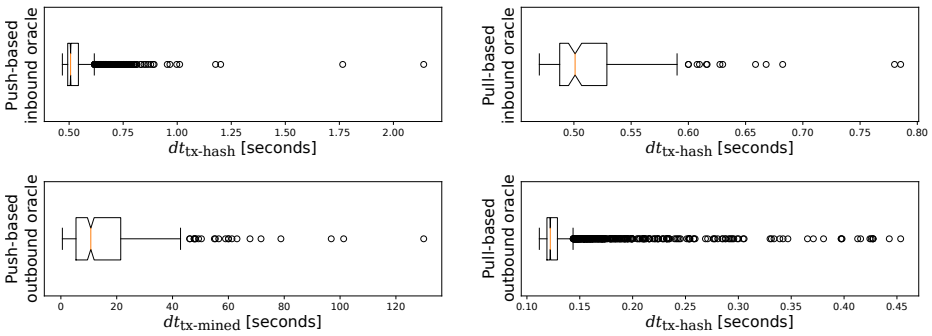

Fig. 11. Performance plots for the four oracle implementations.

6 Discussion and Threats to Validity

In the following, we discuss advantages and disadvantages, our experience from the implementation process, the results analysis above, and finally the limitations and threats to validity of this work. An advantage of the foundational viewpoint taken in this paper is the clear separation and composition of concerns we can achieve. For example, our implementation, following the patterns in this paper, enables us to implement logic for distinct abstraction levels. As such, it is possible to implement behaviour for all oracles. More crucially, adding or changing information sources to the oracle only requires us to revise the sole oracle without the need to change the on-chain implementation logic.

Regarding the results of the analysis, we find that latency and cost are both not particularly high. For instance, when comparing the latency with results from [20], where the median commit time of transactions was around 200 seconds, it is fair to say that the sub-second latency measured in almost all cases (where no transaction inclusion time is part of the latency) is relatively low. This, however, may be different if other blockchain platforms or consensus algorithms are used.

As for cost, we found that a single interaction of either inbound oracle did not incur high fees. For the fairest possible comparison, gas consumption should be used as a metric as it does not depend on current market prices. Comparing the results on this basis, in [11] (a cost-optimized version of [21]) transactions have a typical gas consumption of 24,000 to 27,000 gas. This is in line with the *pull-based inbound oracle*'s gas consumption; for the *push-based inbound oracle*'s gas usage the additional storage cost accounts for the higher gas cost. Specific implementations of this pattern can be optimized in this regard, in particular by storing data on-chain only when necessary. This may be particularly important when many oracle invocations are expected in a given setting, and cost and time delays would add up.

The work we present in this paper has a number of limitations and threats to validity. The patterns are mined using a qualitative mining process (as it is usual). Thus, possible misinterpretations or biases of individual researchers or the whole author team cannot be fully excluded and might have influenced our results. Generalizability can only be claimed for the studied technologies (see Section 2), but we aimed to define foundational patterns to mitigate

this threat as far as possible. Therefore, despite our implementation resorts on Ethereum, our findings are applicable to other blockhain platforms. Nevertheless, we do not claim any form of completeness. Our analyses are preliminary and can only provide a rough indication of time performance and costs; for claiming generalizability beyond the scope of the studied cases, more research would be needed. Furthermore, the use of a testnet like Ropsten may reduce the representativeness of the analysis results for practical applications. We mitigated these effects by not relying on time and cost measurements from the testnet in our discussion, and by basing relevant cost analyses on data from the Ethereum mainnet instead. In future work, we will also study different strategies on data structures and message rates to further mitigate the impact that information exchanges have on the overall execution costs.

7 Conclusion

In this paper, we have investigated how blockchain oracles can be characterized for the communication between the on-chain and off-chain realms. We abstract individual technical solutions adopted in existing implementations into four foundational oracle patterns. In addition, we have studied their relations, benefits, liabilities, and consequences. Finally, we have quantitatively analysed the four patterns in terms of time performance (latency) and cost impacts. We find that neither cost nor latency are particularly high for a single invocation of any of the patterns, except that latency can be dominated by transaction inclusion time. Also, in our experiments the patterns were in most cases subject to different distributions in terms of cost and latency; the results show these characteristic differences.

In future research, we will deepen our analysis with further studies conducted on multiple blockchain platforms, further study how exchanged data rate and quantity has an impact on execution costs, and apply the patterns to more use cases spanning over different fields including autonomous robotic swarm systems [19]. Furthermore, we want to study the use of patterns for information exchange between blockchains. The combination of oracle patterns would also be the subject of our future studies.

Acknowledgements. The authors want to thank the Research Institute for Computational Methods at WU Vienna for supplying computational resources. E. Castelló Ferrer acknowledges support from the Marie Skłodowska-Curie actions (EU project BROS—DLV-751615). The work of C. Di Ciccio was partly supported by MIUR under grant "Dipartimenti di eccellenza 2018-2022" of the Department of Computer Science at Sapienza University of Rome.

References

1. Abou El Houda, Z., Hafid, A., Khoukhi, L.: Co-IoT: A collaborative ddos mitigation scheme in IoT environment based on blockchain using SDN. In: IEEE GLOBECOM. pp. 1–6 (2019)
2. Adler, J., Berryhill, R., Veneris, A.G., Poulos, Z., Veira, N., Kastania, A.: Astraea: A decentralized blockchain oracle. In: iThings/GreenCom/CPSCom/SmartData. pp. 1145–1152. IEEE (2018)
3. Ahmed, S., ten Broek, N.: Blockchain could boost food security. Nature **550**(7674), 43–43 (2017)
4. Beniiche, A.: A study of blockchain oracles. arXiv preprint arXiv:2004.07140 (2020)

5. Chen, J., Micali, S.: Algorand: A secure and efficient distributed ledger. Theor. Comput. Sci. **777**, 155–183 (2019)
6. Delgado-Mohatar, O., Sierra-Cámara, J.M., Anguiano, E.: Blockchain-based semi-autonomous ransomware. Future Generation Computer Systems (2020)
7. Dhillon, V., Metcalf, D., Hooper, M.: The Hyperledger project. In: Blockchain enabled applications, pp. 139–149. Springer (2017)
8. Di Ciccio, C., Cecconi, A., Dumas, M., García-Bañuelos, L., et al.: Blockchain support for collaborative business processes. Informatik Spektrum **42**, 182–190 (May 2019)
9. Di Ciccio, C., Meroni, G., Plebani, P.: Business process monitoring on blockchains: Potentials and challenges. In: BPMDS/EMMSAD@CAiSE. pp. 36–51. Springer (2020)
10. Eberhardt, J., Tai, S.: On or Off the Blockchain? Insights on Off-Chaining Computation and Data. In: European Conference on Service-Oriented and Cloud Computing (ESOCC) (2017)
11. García-Bañuelos, L., Ponomarev, A., Dumas, M., Weber, I.: Optimized execution of business processes on blockchain. In: BPM. Springer (2017)
12. Glicksberg, B.S., Burns, S., Currie, R., Griffin, A., Wang, Z.J., Haussler, D., Goldstein, T., Collisson, E.: Blockchain-authenticated sharing of genomic and clinical outcomes data of patients with cancer: A prospective cohort study. Journal of medical Internet research **22**(3), e16810 (2020)
13. Heiss, J., Eberhardt, J., Tai, S.: From oracles to trustworthy data on-chaining systems. In: 2019 IEEE International Conference on Blockchain (2019)
14. ISO/TC 307: ISO/TR 2345 blockchain and distributed ledger technologies – overview of and interactions between smart contracts in blockchain and distributed ledger technology systems. Tech. rep., ISO (2019)
15. Krejci, S., Sigwart, M., Schulte, S.: Blockchain- and IPFS-based data distribution for the internet of things. In: Eur. Conf. Service-Oriented and Cloud Computing. pp. 177–191 (2020)
16. Mendling, J., Weber, I., van der Aalst, W.M.P., et al.: Blockchains for business process management - challenges and opportunities. ACM Trans. Management Inf. Syst. **9**(1), 4:1–4:16 (2018)
17. Neidhardt, N., Köhler, C., Nüttgens, M.: Cloud service billing and service level agreement monitoring based on blockchain. In: EMISA. CEUR Workshop Proc., vol. 2097, pp. 65–69 (2018)
18. de Pedro Crespo, A.S., Levi, D., García, L.I.C.: Witnet: A decentralized oracle network protocol. CoRR **abs/1711.09756** (2017)
19. Strobel, V., Castelló Ferrer, E., Dorigo, M.: Blockchain technology secures robot swarms: A comparison of consensus protocols and their resilience to byzantine robots. Frontiers in Robotics and AI **7**, 54 (2020)
20. Weber, I., Gramoli, V., Staples, M., Ponomarev, A., Holz, R., Tran, A., Rimba, P.: On availability for blockchain-based systems. In: IEEE Intl. Symp. Reliable Distributed Systems (SRDS) (2017)
21. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: BPM. pp. 329–347. Springer (2016)
22. Wong, D.R., Bhattacharya, S., Butte, A.J.: Prototype of running clinical trials in an untrustworthy environment using blockchain. Nature communications **10**(1), 1–8 (2019)
23. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger (2014)
24. Xu, X., Pautasso, C., Zhu, L., Gramoli, V., Ponomarev, A., Tran, A.B., Chen, S.: The blockchain as a software connector. In: WICSA. pp. 182–191. IEEE Computer Society (2016)
25. Xu, X., Pautasso, C., Zhu, L., Lu, Q., Weber, I.: A pattern collection for blockchain-based applications. In: EuroPLoP. pp. 3:1–3:20. ACM (2018)
26. Xu, X., Weber, I., Staples, M.: Architecture for Blockchain Applications. Springer (2019)
27. Zhang, F., Cecchetti, E., Croman, K., Juels, A., Shi, E.: Town crier: An authenticated data feed for smart contracts. In: ACM Conf. Computer and Communications Security. pp. 270–282 (2016)