Can We Effectively Use Smart Contracts to Stipulate Time Constraints?

Tobias Eichinger  
*Service-centric Networking*  
Technische Universität Berlin  
Berlin, Germany  
tobias.eichinger@tu-berlin.de  
0000-0002-8351-2823

Marcel Ebermann  
*Service-centric Networking*  
Technische Universität Berlin  
Berlin, Germany  
marcel.ebermann@tu-berlin.de

Abstract—Smart contracts provide the means to stipulate rules of interaction between mutually distrustful organizations. They encode contractual agreements on the basis of source code, which else need to be contractualized in natural language. While the mediation of contractual agreements via smart contracts is seamless in theory, it requires that the conditions of an interaction are accurately made available in the blockchain. Time is a prominent such condition. In the paper at hand, we empirically measure the consistency of a smart contract to yield equal results on the basis of the time of an interaction and its potentially inaccurate representation in the blockchain. We propose a novel metric called execution accuracy to measure this consistency. We specifically measure the execution accuracy of a time interval-constrained smart contract that executes distinct logic within and without some constraint interval. We run experiments for the local Ganache and Quorum and the public Görli and Rinkeby Ethereum blockchains. Our experiments confirm our intuition that execution accuracy decreases near interval bounds. The novelty of our proposed metric resides in its capacity to quantify this decrease. We demonstrate how time constraints can be effectively stipulated on the basis of execution accuracy measurements.

Index Terms—time-sensitive smart contract, execution accuracy, time injection, block timestamp method, injection accuracy, time interval constraint

I. INTRODUCTION

Smart contracts provide the means to stipulate rules of interaction between mutually distrustful organizations. They have been proposed to execute business [1], [2] and manufacturing [3], [4] processes seamlessly across multiple organizations. Although conceptually seamless, a practical difficulty is the accurate time injection into a blockchain, where we denote by time injection the act of making world time available for smart contract execution [5]. Consider for instance a smart contract that periodically updates a variable holding the current world time as a UNIX timestamp. As part of the blockchain state, world time can now be used within smart contracts.

Time is a key aspect of business and manufacturing processes [6], [7]. Consider for instance two manufacturing activities that require a certain time delay. Only if the start and end time of an activity are accurately injected into a blockchain, delays and time constraints can be correctly coordinated. In particular, coordinating an entire process requires time to be injected not only once but continuously throughout the process. We denote by continuously injecting time into a blockchain as time tracing. In brief, process execution depends on time tracing, which in turn depends on time injection.

Ladleif and Weske [8] survey five distinct time injection methods for use in blockchains. They conclude that there is no objectively best time injection method. The so-called parameter method is ideal for scenarios in which senders of transactions are trustworthy. Here, senders of a transaction truthfully attach the current world time to a transaction. World time and injected time are hence trivially identical. In this case, the result of executing smart contract logic on the basis of world time and injected time is identical.\(^1\) In contrast, if world time and injected time are not equivalent, execution may yield inaccurate results due to inaccurate time injection. Figure 1 illustrates inaccuracies caused by inaccurate time injection.

We denote by execution accuracy the probability that the execution of smart contract logic on the basis of world time and injected time yield identical results. The parameter method mentioned in the previous paragraph yields perfect execution accuracy. However, it is only useful in scenarios in which

\(^1\)Here, we assume that execution logic is independent from other transactions calling the smart contract.
sends of transactions are trustworthy. If senders are not trustworthy, the block timestamp method is an alternative time injection method [8]. Here, world time is injected by the miner instead of by the sender. World time is injected as the block timestamp of the block the calling transaction is mined into.

We denote by injection accuracy the offset between world time and injected time. The injection accuracy of the block timestamp method then depends on for instance (a) the block time, that is the difference in timestamps of consecutive blocks, (b) the mining fee offered to the miners, and (c) the miners themselves who set block timestamps. While injection accuracy for the block timestamp method is well-understood [9], [10], execution accuracy is not. This is due to the fact that injection accuracy solely depends on the network properties of the blockchain, while execution accuracy additionally depends on the specific execution logic of smart contracts.

In summary, coordinating process execution in a blockchain is only meaningful if the result of calling a smart contract is consistent over world time and injected time. Injection accuracy cannot fully describe such consistency, since injection accuracy is independent from smart contract execution logic. In order to address this gap, we propose the following contributions:

- We introduce execution accuracy as a metric that measures the consistency with which time-sensitive smart contract execution on world time and injected time match.
- We measure execution accuracy of a time interval-constrained smart contract with respect to the block timestamp method. Measurements are made on two local Ethereum implementations Ganache\(^2\) and Quorum\(^3\) and the Görli\(^4\) and Rinkeby\(^5\) Ethereum test networks.
- We share the code of our test bed with the community.

II. RELATED WORK

We first review literature that studies time injection independently from the execution logic of smart contracts with respect to their injection delay and injection accuracy. Afterwards, we tend to application-specific aspects of time injection.

A. Accuracy and Latency of Time Injection

Blockchains can only validate whether time was injected correctly, that is formally correctly according to the blockchain protocol. Measuring injection accuracy is infeasible from within a blockchain. Injection accuracy needs to be measured outside of it. In particular for time injection via so-called oracle contracts, measuring injection accuracy can be translated into a trust issue toward the stakeholder that injects time [5], [9]. Roughly, injected time can be assumed to be accurate if the stakeholder injecting time is trustworthy. Applications typically do require continuous injection of time instead of single instances of time injection. We tend to prior work in applications that continuously inject time.

B. Time Injection in Applications

Tracing digitally on a blockchain what happens physically in the world is key to collaborative manufacturing processes [3], [11], [12]. This is because smart contracts cannot immediately access the state of the physical world. In particular, they cannot immediately access world time. Consider for instance a manufacturing process that requires a forged building component to cool down before it can be relocated to the assembly line. Either the temperature of the building component or the duration of its cooling process can be traced in the blockchain such that the manufacturing process can be executed correctly. Using a digital model that traces the state of a physical product is generally denoted using a digital twin [13].

While collaborative manufacturing processes mainly trace the state of a product, collaborative business processes typically trace interactions between stakeholders [6]. The business process itself is executed in a process engine [14]. Here, smart contracts define conditions, particularly temporal conditions, for interactions and contractual agreements between stakeholders [6], [7]. Consider for instance a smart contract that sets a deadline only before which a cooled down building component is accepted for relocation. The relocation service provider may then refuse to relocate a building component when the smart contract evaluates that the deadline has passed.

C. Time Tracing and Time Synchronization

Time tracing is distinct from time synchronization. Time synchronization aims to equalize offsets in distinct world time measurements [15]. In contrast, time tracing is about continuously persisting references to world time in a blockchain. Since blockchains establish a concept of time that is distinct from world time [16], time tracing cannot be considered synchronization as no offsets are harmonized. It is important to see that block timestamps represent references to world time, they are not themselves world time timestamps.

We now explain how block timestamps as references to world time timestamps can be injected into a blockchain and used to stipulate time interval constraints in smart contracts.

III. CONCEPT

We formalize the block timestamp method. We then define a prototypical time interval-constrained smart contract. Finally, we present our experimental test bed and describe how it can be used to measure execution accuracy.

A. The Block Timestamp Method

We first introduce some notation. Let \( B = (B_i)_{i \in \mathbb{N}} \) be an infinite sequence of blocks that represent a blockchain. Every block \( B_i \) holds a sequence of transactions. Transactions in a block have been found valid by a miner,\(^6\) where valid means that the state transitions the transactions induce conform with the underlying blockchain protocol. Any finite subsequence

\(^2\)https://trufflesuite.com/ganache/ (accessed 5 May 2022)
\(^3\)https://consensys.net/quorum/ (accessed 5 May 2022)
\(^4\)https://goerli.net/ (accessed 5 May 2022)
\(^5\)https://www.rinkeby.io/ (accessed 5 May 2022)

\(^6\)The block timestamp method is not limited to proof-of-work consensus mechanisms as the term miner suggests. It can be applied equally well to for instance proof-of-stake consensus. We generally denote by miner the entity that attaches the block timestamp to the block.
of blocks \(B_i\) \(1 \leq i \leq N\) hence represents a valid state of the blockchain \(B\). We associate every block \(B_i\) with a timestamp \(t_i \in \mathbb{R}_{\geq 0}\) such that \(t_i < t_j\) for all \(i < j\). Then blockchain \(B\) is associated with an infinite and strictly increasing sequence \(t = (t_i)_{i \in \mathbb{N}}\) of so-called block timestamps. Block timestamps are not essential to blockchains, yet convenient since they reference when a block was mined.

We formalize the block timestamp method as described for instance in [8]. Let \(x\) be a transaction sent at world time \(t_x\) that calls some time-sensitive smart contract \(C\). We denote by \(\hat{t}_x\) the injected time of world time \(t_x\). Let further \(B_i\) be the latest block and \(t_i\) its block timestamp. Then \(x\) is communicated to miners within the blockchain network. Miners determine a valid next block \(B_{i+1}\) that includes \(x\) and is associated to some block timestamp \(t_{i+1} > t_i\). Particularly, the block timestamp method sets \(t_{i+1} = t_x\). The reference time used to execute the time-sensitive smart contract \(C\) is exactly the block timestamp. We now present our experimental test bed in which time-sensitivity represents a time interval constraint.

### B. Time Interval-Constrained Smart Contract

We define a prototypical time-sensitive smart contract that implements an interval time constraint. Similarly to the example shown in Figure 1, the prototypical time-sensitive smart contract then executes distinct behavior within and without some reference time interval \(I\). Formally, let \(C\) be a smart contract initialized with some interval \(I = [a, b] \subseteq \mathbb{R}_{\geq 0}\) and two binary state variables \(p\) and \(\hat{p}\). Let further \(x\) be some transaction submitted to the blockchain network at world time \(t_x\) and included into a block \(B_i\) with block timestamp \(t_i\). Then \(C\) sets its state variable \(p = 1\) if world time \(t_x\) satisfies the time constraint \(t_x \in I\), and else \(p = 0\) \((t_x \notin I\)). Analogously, we set \(\hat{p} = 1\) and \(\hat{p} = 0\) for injected time \(\hat{t}_x\).

Table I shows the four possible states defined by the binary state variables \(p\) and \(\hat{p}\). In analogy to binary classification in machine learning, we associate the state variable \(p\) with the true condition of world time. From the perspective of the blockchain, we interpret injected time as a prediction of the true world time when transaction \(x\) was sent. We associate the state variable \(\hat{p}\) with the predicted condition of world time. Table I thus technically represents a confusion matrix.

Recall that execution accuracy measures the consistency with which a smart contract yields the same result with reference to world time and injected time. In other words, it measures the probability that the smart contract \(C\) enters either a true positive or true negative state. We now present the test bed we use to measure this probability.

### C. Test Bed

We empirically estimate the probability that a transaction \(x\) submitted to the blockchain network at world time \(t_x\) either yields a true positive or true negative smart contract state. Since the state of the smart contract \(C\) depends on world time \(t_x\) and injected time \(\hat{t}_x\), we need to make both available at execution time. We use the parameter method (see Section I) to make world time available and choose another time injection method such as the block timestamp method (see Section III-A) to make injected time available. The smart contract \(C\) can then determine its state after being called by a transaction.

In order to estimate probabilities for each of the four possible states to occur, we call the smart contract at regular intervals by sending a transaction to it. We send a total of \(N\) transactions over an experiment interval \(J \supset I\) that includes the constraint interval \(I = [a, b]\) of the time-sensitive smart contract \(C\). Note that sending a transaction to \(C\) overwrites any previous state of \(C\). In order to persist all state duples over the course of the experiment, we first initialize an array of length \(n\) in the smart contract. Throughout the experiment, we then persist all \(n\) state duples \([p, \hat{p}]\) in the array. After the experiment, we fetch the results by reading the filled array from the smart contract. Figure 2 shows a schematic workflow of an experiment. It remains to formally define execution accuracy and how to read it off the array.

### D. Execution Accuracy

Accuracy is a standard metric used to evaluate the quality of a binary classifier in machine learning. It measures how well a binary classifier predicts the true class of an observation out of the two possible classes positive and negative correctly. Accuracy can be understood as the probability that a binary classifier correctly predicts the class of an observation to be positive when it is truly positive (true positive), and negative if it is truly negative (true negative). For a test set of \(N\) observations, accuracy \(A\) of a binary classifier is defined as

\[
A = \frac{\text{true positives} + \text{true negatives}}{N}. \tag{1}
\]

We now translate this accuracy metric for binary classifiers into the already outlined execution accuracy metric by estab-
Fig. 3. Relative frequencies of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for a time interval-constrained smart contract and the block timestamp method on the GoQuorum (left) and Ganache (right) local Ethereum blockchains for distinct blocktimes \( b = 1s, 4s, 8s \) (top to bottom).

establishing a link to the four distinct states of a time interval-constrained smart contract as shown in Table I.

We interpret the time-sensitive smart contract \( C \) as a binary classifier. From the perspective of the blockchain, we interpret observations as instances of injected time \( t_x \). Then, the state variable \( p \) represents the true class of an observation and the state variable \( \hat{p} \) represents the predicted class of an observation. More specifically, the state value 1 is associated with the positive class and the state value 0 is associated with the negative class. For a set \( X \) of \( n \) transactions, we define execution accuracy \( A_{\text{execution}} \) as

\[
A_{\text{execution}} = \frac{|\{x \in X | t_x, \hat{t}_x \in I \lor t_x, \hat{t}_x \notin I\}|}{n} + \frac{|\{x \in X | \text{C is true positive}\}|}{n}
\]

in analogy to accuracy \( A \) in Equation (1), where \( |\{\cdot\}| \) denotes the number of elements in the set \( \{\cdot\} \). Observe that execution accuracy depends on the constraint interval \( I \). Since \( I \) is part of the time-sensitive execution logic of \( C \), we see in particular that execution accuracy depends on \( C \)'s execution logic as desired. We now present empirical measurements of execution accuracy on four distinct Ethereum blockchains.

IV. EVALUATION

We measure execution accuracy of a time interval-constrained smart contract on the Ganache and Quorum local Ethereum blockchains and the Görli and Rinkeby Ethereum test network. We will see that execution accuracy can behave differently on else equal configuration parameters. We share the source code of our test bed with the community.\(^7\)

A. Experimental Setup

We implement a test bed following the description in Section III-C and a time interval-constrained smart contract \( C \) following the description in Section III-B. More specifically, we use constraint intervals of \( I_{\text{local}} = [15, 30] \), \( I_{\text{test}} = [60, 120] \) and experiment intervals \( J_{\text{local}} = [0, 45] \), \( J_{\text{test}} = [0, 135] \) for the local and test networks respectively, where we measure time in seconds. We inject time via the block timestamp method

\(^7\)https://github.com/marcelTUB/Execution-Accuracy-Testbed
as described in Section III-A. We make available world time for smart contract execution via the parameter method. More specifically, we attach a string representing the current world time when sending a transaction to the smart contract $C$.

For each of the four blockchains to measure, we run 30 experiments. Over the course of an experiment, we send one transaction per second. After all transactions are mined, we read the now filled array holding the state duples $[p, \hat{p}]$. Recall that $p$ and $\hat{p}$ are the two binary state variables of $C$ as shown in Table I. There, we see that $C$ enters one of four distinct states after being called by a transaction: true positive ($p = 1; \hat{p} = 1$), false positive ($p = 0; \hat{p} = 1$), true negative ($p = 0; \hat{p} = 0$), and false negative ($p = 1; \hat{p} = 0$). If the array only holds true positives and true negatives, execution accuracy is perfect (see Equation (2)).

We will see in the following sections that the relative frequency with which a calling transaction $x$ changes the state of the smart contract $C$ to any of the four states depends on the time $t_x$ of its sending. Note that the blockchains we measure use integer-valued block timestamps that represent full seconds. We therefore split results by their time of sending $t_x$ into batches of full seconds. Consider for instance the experiment interval $J = [0, 3]$. We then split results into three batches of transactions sent during the subintervals $[0, 1)$, $[1, 2)$, and $[2, 3)$ respectively. Note that sending many transactions within a short timeframe is prohibitive on most test blockchains. We therefore average results per batch over multiple rounds of experiments per blockchain.

### B. Execution Accuracy on Local Blockchain Networks

On local blockchain networks, measuring the impact of distinct parameter configurations on execution accuracy is readily feasible, as setting up a newly configured blockchain comes with little effort. We thus experimented with many distinct parameter configurations locally. We find that varying the blocktime and recommit interval length has an immediate impact on execution accuracy.

The blocktime defines the offset between block timestamps of consecutive blocks. Recomit intervals define periods of time when blockchain nodes aggregate transactions submitted to the blockchain. Aggregating transactions before they are presented to miners is usually beneficial from a cost perspective, since less candidate blocks are attempted to be validated by miners overall. Certainly, the cost aspect only holds true for non-local blockchains, while the impact of aggregation on execution accuracy pertains to local blockchains.

As expected, we find that increasing either the blocktime or the length of recommit intervals jeopardizes execution accuracy on average. Since the recommit interval length parameter is not available on the Quorum blockchain, we omit reporting detailed results on changing the recommit interval length. Instead, we report results on varying the blocktime.

Figure 3 shows relative frequencies with which the smart contract $C$ enters its four distinct states. Results are batched into batches of length 1 as described in Section IV-A. At large, both Ganache and Quorum yield similar results. We can see that the relative frequencies differ before, during, and after the constraint interval. We thus describe each segment separately.

**Before Constraint Interval ($t_x \in [0, 15)$):** To the left of the constraint interval, we only observe true negatives and false positives. Recall that execution accuracy equals the sum of the relative frequencies of true positives and true negatives (see Equation (2)). Since the relative frequency of true positives is zero in this segment, execution accuracy is immediately equal to the relative frequency of true negatives. We thus see that execution accuracy is initially perfect and deteriorates approaching the left bound of the constraint interval. A comparison of Ganache and Quorum yields that Ganache is more execution accurate than Quorum.

**During Constraint Interval ($t_x \in [15, 30)$):** Within the constraint interval, we only observe true positives and false negatives. Inversely to the segment before the constraint interval, we have that execution accuracy is equal to the relative frequency of true positives instead of true negatives. Execution accuracy generally becomes lower near the interval bound. A comparison of Ganache and Quorum yields that Quorum is more execution accurate than Ganache at the lower interval bound, yet less execution accurate at the upper interval bound.

**Past Constraint Interval ($t_x \in [30, 45)$):** To the right of the constraint interval, we observe true negatives and false positives. Similarly to the segment before the constraint interval, we again have that execution accuracy equals the relative
frequency of true negatives. Execution accuracy is lower in
time batches near the upper interval bound and increases to
perfect execution accuracy. A comparison of Ganache and
Quorum yields that Quorum is more execution accurate than
Ganache.

In summary, we see that execution accuracy is generally
perfect, yet decreases near interval bounds. This decrease
is asymmetric, that is execution accuracy is higher past an
interval bound than before. The extent of this discrepancy
depends largely on the blocktime, yet also on the blockchain
implementation at hand. We now measure the two test net-
works Rinkeby and Goerli that we cannot configure at will.

C. Execution Accuracy on Test Networks

Figure 4 shows relative frequencies for the Rinkeby and
Görlı test networks. We see that the behavior of relative
frequencies resembles that measured on local blockchains at
large. Observe however that the Görlı test network exhibits
a lengthy period of false negatives in the interval \([0,30]\). An
analysis yields that this is due to the Görlı network sometimes
having unexpectedly long response times. Transactions are
sent long before the constraint interval, yet still found to
satisfy the time interval constraint as they are presented and
mined belatedly during the time of the constraint interval. We
conclude that the Rinkeby network is more execution accurate
than the Görlı network on average.

So far, we have measured the execution accuracy at the
interval bounds of a time interval-constrained smart contract.
It remains to discuss how mutually distrustful organizations
can leverage these measurements to effectively stipulate time
constraints via smart contracts in practice. We address the
question put forth in the paper title: Can we effectively use
smart contracts to stipulate time constraints?

V. DISCUSSION

Smart contracts can be used as monitors or mediators
to support a collaborative process [2]. We delineate both
usage patterns and discuss under which circumstances time
constraint stipulation does and does not fail.

A. Monitoring Smart Contracts

An example of a monitoring smart contract is shown in
Figure 1. Papers are submitted by sending a transaction to the
monitoring smart contract. The smart contract evaluates the
validity of its time constraint, that is it evaluates whether pa-
pers are submitted before the deadline. Technically, it monitors
paper submissions by logging their injection times.

The validity of a submission only depends on its injection
time. It does not necessarily depend on when it was submitted.
Consider for instance the parameter method in which time is
injected as a timestamp attached to a transaction by its sender
(see Section I). Papers submitted after the deadline will still
be found valid by the smart contract if the timestamp attached
to the transaction is sufficiently dated back by the sender.

B. When Does Monitoring Time Constraint Stipulation Fail?

Time constraint stipulation in monitoring smart contracts
fails if the expected behavior of the smart contract, executed
with respect to the time a transaction is sent, does not meet the
actual behavior of the smart contract, executed with respect
to the associated injected time of the transaction. Hence by
definition, execution accuracy captures the probability that the
stipulation of time interval constraints in monitoring smart
contracts is successful (see Section III-D). With reference
to the paper submission monitoring smart contract, authors
expect that their submissions are always valid if submitted
before the deadline. Therefore, time constraint stipulation fails
if either timely submissions are found invalid (false negative),
or belated submissions are found valid (false positive).

We have seen in the previous subsection that the validity of a
submission is assessed on the basis of its injected time and not
on the basis of the actual time it was submitted. The choice of
the underlying time injection method hence has an impact on
the effectiveness with which time constraints can be stipulated.
For instance, stipulating time constraints in monitoring smart
contracts on the basis of the parameter method will fail if
authors inaccurately or maliciously attach false timestamps
to their submissions. Conversely, stipulation will not fail if
authors accurately and truthfully attach their submission times.

In order to understand when time constraint stipulation fails
under the block timestamp method (see Section III-A), we
consult the execution accuracy measurements shown in Figure
4. For the sake of the argument, we assume that the monitoring
smart contract is deployed in the Rinkeby blockchain. We thus
associate the paper submission deadline with the right bound
of the constraint interval shown in the left of Figure 4. There,
we see that papers submitted 10 seconds prior to the deadline
\((110 \leq t < 120)\) have a probability of higher than 60% that
the smart contract falsely evaluates the submission to be invalid
(see FN). We see that the stipulation of time constraints in
monitoring smart contracts on the basis of the block timestamp
method will fail near interval bounds, and is thus ineffective.

C. Mediating Smart Contracts

An example of a mediating smart contract is mentioned in
Section II-B. In the context of a collaborative manufacturing
process, some organization \(A\) forges a building component,
which, only after it has cooled down, another organization \(B\)
may relocate to the assembly line. Both organizations stipulate
this temporal constraint in a mediating smart contract. More
specifically, the smart contract mediates the end of the cooling
period between the organizations via its binary state variable
\(\hat{p} \in \{ 'hot', 'cool' \}\). We assume that the state variable \(\hat{p}\)
is initialized as 'hot'. State transition to 'cool' is triggered by
any transaction \(x\) sent to the smart contract that is associated
with injected time \(I_x \geq T\), where \(T\) marks the end of the
cooling period. Once organization \(B\) reads that \(\hat{p} = 'cool'\), it
initiates relocation of the building component.

The difference between monitoring and mediating smart
contracts resides in what the smart contract state \(\hat{p}\) refers to.
In monitoring smart contracts, \(\hat{p}\) refers to the world state \(p\) of
an object, such as a submitted paper, at the time of sending a transaction. Contrarily in mediating smart contracts, \( \hat{p} \) refers to the world state of an object, such as a building component, at the current time of manufacturing. For the paper submission use case, one is interested in the validity of the time constraint at the time of submitting a paper, while for the collaborative manufacturing use case, one is interested in the validity of the time constraint at the current time of manufacturing.

Assume for instance that a blockchain uses \( n \) confirmation blocks. A state transition in the mediating smart contract from \( \hat{p} = \text{'hot'} \) to \( \text{'cool'} \) in the most recent block will then have been triggered by a transaction sent before \( n \) blocktimes. Notably, even if a transaction were to trigger a state transition to \( \hat{p} = \text{'cool'} \) prematurely, thus violating the time constraint at the time of sending the transaction, the manufacturing process may still not fail. It does not fail if for instance the building component has cooled down by the time the state transition to \( \hat{p} = \text{'cool'} \) is confirmed in a block. Therefore, stipulation of time constraints in mediating smart contracts is subject to state transitions of \( \hat{p} \) presented in the most recent block. It is not subject to the time when state transitions of \( \hat{p} \) are triggered.

D. When Does Mediating Time Constraint Stipulation Fail?

Time constraint stipulation in mediating smart contracts fails if the collaborative process fails due to a misrepresentation of world state in the blockchain. With reference to the collaborative manufacturing mediating smart contract, time constraint stipulation fails if the building component is moved by organization \( B \) before it has cooled down. More specifically, time constraint stipulation fails if the current smart contract state \( \hat{p} \), presented in the most recent block, indicates that the building component has cooled down, while in fact it has not cooled down (false positive), or the current smart contract state \( \hat{p} \) indicates that the building component is hot, while in fact it has cooled down (false negative).

We have seen in the previous subsection that the smart contract state \( \hat{p} \) refers to the current state of the building component. In contrast to monitoring smart contracts, it does not refer to the state of the building component at the time of sending a transaction. Thus for mediating smart contracts, execution accuracy quantifies the probability that time constraint stipulation is successful only if state transitions are persisted in the blockchain immediately after they are triggered by a transaction. In practice, this is certainly never given. Blockchains naturally introduce persistence delays due to for instance properties of the consensus mechanism such as block proposition, block creation, and block propagation. These implementation-specific delays need to be accounted for.

Persistence delays are not affected by the choice of the underlying time injection method. The underlying time injection method only has an impact on the state transition of \( \hat{p} \). Therefore, under the block timestamp method, execution accuracy is perfect if we assume that block timestamps are accurately and truthfully attached by miners. Execution accuracy is perfect, since the miner accurately and truthfully fetches the current world time on the basis of which he accurately evaluates the validity of the time constraint. We have seen in the previous paragraph that perfect execution accuracy only implies effective time constraint stipulation if there are no persistence delays. We argue in the following that the stipulation of time constraints in mediating smart contracts can be generally made effective when time delays are constant.

We distinguish two cases. We first assume that there are no persistence delays \((c = 0)\), and afterwards that persistence delays are constant and positive \((c > 0)\). In the first case, stipulation of time constraints is effective following the argumentation in the previous subsection. We thus more generally assume that persistence delays are constant and positive \((c > 0)\). In this case, time constraint stipulation is not effective, since state transitions of \( \hat{p} \) come at a constant delay of \( c \). However, time constraint stipulation can be made effective by shifting the constraint interval by the constant length of persistence delays \((1 = [a - c, b - c])\). Certainly, persistence delays are not constant in practice and vary over time. We thus argue that the stipulation of time constraints in mediating smart contracts will only be effective if the variance of persistence delays can be kept sufficiently small.

VI. Conclusion

The stipulation of time constraints via smart contracts on blockchains is inherently inaccurate. To date, this inaccuracy has only been characterized as the result of protocol and network delays and latencies irrespective of smart contract execution logic. We extend this characterization by proposing execution accuracy, a novel metric that quantifies this inaccuracy solely on the basis of smart contract execution logic.

We specifically study interval time-constrained smart contracts that execute distinct logic within and without a pre-defined time interval. Here, execution accuracy describes the probability that the smart contract yields the same result on the basis of the time of sending a transaction to it, and when the transaction is included into a block. Our analysis confirms that execution accuracy decreases near interval bounds and extends prior work by quantifying this decrease.

The scope of the paper at hand is limited in three aspects. First, we only study absolute constraint intervals, that is constraint intervals that have fixed bounds. The study of execution accuracy for dynamic interval bounds is due. Second, we only study stateless execution logic, that is execution logic that is independent from the state of a smart contract. The study of stateful execution logic is due. Third, we only study the block timestamp time injection method. In particular, measuring execution accuracy of oracle-based time injection is due.

Acknowledgements

This research is carried out in the frame of the TRAPEZE project, funded by the European Union’s Horizon 2020 research and innovation programme under grant agreement N. 883464. The authors kindly thank the anonymous reviewers.
for their thoughtful and detailed feedback that significantly helped to improve the quality of the paper at hand.

REFERENCES

[1] J. Mendling, I. Weber, W. V. D. Aalst, J. V. Brocke, C. Cabanillas, F. Daniel, S. Debois, C. D. Ciccio, M. Dumas, S. Dustdar, A. Gal, L. García-Bañuelos, G. Governatori, R. Hull, M. L. Rosa, H. Leopold, F. Leymann, J. Recker, M. Reichert, H. A. Reijers, S. Rinderle-Ma, A. Soiti, M. Rosenmann, S. Schulte, M. P. Singh, T. Slaats, M. Staples, B. Weber, M. Weidlich, M. Weske, X. Xu, and L. Zhu, “Blockchains for business process management - challenges and opportunities,” ACM Trans. Manage. Inf. Syst., vol. 9, no. 1, pp. 1–16, 2018.

[2] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling, “Untrusted business process monitoring and execution using blockchain,” in Business Process Management, Springer, 2016, pp. 329–347.

[3] C. P. Nielsen, E. R. da Silva, and F. Yu, “Digital twins and blockchain – proof of concept,” Procedia CIRP, vol. 93, pp. 251–255, 2020, 53rd CIRP Conference on Manufacturing Systems 2020.

[4] M. Westerkamp, F. Victor, and A. Küpper, “Tracing manufacturing processes using blockchain-based token compositions,” Digital Communications and Networks, vol. 6, no. 2, pp. 167–176, 2020.

[5] R. Mühlberger, S. Bachhöfer, E. Castelló Ferrer, C. Di Ciccio, I. Weber, M. Wöhrer, and U. Zdun, “Foundational oracle patterns: Connecting blockchain to the off-chain world,” in Business Process Management: Blockchain and Robotic Process Automation Forum. Springer, 2020, pp. 35–51.

[6] A. Abid, S. Cheikhrouhou, and M. Jmaiel, “Modelling and executing time-aware processes in trustless blockchain environment,” in Proceedings of the 14th International Conference on Risks and Security of Internet and Systems. Springer, 2020, pp. 325–341.

[7] S. Cheikhrouhou, S. Kallel, N. Guermouche, and M. Jmaiel, “The temporal perspective in business process modeling: a survey and research challenges,” Service Oriented Computing and Applications, vol. 9, pp. 75–85, 2015.

[8] J. Ladleif and M. Weske, “Time in blockchain-based process execution,” in Proceedings of the 24th IEEE International Enterprise Distributed Object Computing Conference. IEEE, 2020, pp. 217–226.

[9] J. Ladleif, I. Weber, and M. Weske, “External data monitoring using oracles in blockchain-based process execution,” in Business Process Management: Blockchain and Robotic Process Automation Forum. Springer, 2020, pp. 67–81.

[10] R. Yasaweerasinghelage, M. Staples, and I. Weber, “Predicting latency of blockchain-based systems using architectural modelling and simulation,” in Proceedings of the 2017 IEEE International Conference on Software Architecture. IEEE, 2017, pp. 253–256.

[11] C. Mandolla, A. M. Petruzelli, G. Percoco, and A. Urbinati, “Building a digital twin for additive manufacturing through the exploitation of blockchain: A case analysis of the aircraft industry,” Computers in Industry, vol. 109, pp. 134–152, 2019.

[12] B. Putz, M. Dietz, P. Empl, and G. Pernul, “Ethertwin: Blockchain-based secure digital twin information management,” Information Processing & Management, vol. 58, no. 1, 2021, article 102425.

[13] E. VanDerHorn and S. Mahadevan, “Digital twin: Generalization, characterization and implementation,” Decision support systems, vol. 145, 2021, article 113524.

[14] O. López-Pintado, L. García-Bañuelos, M. Dumas, I. Weber, and A. Ponomarev, “Caterpillar: A business process execution engine on the ethereum blockchain,” Software: Practice and Experience, vol. 49, no. 7, pp. 1162–1193, 2019.

[15] E. Regnath, N. Shivaraman, S. Shrejith, A. Easwaran, and S. Steinhorst, “Blockchain, what time is it? trustless datetime synchronization for iot,” in Proceedings of the 2020 International Conference on Omni-layer Intelligent Systems. IEEE, 2020, pp. 1–6.

[16] M. Swan, “Blockchain temporality: Smart contract time specifiability with blocktime,” in Rule Technologies. Research, Tools, and Applications. Springer, 2016, pp. 184–196.