LTO.network

Blockchain for Decentralized Workflows

www.lto.network

Abstract

Digitalization and automation of business processes offer great benefits in terms of productivity and cost reduction. Organizations struggle to tap into these benefits for inter-organizational processes, partly due to a lack of trust. Bitcoin has proven how blockchain can use distribution and cryptography to provide a system that does not rely on trust.

LTO builds upon this foundation with a decentralized workflow engine employed for ad-hoc collaboration. Information is shared between parties using per-process private event chains and hashed on a public blockchain. This hybrid approach allows organizations to meet any data protection regulations and prevents scalability issues that are typically associated with blockchain projects.

INTRODUCTION

The digital revolution has resulted in many changes that make our lives more efficient[1]. This wave of progress has taken place primarily in consumer-facing and internal business processes. When it comes to inter-organizational processes we have to acknowledge that the changes are less drastic. Faxing has largely been replaced by e-mail, and the typewriter is replaced by a word processor, but beyond these superficial changes, execution of the underlying processes have hardly changed.

The first reason that automation has been absent is the reluctance of corporations to rely on external systems operated by a counterparty[2], as the distribution of information plays an important role in the outcome of a relationship[3]. Where there is a natural power imbalance, one party may take control, forcing all others to use its centrally managed system. We see this when dealing with the government and to some extent with corporations. In a situation where no single party can claim control, automation simply doesn't happen[4].

To achieve automation for inter-organizational processes, for over two decades people have experimented with decentralized workflows[5]. In these studies and experiments, a high level of trust and fair play is assumed, focusing primarily on solving technological challenges. In reality, this is a false assumption as a lack of trust prevents successful pilots from making it to production.

A second reason for the absence of automation is the correlation between efficiency and corruption[6]. Traditionally large corporations and government bodies require a large number of people to execute a process. A fair amount of bureaucracy is required to coordinate such processes. This increases the costs of bribery, reducing the incentive to automate. Increasing efficiency negates this effect.

This paper shows how both problems may be solved using blockchain, providing a solution where all parties can be on equal footing.

CONTENTS

Part I. Live Contracts 3

1 Live vs Smart Contracts 3
- 1.1 Ricardian Contracts . . . 3
- 1.2 Enforcement . . . 3
- 1.3 User interface . . . 3

2 Finite state machine 3
- 2.1 Deterministic Finite State Machine . . . 4
- 2.2 Extended Finite State Machine . . . 4
- 2.3 Communicating finite state machines . . . 4
- 2.4 Contract as automaton . . . 4

3 Alternative modeling methodologies 4
- 3.1 Petri Nets . . . 4
- 3.2 BPMN . . . 5
- 3.3 DEMO . . . 5

4 Scenario 5
- 4.1 States . . . 5
- 4.2 Actions . . . 5
- 4.3 Actors . . . 5
- 4.4 Assets . . . 5

5 Data-objects 5
- 5.1 Immutability . . . 6
- 5.2 Forms . . . 6
- 5.3 Documents . . . 6
- 5.4 Custom types . . . 6

6 Identities 6
- 6.1 Inviting identities . . . 6
- 6.2 Updating an identity . . . 6

7 Process 6
- 7.1 Actions . . . 6
- 7.2 Manual actions . . . 7
- 7.3 System actions . . . 7
- 7.4 Sub-processes . . . 7
- 7.5 Projection . . . 7
- 7.6 Data operators . . . 7
- 7.7 Passive testing . . . 7

8 Adaptive workflows 7
- 8.1 Comments . . . 7
- 8.2 Deviation . . . 7
- 8.3 Scenario update . . . 7

9 Event chain 8
- 9.1 Cryptographic signatures . . . 8
- 9.2 Hash chain . . . 8

10 Distribution 8
- 10.1 Private chain . . . 8
- 10.2 Genesis . . . 8

11 Consensus mechanism 8
- 11.1 Chance of a conflict . . . 8
- 11.2 Branch validation . . . 9
- 11.3 Cascading rules . . . 9
- 11.4 Unanchored events . . . 9
- 11.5 Merging branches . . . 10
- 11.6 Forks . . . 10

12 Privacy 10
12.1 Linked data . . . 10
12.2 GDPR . . . 10
12.3 Zero-knowledge proofs . . . 10

13 Common patterns 10
13.1 Chain interaction . . . 10
13.2 Explicit synchronization . . . 10

Part II. Global blockchain 12

14 Centralized vs decentralized anchoring 12

15 Consensus algorithm 12
15.1 Leasing . . . 12
15.2 Raffle factor . . . 12
15.3 Forge probability . . . 13
15.4 Fair PoS . . . 13
15.5 Generator signature . . . 13
15.6 NG protocol . . . 13

16 Transaction types 13
16.1 Anchoring . . . 13
16.2 Authentication and authorization . . . 14
16.3 Certificates . . . 14
16.4 Chain of trust . . . 14
16.5 Smart accounts . . . 14

17 Summary blocks 14
17.1 Key block size . . . 14
17.2 Growth without aggregation . . . 15
17.3 Segregated witness . . . 15
17.4 Aggregation . . . 15
17.5 Difference to pruning . . . 15
17.6 Summary block size . . . 15
17.7 Total size . . . 15
17.8 History nodes . . . 16

18 Network vulnerability 16
18.1 Importance inflation . . . 16
18.2 Nothing at stake . . . 16
18.3 LPoS centralization . . . 16
18.4 Denial of service attack . . . 16
18.5 SHA-2 vulnerability . . . 17

Part III. Platform 18

19 Architecture 18
19.1 Micro architecture . . . 18
19.2 Application layers and services . . . 18

20 UI Layer 18

21 Application Layer 18
21.1 Web server . . . 18
21.2 Workflow engine . . . 18

22 Private chain layer 18
22.1 EventChain service . . . 18
22.2 Event enqueue service . . . 18
22.3 Event dispatch service . . . 18

23 Public chain layer 19
23.1 Anchor service . . . 19

24 Container orchestration 19

Part I. Live Contracts

Business process modeling is a common strategy for any medium or large organization[7]. Creating a visual representation of a workflow process allows it to be analyzed, improved, and automated (figure 1). Unlike procedures that are written in a natural language or written in a programming language, these models can be understood by both humans and computers.

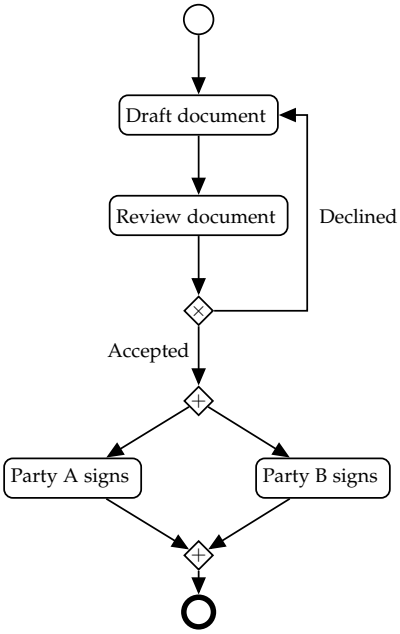


Fig. 1: *A BPMN diagram can be used to visualize workflows.*

For inter-organizational cooperation, modeling is not done merely to improve communication. The parties involved must specify the process to serve as binding agreement[8]; on the LTO platform this is called a **Live Contract**.

The LTO platform creates an ad hoc private blockchain for each Live Contract. Such a blockchain is not intended as an immutable ledger but ensures all parties have an up-to-date countersigned history of events and shared states.

1 Live vs Smart Contracts

Live Contracts have a similar goal as smart contracts as implemented on Ethereum[9]. Both define and solidify logic that can be applied in a trustless and verifiable way.

The philosophy behind these two types of digital contracts is very different however. Ethereum describes smart contracts, as cryptographic ”boxes” that contain value. These boxes only unlock it if certain conditions are met[10].

Live Contracts do not directly hold value but describe how two or more parties should interact. The intent is much closer to that of a traditional (paper) contract.

1.1 Ricardian Contracts

A Live Contract fits within the definition of a Ricardian contract[1][11]. Most notably, it's easily readable by both people and programs. This is an inherent property of the Live Contract that is obtained by the way it is defined. There is no separate natural language version for legal purposes and a coded version for programme execution.

1.2 Enforcement

On-chain enforcement is poorly suited for many real-world cases. Smart contracts rely on proactive enforcement, meaning either breaching the agreement must be impossible or dropping out must be possible for each side[12].

Take a non-disclosure agreement as an example. The blockchain can't prevent a party from disclosing information, nor can it force a party to actively participate in resolving a breach. For such a contract to work, a self-enforcing agreement[13] must hold the full penalty as a deposit, so each party is better off participating in a resolution.

Having to tie up large amounts of funds as deposits for penalty fees on arbitrary contracts is impractical for most organizations[14]. Additionally, the effectiveness of penalty interest and similar measures are limited to the value held by the smart contract.

Most inter-organizational business processes call for off-chain dispute resolution via an authoritative party. A Live Contract can facilitate in resolving a dispute. This may include conflict negotiation, mediation and even arbitration (by arbiter or judge).

Running a process on the LTO platform forms a verifiable history of events, reducing the amount of asymmetrical information. The distribution of information influences negotiations in case of a dispute[3] and influences the assessment by the authoritative third party.

1.3 User interface

Ethereum provides an internal Turing-complete scripting language, which a programmer can use to construct any smart contract or transaction type that can be mathematically defined[10]. This makes it very abstract as the state contained within the contract has no intrinsic meaning.

To interact with such contracts a user interface must be created for each specific smart contract, or more precisely, the interface of such a contract[15]. Standards like ERC-20[16] and ERC-721[16] manage to decouple the UI from the contract logic in some cases but restrict the possibilities when designing a contract.

With Live Contracts, information does have an intrinsic meaning. While this restricts use cases, it does enable generating an interface purely based on the data provided by the contract and its process. As a result, any workflow can be digitalized and executed on the LTO platform without the need of creating a specific UI for each one.

2 Finite state machine

A Live Contract defines a workflow as a **Finite State Machine** (FSM)[17]. This makes it possible to visualize the workflow as a flowchart (figure 2), which makes it understandable for both humans and computers.

1. A Ricardian Contract can be defined as a single document that is a) a contract offered by an issuer to holders, b) for a valuable right held by holders, and managed by the issuer, c) easily readable by people (like a contract on paper), d) readable by programs (parsable like a database), e) digitally signed, f) carries the keys and server information, and g) allied with a unique and secure identifier.

2.1 Deterministic Finite State Machine

Any blockchain logic needs to be deterministic[18]. Where computer programs may require extra effort to comply with this requirement, a DFSM is deterministic by definition.

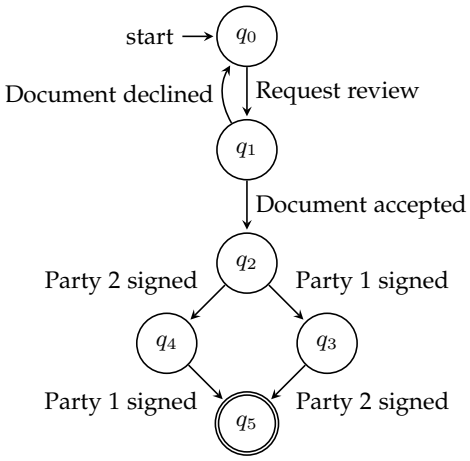


Fig. 2: *Example of a Finite State Machine visualized as a flowchart*

2.2 Extended Finite State Machine

Also visualized in figure 2 is how a problem arises when multiple actions are required to get to a state, but the order in which they occur is arbitrary. This *can* be modeled as a transition path for every possible order, as is done in figure 2. However, with this approach, the number of states and state transitions will grow exponentially with the number of actions. This makes the visualization of the workflow less clear, but also makes it harder and more error-prone to define the workflow.

This is why instead of using a regular FSM, a Live Contract uses an **Extended Finite State Machine**[19] (EFSM), allowing for conditional state transitions.

Figure 3 defines the same workflow as in figure 2 using an EFSM.

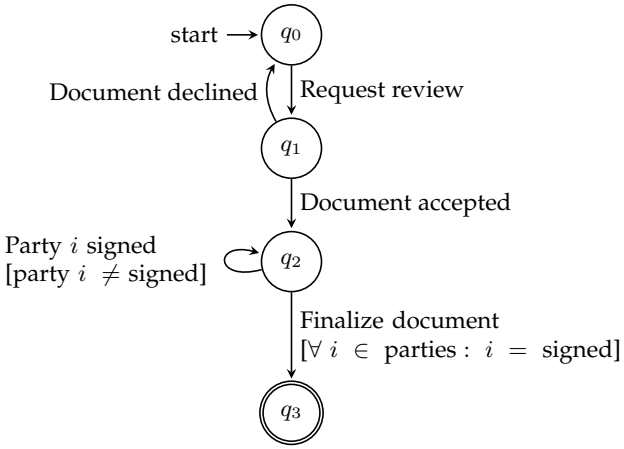


Fig. 3: *Example of an Extended Finite State Machine: conditions in brackets have to be true for the transition to be valid*

2.3 Communicating finite state machines

Finite state machines are limited to sequential behavior, they do not support concurrent processes. In order to represent workflows with concurrency, each sequence of parallel instructions may be represented as individual FSM.

The event chain (see section 9) is able to function as a communication channel between two FSMs. When two processes are isolated by using different event chains, this communication channel is non-deterministic, which inherently makes the whole system non-deterministic[20]. This can be overcome by making acknowledgments part of the FSM as shown in section 13.1.

2.4 Contract as automaton

A Finite State Machine can be applied as an agreement between the participants by formalizing obligations, permissions and prohibitions that parties impose on the other[21]. Contracts like financial agreements[22] and service contracts[23] can be fully digitized as FSM.

However, these representations are not sufficient to be used as workflow, as they do not define the orchestration, communication, and choreography within a process. These factors can be incorporated, however, this causes the FSM to grow exponentially in complexity[24].

For practicality, an FSM will at best represent an incomplete contract. This doesn't have to be a problem as these gaps may be filled by default rules[25]. The system does allow renegotiation of a Live Contract, either to resolve a particular situation or in general, as shown in section 8.

Another thing to note is that not every action in a process constitutes a binding factor. In figure 1, the acceptance of the text of a document does not constitute a binding agreement, as this only occurs when the document is signed. To facilitate this distinction, actions can be categorized as either informative or performative actions[26].

3 Alternative modeling methodologies

Communicating extended finite state machines are commonly used in describing telecommunication systems and other real-time systems[27], but not business processes.

More common notations for workflows provide additional challenges when modeling decentralized inter-organizational processes.

3.1 Petri Nets

Petri nets[28] are a graphical way of representing a system in which there are multiple independent activities running at the same time. The ability to model multiple activities differentiates Petri nets from finite state machines. In a finite state machine, there is always a single "current" state that determines which action can occur next. In Petri nets, there may be several states any one of which may evolve by changing the state of the Petri net. Alternatively, some, or even all, of these states may evolve in parallel causing several independent changes to the Petri net to occur at once[29].

Workflow nets (WF-nets) are a subset of Petri nets that may be used to describe business processes[30]. WF-nets can describe the whole process rather than only one sequence.

Extended finite state machines use a global state to hold information. This information must be isolated per sequence. Failure of making the data immutable can be exploited as shown in section5.1. With Petri nets, using global information isn't possible. Instead, information needs to flow through the workflow.

With the approach of CEFSMs, sequences are defined as individual processes. This makes it trivial to apply access control. When representing the workflow as a whole, this can't be addressed in the same way.

Petri nets have an interesting notation, which has proven to be able to correctly represent a business workflow. Given it's possible to address the mentioned challenges, the WF-nets notation might be preferable over using individual FSMs.

Due to the similarity between Petri nets and FSMs, switching to WF-nets does not fundamentally change the solution as described in this paper.

3.2 BPMN

BPMN is the industry standard in business model processing and would be a likely candidate for a modeling notation for LTO. However, there are a number of limitation that are particularly problematic for inter-organizational systems[31].

The Business Process Execution Language (BPEL) typically associated with BPMN, is a non-deterministic Turing complete language towards web services[32]. This makes it ill-suited for automation on a blockchain.

A proposed alternative is to translate a BPMN model into a Petri net, which is translated in a smart contract[33].

While the translation into a (Turing complete) smart contract is unnecessary, the translation from BPMN into a Petri net (or CEFSM) might be interesting in order to support the current industry standard.

3.3 DEMO

"DEMO" is an acronym for "Design & Engineering Methodology for Organizations". This methodology for describing organizations and their business processes is based around the "communicative action". It uses four models to create a holistic view; the Construction Model (CM), Process Model (PM), Action Model (AM), and Fact Model (FM)[34].

Rather than considering each step of a process individually, it establishes a generalized workflow for every single performative transaction. In such a transaction, there are two roles: the initiator and the executor. The standard sequence that follows is as follows. The initiator does a request, the executor makes a promise, the executor performs the actions and makes a statement about the result, the initiator either accepts this as transaction completed or rejects it[26].

Other alternative flows are also modeled, like the executor declining the request, the initiator wanting to revoke the request, the executor being unable to fulfill the promise, etc. These alternative flows are often not, or only partially, modeled when using other modeling methodologies, while in practice these are always present.

In the Process Model (PM) it combines these transactions to model a complete business process. The difference between this model and a workflow is that in this model everybody works in parallel as much as possible, specifying dependencies between transactions where needed. The models don't give a clear overview of where we are in a process. Also, mutual exclusion of information (don't edit a document that's ready to be signed) is not immersive but needs to be specified.

DEMO might be a good way to make a high-level model, which yields a workflow that can then be fine-tuned. This reduces the reliance on deviations (see section 8.2).

4 Scenario

A workflow is defined as a data-object; *the scenario*. It consists of:

- q_x as a state with q_0 as the initial state,
- Q as the set of all possible states $Q = \{q_0, ..., q_{n-1}\}$,
- σ_x as an action,
- Σ as the set of all possible actions $\Sigma = \{\sigma_1, ..., \sigma_n\}$,
- δ as the transition function $\delta : Q \times \Sigma \to Q$,
- F as the set of final states with $F \subset Q \mid F = \emptyset$,
- $\bar{I}$ as all actor definitions,
- $\bar{A}$ as all asset definitions,
- D as the set of embedded data-objects.

4.1 States

States in set Q typically consists of:

- a title: a short title for the state,
- a set of actions with transaction as $\{(\overrightarrow{q_x}, \delta), ...\}$,
- a description: a long description for the state,
- a map or instructions for specific actors.

The state describes the actions that may be performed in this state and includes the state transition. This allows actions to be used in different states.

4.2 Actions

The scenario defines Σ, the set of all possible actions that can be performed in the workflow. Examples of such actions are filling out a form, reviewing a document, and sending an HTTP request or response. The action type and object properties are defined using JSON schema.

An action defines which of the actors from set I may execute it, as well as, optionally, additional constraints as described for the EFSM.

When an action is executed, a state transition is triggered. Actions can be categorized in manual actions that require human interference to be executed and system actions that can be executed automatically by the system.

Optionally actions can define instructions on updating actors and assets using data from the response.

4.3 Actors

Set $\bar{I}$ defines all actors that can have a role in a process. Each actor is defined as an object using JSON Schema. Actor properties that are relevant to the process must be defined.

An actor in a scenario is only a static definition that may be instantiated in a process.

4.4 Assets

$\bar{A}$ defines all assets that are available in a process. An asset is a variable data-object. Properties that are relevant to the process must be defined.

Note that the scenario only defines the structure of assets. Assets can only be instantiated within the process.

5 Data-objects

Besides scenarios, other types of data-objects may be defined. All data-objects, including scenarios, use JSON schema as a type definition. Common examples are forms, documents, and templates.

Data-objects can be embedded in a process or linked and stored independently.

Linked objects are identified by the SHA256 hash of its JSON representation. To ensure JSON encoding the data always yields the same result, a deterministic method of JSON encoding is applied.

5.1 Immutability

Data-objects are immutable to the extent that applying a modification to a data-object, yields a new data-object. If this data object is embedded in the process as an asset, the old object is replaced with the modified one.

Importantly, if the same data-object is available in multiple processes, changing the object in one process will never propagate to other processes.

Failing to do so could lead to exploitable situations. Figure 1 shows the process of negotiating and signing a document. It's clear that the document must not be allowed to be modified during the signing sequence.

5.2 Forms

A form definition uses a JSON Schema to define the data structure that should result from filling out the form. Optionally a UI Schema can be used to specify how a corresponding field may be rendered and displayed.

There are several similar implementations[35–38] for this. The aim is to work together with these projects to form a unified standard.

5.3 Documents

Digital workflows can largely eliminate the need for paper documents. However legal compliance, backward compatibility, and simply convention may still require the use of documents. By defining templates as part of the Live Contract, natural language documents can be generated using data gathered by the process.

We recommend using the Open Document Format[39], which supports fields and conditional sections for creating templates.

5.4 Custom types

Any JSON schema that defines an object can be used as a data-object type. Usage of custom types causes the risk of a workflow not working properly as other parties may participate via a node that doesn't support that type. Data with unknown types will be stored "as is" and is unavailable outside of the context of the process.

6 Identities

An identity defines a person, team or organization within a Live Contract. An identity always has the following information:

- identifier,
- node URI,
- custom information,
- sign keys,
- an encrypt key.

Identities are not the same as actors. An actor is an abstract role like 'student', an identity might be 'Bruce Willis' or 'Acme corp'.

Sign-keys is a map with one or more public keys associated with the identity. The 'user' key belongs to the identity and can only be used by him/her to sign an action. The 'system' key is owned by the node that the identity uses and is used to sign automated actions. Other key types may have a meaning defined within a process.

The public encryption key can be used to encrypt data, that can only be decrypted by this identity.

6.1 Inviting identities

To add parties to a process, the scenario needs to define actions to add other identities. If the public keys are known within the process, the identity can be added directly.

When the public key is not known, the identity needs to be invited (figure 4). This can be done through any means deemed secure enough, including e-mail. The inviting system generates a one-time key and sends it to the invited identity. The invited party must replace this with its own secure 'user' and 'system' keys.

Before a new identity can fully participate in a process, additional authentication may be required. This can range from SMS verification to federated identity verification and even notary approval.

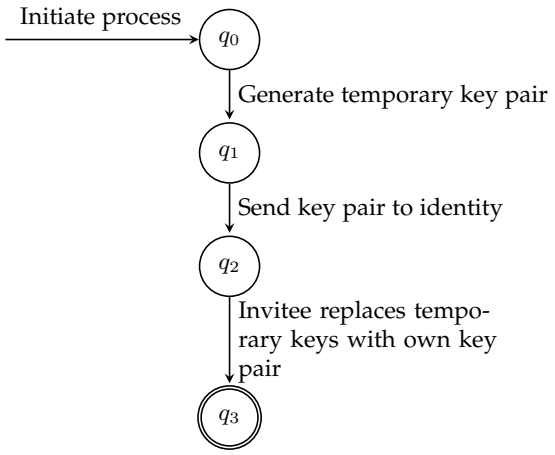


Fig. 4: *Inviting an identity to join the process.*

6.2 Updating an identity

An identity is free to modify its own information, except for the identifier. This also allows a party to switch to another node. In cases where a user should not be allowed to switch nodes, it's up to the node of the identity to reject that change. Removing an identity can be done by clearing the sign keys, encrypt key and node URI.

Updating other identities is only possible if such an action is defined in the scenario and allowed in the current state.

7 Process

Where a scenario is the stateless definition of a workflow, the process is a stateful instantiation, consisting of:

- θ_x as response, where $f : q_x \mapsto \theta_x$
- Θ as an ordered list of all responses $\Theta = \{\theta_0, ...\theta_n\}$
- q_t as the current state
- I as set of all available actors
- A as set of created assets

7.1 Actions

Executing an action always yields a response. The response must be signed by the actor and submitted as a new event. Nodes independently determine the new state based on the current state and the executed action.

The scenario is defined as a deterministic FSM. However this only concerns the state transitions and the projection. On systems like Ethereum and Hyperledger, all logic must be deterministic, as it's executed by all nodes and needs to yield identical results on all systems[40].

With LTO only a single node or a single actor executes an action, meaning actions do not need to be deterministic. As such, concepts like oracles are not needed.

7.2 Manual actions

Applications built on the LTO platform must inform human actors which actions they may perform in the current state. A human actor will sign his own response event before submitting it to his node, which will distribute it to all parties.

7.3 System actions

System actions do not require human interference but are executed by the node automatically. As such, the node signs the response, rather than the human user.

These actions are always performed by a single system and do not have to be deterministic. Other parties validate the response and can reject it if needed. Because there is no human interference involved in system actions, the actions are signed by the system itself instead of the actor.

It's also possible to schedule a system action to be executed at a later time. This is specified in the scenario. It allows for a timeout on a state or polling an external source at a predetermined frequency.

System actions are automatically executed and may yield an error when used incorrectly or when they fail otherwise. For these actions, not one but two state transitions must be defined. One for a successful execution and one in case of an error.

7.4 Sub-processes

A process based on an FSM can only be in one state at a time. Sub-processes allow a Live Contract to hold multiple states and make it possible for different procedures to be done in parallel. While these processes share an event chain, the data for each process is still isolated.

To facilitate sub-processes, a Live Contract may contain sub-scenarios that can be instantiated from the main scenario.

7.5 Projection

Besides the FSM state, the process also contains other stateful data in assets and actors. The payload of every response can be used to update this data. The rules on how the payload updates the data are defined in the scenario. Updating the projection is deterministic and therefore applying a given set of responses against the scenario will always yield the same projection.

The projection can be used to set the parameters of an action. It's also the data for the conditions that make it an *Extended* Finite State Machine.

7.6 Data operators

Data operators may be used in the scenario for specifying how the projection affects the process. These operators are deterministic functions without side effects. They can be used for arithmetic or logical operations. The result of these operations may be stored in the projection and can be used, for example, to base state transitions on.

7.7 Passive testing

A scenario that contains a loop consisting exclusively of system actions could result in an infinite loop causing a massive amount of transactions. When validating, we want to reject scenarios that have such a construct.

Determining if a program can run forever is known as the halting problem[41]. The problem has been proven unsolvable for Turing-complete machines. However, it can be solved for FSMs[42]. Since an FSM has a finite amount of transition paths, they can all be checked for loops.

Passive tests for EFSM models are complicated by the presence of infeasible paths. This problem has been well-documented but remains unsolved[19, 43]. For simplicity reasons, we can assume any path to be feasible by ignoring the conditions. We accept that this may cause false positives.

8 Adaptive workflows

A scenario will model the most typical cases of a process. It's impossible to foresee all situations in advance and tedious to model every possible edge case. Taking a code-is-law approach would make the system rigid. Instead, Live Contracts supports three methods of resolving such issues.

8.1 Comments

Comments are used to communicate with other identities. They can, for example, be used to resolve conflicts or conduct discussions outside of the process. Using comments instead of off-chain communication methods makes sure that the conversations are logged on the blockchain. It also allows backtracking to check when in the procedure certain conversations took place.

Comments are not restricted to text messages. It is also possible to use images or documents to assist in the communication. Comments are not part of the process, meaning that adding a comment does not trigger a state transition. This way it is always possible to conduct a discussion about subjects that were not predefined in the procedure.

8.2 Deviation

Any party may propose a deviation from the main flow by defining a partial scenario. This sub-flow must start from one of the states of an existing scenario and end in a state of that scenario. Deviation flows are only executed once as they are no longer available when the process returns to an existing state.

All parties need to agree upon the deviation. Note that deviations might lead to forks that can only be resolved through manual conflict resolution.

A deviation can be used to resolve disputes. Any party may propose it to dispute the correctness of a previous event and present a solution on how to correct that.

Another typical case of using deviations is a payment arrangement. Organizations obviously don't want to make that option known at forehand. Predefined sub-flows allow such arrangements while keeping them under wraps.

8.3 Scenario update

It may be required to change the scenario for a running process, for instance, when an agreement is updated or a new law is passed.

A party may provide a new scenario for a given process through a deviation flow. This flow moves the state out of the outdated scenario and ends in a transition into the new scenario.

9 Event chain

In order to determine the state of the FSM and the projection, we need to process the set of responses in the given order. Inserting or removing an event, changing the order of the events or modifying the payload might result in a radically different state.

In a centralized solution, the controlling party is responsible for data integrity. All parties rely on trust on this party to do so, as it represents a single source of truth. In a decentralized system, this power and responsibility is shared among all parties.

To facilitate this, the event chain works like an ad hoc private blockchain. Each response is wrapped in an event, which can be viewed as a block with a single action. The sequence of these events form a hash chain that is shared between the parties. The consensus algorithm ensures the parties agree upon the sequence of events.

9.1 Cryptographic signatures

To ensure nobody can falsify or forge events of others, each event is signed before submitted using asymmetric cryptography. The signed event also serves as a receipt, allowing other parties to prove that the action has been executed by the signing identity.

The platform uses ED25519[44] signatures. These elliptic-curve signatures are widely used, well supported and considered secure by institutions like NIST[45] and ENISA[46]. Elliptic curve cryptography allows for faster single-signature verification and signing without losing security. It also reduces the needed size of both the keys and the signatures. Note that this method on itself doesn't grant complete security, as any party is still able to falsify or forge their own events. In other words, cryptographic signatures can't prove an event did not occur.

9.2 Hash chain

Each event can be uniquely identified using its SHA-2 256bit hash. This industry standard algorithm is fast and resistant to pre-image and second-preimage attacks as well as collisions[47]. It's the recommended cryptographic hashing algorithm by NIST[48].

Embedding the hash of the previous event in the hash of the next event creates a hash chain, which records the chronology of the events. When used in combination with cryptographic signatures, a hash chain provides an adequate measure of proving that a specific sequence of events resulted in the current state[49].

10 Distribution

Rather than requiring parties to pull information from a central server or from each other, each party is responsible for pushing events to the system of all other parties.

Systems need to be always available in order for events not to get lost. Decoupling and the use of a message queue reduce issues with temporary unavailability. In a typical case, all parties will connect to a node they trust which receives and processes events for them. This node is part of the larger system (see section 19.1).

With the focus on organizations and governments, it's up to these organizations to run a node. Users connect to the node of their organization, or a publicly available node of their choosing, to participate in a process.

10.1 Private chain

The event chain is a private chain that is only shared between the nodes chosen by the identities. Nodes are not aware of private chains that they are not part of.

A node stores and facilitates many event chains at the same time. Unlike side chains, event chains are completely isolated. Chains do not affect each other directly. This allows for horizontal scaling, given that the activity per event chain is reasonably low.

10.2 Genesis

Anybody may create a new event chain at will. The genesis block of this chain contains the identity of the user that's creating the process, the subsequent block contains the scenario. As part of the scenario, other identities will be invited to this private blockchain.

11 Consensus mechanism

LTO is a distributed system, where all parties are able to participate via their own node. Nodes distribute all events to their peers, who process them. This means there is a brief moment where the state of the process between the nodes differs. Eventual consistency[50] guarantees that, given that there is no new event submitted, eventually the state of the process on all nodes will be the same.

However, sometimes new events are submitted before consistency has been achieved. At this moment, it is possible that two or more nodes append an event upon the event chain. During a Byzantine failure[51], all nodes believe their information is valid. However, the overall system is in an inconsistent state. In this state, nodes would no longer accept new events from one another and need to be able to come to a consensus, rather than halt.

Distributed applications use a different kind of consensus algorithm for this. In general, this is a case of *Byzantine fault tolerance* (BFT). Early Byzantine fault tolerance methods do not scale well [52]. The invention of better scaling consensus algorithms like *proof-of-work* [53], *proof-of-stake* [54] and *proof-of-authorization* [55] made it possible to create distributed networks with a large number of participants, also called distributed ledger technology.

While these consensus methods scale much better than traditional BFT methods, they have a need for a relatively high amount of participants in order to be secure. The event chain is a private blockchain with relatively few participants, meaning those algorithms won't work. Rather than trusting on a majority vote, nodes consider their state correct unless proven otherwise.

11.1 Chance of a conflict

Event chains rely on optimistic concurrency control. Many conflicts would put a strain on the consensus algorithm which can be relatively slow as it may have to wait on a block to be generated.

We define a distributed event chain as follows:

- Let N be the set of entities $\{n_1, n_2, n_3, \dots\}$ contributing to the event chain.
- Let C_n be the event chain, a sequence consisting of events $(e_1, e_2, e_3, \dots)$, belonging to entity n and let C be the set of all copies of the event chain $\{C_n | n \in N\}$.
- Let's define a conflict or branch as $\exists\ i, j \in N : i \neq j,\ C_{i_0} = C_{j_0}, C_i \not\subset C_j,\ C_j \not\subset C_i$.

For a conflict to occur by accident, two parties must add a block to their chain before they received the others chain update.

Let's call the chance of somebody propagating an update to the chain $P(x)$. This chance depends on the amount of blocks being added to the chain during a given time frame, the time it takes to propagate this block to the rest of the network and the amount of entities contributing to the chain in this time frame. Assuming everybody contributes equally to the network this can be derived to formula (1)

$$P(x) = \frac{f \cdot t}{n} \tag{1}$$

with:
f = Total amount of transactions / time frame
n = Total amount of active participants
t = Time it takes to propagate a block to the rest of the network

This chance can be used to calculate the probability that a conflict will occur. This probability is derived by subtracting the chance of not having a conflict from 1. When there is no conflict it means either nobody has contributed to the chain that moment, the chance of which is calculated in formula (2),

$$(1 - P(x))^n \tag{2}$$

or only one node contributed to the chain, the chance of which is calculated in formula (3).

$$P(x) \cdot (1 - P(x))^{n-1} \cdot n. \tag{3}$$

Therefore the chance of a conflict is calculated by (4).

$$P(c) = 1 - (1 - P(x))^n - P(x) \cdot (1 - P(x))^{n-1} \cdot n. \tag{4}$$

With a network delay of 1200ms we see about chance on a conflict of $< 2\%$:

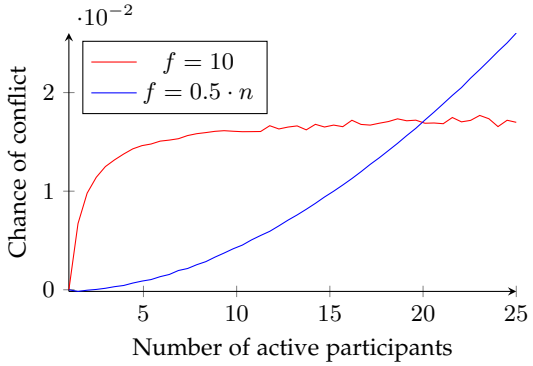


Fig. 5: This plot shows how the chance of a conflict occurring increases when the number of participants increases. However, it stabilizes if the total number of transactions stays the same.

Figure 5 shows that the chance of a conflict occurring is more or less constant when the number of participants increases but the total number of transactions stay the same. However, when the number of participants increases usually the number of transactions increase as well. For example, more participants may lead to more comments. When this is the case, the chance of a conflict increases exponentially with the number of participants.

LTO works with very few active participants on a single event chain. This reduces the chance of a conflict. With 5 or more participants, the number of active participants is no longer relevant. With more than 10 participants the chance of a conflict becomes more or less linear based on the network delay and transaction frequency.

By using individual shared nothing events chains with decoupled message queues, the transaction frequency will be very low. This marginalizes the chance on a conflict.

11.2 Branch validation

A node can only prove the other party was aware of the chain up to the point of its last event. Any party is allowed to branch the chain after their last commit. If a party tries to branch the chain from a point before his last event, that branch is automatically discarded by all (other) parties and the event is logged.

Before accepting the new events from the conflicting branch, they are validated similar to any received set of events; The event must be signed correctly by one of the identities and the event must be anchored where the timestamp of the event matches that of the anchor within given boundaries.

11.3 Cascading rules

In case of a conflict, the following rules are applied in this order

- the priority of the event,
- the priority of the actor,
- the order of anchoring.

An action in the scenario may be given priority, this translates to the event priority. Additionally, some other event types like comments have a lower priority as the exact sequence is not of any importance. In case of a conflict occurs, the block that contains the action with the highest priority will be accepted.

In case the priority is the same, rules are applied based on identity, creating different levels of authority within the process. Every actor in the scenario may be given a priority level. By default, this priority is the same for all actors. To resolve the conflict, the events added by the actor with the highest priority must be accepted.

If both the events priorities and the identities authorities are the same, the third method of consensus is applied. Nodes must anchor events on the global blockchain. The order of transaction on this global blockchain is fixed in mined blocks. Therefore the order in which the events have been anchored on the global blockchain can be used to achieve consensus. When this is the case, the block that was anchored first must be accepted. As such, a consensus of the private event chain is reached via consensus on the public blockchain. On the public chain, consensus is reached by anonymous collaboration between a large number of participants.

Using priorities allows a front-running attack, where an actor may respond upon an event by creating a new branch that subsequently invalidates that action. Priorities should only be used where this is not a problem.

11.4 Unanchored events

When a block is received that has not been anchored yet, one may decide to accept the block anyway. This is, of course, if no conflict arises by accepting it. If the anchoring of the block has merely been delayed, accepting it would prevent unnecessary delays in the process itself. On the other hand, if the block is never anchored, no real problems arise. If everybody accepts the block, the process may continue as normal and the block may be anchored later.

11.5 Merging branches

In case of a fork, most blockchain applications will pick one chain and ignore everything that happened on the other branches. With a blockchain like Bitcoin, all transactions will be included in the mined blocks of each branch eventually.

On the event chain, the events themselves build up the hash chain. Picking one of the forks would lose information about an executed action. Instead, when a node becomes aware of another branch that has precedence over its own branch, it must base the events it has locally on top of the other chain. This is similar to a rebase action when using git[56].

11.6 Forks

Even though there is a guaranteed way to achieve consensus, a participating party might decide to ignore the other chains and keep the fork in place. For most blockchain applications, like Ethereum[57], there is no reason to interfere with this, as value comes from participating on the main chain only.

Live Contracts are a tool to digitize and partly automate existing processes. Even though the blockchain allows for forks, those processes usually do not. In the case of a fork, parties can start a secondary procedure to try to resolve the conflict manually.

12 Privacy

LTO is built for running processes between parties. Beyond these parties, no-one needs to be aware of the process or even the collaboration.

Public blockchains allow anonymous accounts, however these function as pseudonyms. Any transaction may reveal the identity of an account, exposing the full transaction history. Smart contracts require the data to be public as it needs to be available for every node. On consortium blockchains, participants are aware of each other.

Ad hoc private blockchains uniquely allow a random assortment of participants to collaborate without the need for approval and without making information public. These blockchains can be completely erased when the process has been completed.

12.1 Linked data

Each party connects via its own node or a node it trusts. Each node has a private storage service, where users can store data. Users have complete control over data stored here, similar to data stored on a service like DropBox. They can remove their personal data at any time. The data is not shared or processed without a valid data processing agreement and explicit approval of the user.

When an action results in linked data, that data is not directly shared with other parties, only a hash is added to the blockchain. LTO prevents the hash from being used for anything else than verifying received data, by putting it in an envelope together with a timestamp and some random data. The hash created to form the envelope will never occur on the blockchain more than once.

When an organization indicates it wants to perform an action that requires linked data, the node of that organization will automatically do a request. The data owners node checks if the specified action is valid and thus may indeed be performed by this actor in the current state.

12.2 GDPR

With the arrival of the new GDPR[58] (General Data Protection Regulation) in Europe, an argument has risen about the fact that a lot of blockchain applications are not GDPR compliant [59]. The two main reasons for this are:

- the fact that the blockchains immutable nature is in conflict with the right to both amend and erase your data,
- the fact that there is no dedicated data controller since it is a distributed environment.

Linked data means that the node chosen by the party to participate will act as the data controller for that user. All other parties are always data processors. Data requests are automatically formatted to be proper data agreements, which include the purpose and time required for processing the data.

Ad hoc blockchains allow the complete chain to be erased if required.

In short, LTO privacy features make the solution GDPR compliant without much additional effort.

12.3 Zero-knowledge proofs

A scenario may require one party to prove to another party that it knows a specific value. A zero-knowledge proof is a method of doing so without conveying any information apart from the fact that it knows that value.

The platform supports zero-knowledge proofs through an interactive proof system. Two parties; the prover and the verifier exchange messages with the goal for the prover to convince the verifier of its honesty (completeness) and for the verifier to expose a dishonest prover (soundness)[60].

A zk-proof for a Live Contract is always an action between two parties. There is no need for experimental non-interactive zero-knowledge proofs like zkSNARKS.

13 Common patterns

13.1 Chain interaction

Some processes may have to interact with other processes to be able to continue. Examples are when a process needs permission from another process to continue or to retrieve the result of a conflict resolution process. Requesting data from another process is done by following the pattern shown in figure 6.

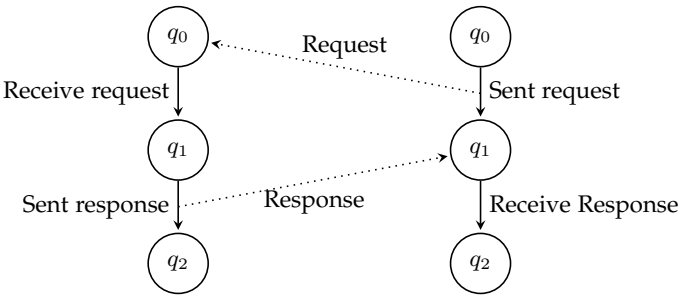


Fig. 6: *Pattern followed when two processes interact.*

13.2 Explicit synchronization

In some cases, it can be especially troublesome if an event is propagated too late. If this is the case, explicit synchronization can be built into the scenario. This requires all parties to acknowledge the current state before continuing the process.

This is a pattern within the FSM intended to prevent parties from forking the chain of events prior to this event. In case of a dispute or conflict, such an acknowledgment can be used to decide which branch should be used to continue the process.

This kind of explicit synchronization only works if all parties acknowledge the current state. If some parties lack the incentive to continue or even have the incentive to fork the process, another solution is required. In this case, the party that wants to continue the process announces this to all other parties. If any party that receives this announcement notices that the announced action is not valid on their chain then they can propagate this. This means that a fork has happened and regular conflict resolution has to be applied. To remove the impact of network delays during this announcement, a set amount of time is waited before assuming that nobody rejected the announced event.

Part II. Global blockchain

The LTO Global blockchain is a permissionless public blockchain, purposely build for verifying information. It exists to support Live Contracts and the private event chains. The global blockchain features anchoring and digital identities that are interoperable across event chains, blockchains, and applications.

Notary transactions can be done on nearly any blockchain. However, on blockchains optimized for financial transactions or general logic, notary type of transactions are expensive and inefficient[61]. Furthermore, optimizations as pruning and sharding can have a negative effect as relevant information is omitted[62, 63].

The global blockchain is of the Nxt family[64]. A unique characteristic of this blockchain is that transactions are based on a series of core transaction types that do not require any script processing or transaction input/output processing on the part of network nodes. This reduces the blockchain size, increases efficiency and allows ways of aggregation particularly beneficial for notary transactions.

Rather than starting from Nxt directly, we use a fork of the WAVES platform[65] as a basis. This platform has implemented a number of improvements, like the NG protocol (section 15.6), that our network will benefit from. The existing transaction types focusing on digital assets, including colored coins, are removed or disabled and replaced by notary transaction types.

14 Centralized vs decentralized anchoring

Other solutions for blockchain anchoring use a centralized approach, where all hashes are collected for a period of time. From these transactions, a Merkle tree is created which is put in a single transaction on a third party blockchain like Bitcoin. Because of this, there is no direct feedback from the system and it can take several hours before the final receipt can be collected[66] from the central service.

The LTO global blockchain is a decentralized solution, where every node sees all transactions. When an anchoring transaction is broadcasted it's instantly visible, and after approximately 3 seconds it's pre-approved using NG (section 15.6). The transaction is in a block within a minute.

Nodes can keep track of all anchored hashes or only of their own. If needed they can create a receipt independently (section 16.1). There is no need for a centralized service.

15 Consensus algorithm

The global blockchain functions as a typical public blockchain. A node is selected as a generator to validate the transactions and forge a block. To determine a generator we use the **Leased Proof of Importance** (LPoI) consensus algorithm. The generator is to be rewarded with the fees of the transactions in the forged block.

Proof of Importance is a variation on Proof of Stake (PoS), where the chance of being selected to forge a block is determined based on the number of tokens you hold and stake. With Proof of Importance, the chance increases based on usage of the network by the node[67, 68].

The token economy that emerges from this consensus algorithm is described in detail in the "LTO Token Economy" paper[69].

15.1 Leasing

NXT, Waves and other blockchains in this family use Leased Proof of Stake[64, 70]. By leasing tokens, the token holder passes the right to forge a block to the selected node. These nodes can work similar to a mining pool, sharing the rewards proportionally among the lessors.

NXT style networks are susceptible to attack when an attacker controls at least 1/3 of all active balances[71]. This is an issue, as projects that have implemented the LPoS algorithm tend to lead to a high level of centralization. The top 2 Nxt nodes control over 50% of the network[72]. With Waves, the top 2 nodes control over 1/3 the network and the top 5 over 50%[73].

In section 18 we discuss the importance of a high amount of decentralization for this network. To prevent nodes with massive leased stakes, there is a limitation on leveraging on leased tokens. Any node needs to own at least 10% of the tokens it stakes. Proof of Importance also counters this effect as it disadvantages nodes that are passive stakers.

15.2 Raffle factor

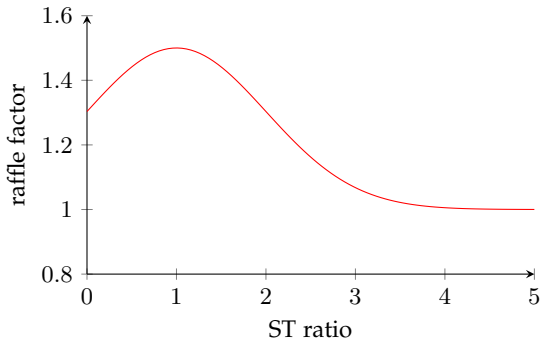
To calculate the usage of the network by the node, which influences the chance to forge a block, the balance of staking versus transactions, the S/T-ratio will be used.

$$\text{ST ratio} = \frac{\text{Staked tokens as \% of total}}{\text{Contributed transactions as \% of total}}$$

The S/T-ratio is related to a 'raffle factor'. The raffle factor is a mathematical formula that influences the chance that a node will be chosen to generate a new block. The more balanced the ST-ratio (closer to 1.0), the higher the Raffle factor. If the ST-ratio is unbalanced (a node does not contribute any transactions), the raffle factor will be 1.0.

$$\text{raffle factor } r = 1 + (0.5 \cdot e^{-0.5 \cdot (\text{ST ratio}-1)^2})$$

This formula results in a large standard deviation of the bell curve.

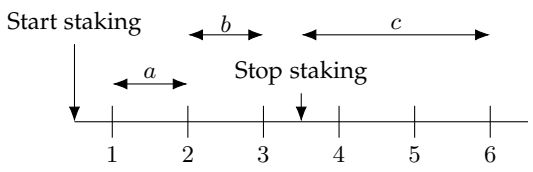


The maximum raffle factor is 1.5, which is halfway between the minimum and the absolute maximum of 2.0. The absolute maximum is determined by the possibility for importance inflation through spam transactions. This is shown in detail in section 18.1.

To fully understand the concept of the raffle factor and its effect on the token economy, please read the "LTO Token Economy" paper[69].

15.3 Forge probability

The chance of being selected to forge is $P(forge) = S \cdot r$. The contributed transactions T are calculated over time. When calculating $P(forge)$, S must be constant over the same period of time to prevent the possibility of abuse.



$a =$ Moment when P(forge) is calculated

$b =$ Moment when you forge a block

$c =$ Moment when tokens are still locked

Fig. 7: Timeline for staking tokens and forging blocks. Time is measured in number of summary blocks.

15.4 Fair PoS

The formula that decides which node is eligible to forge a new block, is based on the Fair Proof of Stake algorithm[74] created by Waves. This is an improvement on the original Nxt PoS algorithm, which overvalues higher stakes.

For a further understanding about the underlying algorithm, please read the "Fair Proof of Stake" paper[74].

To convert this algorithm from PoS to PoI, the formula applies the raffle factor to the staked balance to result in the effective balance as $b_i \cdot r$.

- T_i as block generation time for i-th account,
- X_n is the generation signature,
- r raffle factor,
- b_i percentage of staked vs total staked,
- Λ_n is the base target
- $T_{min} = 5$ is a constant for delay between blocks,
- $C_1 = 70$, a constant defining shape of delay distribution,
- $C_2 = 5E17$ is a constant to adjust base target.

$$T_i = T_{min} + C_1 \cdot log(1 - C_2 \cdot \frac{log(X_n / X_{max})}{r \cdot b_i \cdot \Lambda_n})$$

If you receive a new block before the time delay is reached, this block must be added to the chain and a new time delay must be calculated. The previous T_i is no longer relevant. Each node calculates T_i itself, this information is not supplied by a generator. This means that there is no point in using an incorrect stake in the calculations.

15.5 Generator signature

The block hash is never considered in determining the time delay T_i. That hash is based on the contents of the block, which is determined by the node forging the block. If this hash played any part in determining who could forge the next, it would be trivial to manipulate T_i. A node could create several different blocks and only broadcast the one that has the lowest time for itself.

To prevent this, only the generation signature is used. This signature is a secondary hash chain, using only the previous generation signature, plus the public key of the generator. With Nxt this is completely deterministic, making it susceptible to be taken advantage of[71].

To combat this, as well as reduce the chance of forks, Fair PoS uses the generation signature of 100 blocks ago. Any change in balance on the node or in leases changes T_i for a given X_n. Nonetheless, any group controlling at least 1/3 of the tokens can still get a 30% advantage.

PoI requires the staked balance and raffle factor to be constant over a fixed period of blocks. This would make it even more vulnerable for such an attack.

As a solution, the generation is not a hash that can be publicly calculated. Instead, a node must hash the previous generations signature and sign that hash with its private key. This serves as a generation signature. Contrary to Nxt and Waves, a node can only calculate T_i for itself, making it impossible to determine the generators in advance.

15.6 NG protocol

The NG protocol was proposed to reduce the scalability issues on Bitcoin. While it was never implemented on Bitcoin, Waves NG is active on their main net since December 2017.

With NG, two type of blocks are generated; the micro block and the key block. The node that was previously elected may continue validating transactions, creating micro blocks on average every 3 seconds. When a new node is elected it creates a key block from the outstanding micro blocks. Transactions in a micro block can be considered secure to some extent, and are suitable for low-risk transactions like anchoring.

The reward of the transaction fees is split between the node that forged the micro block and the node that forged the key block in a 40% - 60% split. This split must always be in favor of the key block forger. Otherwise, there would be an incentive to disregard the already forged micro blocks and create new micro blocks yourself.

In a public stress test Waves' NG has been proven to be able to process up to 6000 Tx/min, with a peak of 17,000 Tx/min[75]. Potentially the NG protocol could handle up to 1000 Tx/sec or 60,000 Tx/min.

NG reduces practical latency and is a key component for other optimizations.

16 Transaction types

The LTO global blockchain uses predefined transaction types. This allows for more compact blocks and removes the need for scripting. The list of transaction types may be extended in the future if needed. The types of transactions that are currently possible are:

- Anchor: used to verify transaction from the private blockchains,
- Issue a certificate: used to declare relationships between identities,
- Extend/revoke a certificate: used to extend or remove such a relationship,
- Transfer token: used to send tokens to another identity,
- Stake tokens: used to let a participant stake or lease tokens,
- Cancel staking: used to stop staking or leasing tokens,
- Set script transaction: used to configure smart accounts.

16.1 Anchoring

Anchoring is the method of taking a hash of a document or other data and storing it within a transaction on the blockchain. The goal is to make it impossible for anyone, including the creator, to back-date or forward-date his document[76].

Every event of the private event chain is anchored onto the global blockchain. Third party applications may use the global chain to anchor documents for proof of existence. We estimate that 99% of all transactions on the global chain will be anchoring transactions. Considering that most transactions are for anchoring, aggregating these reduces the disk space required by the blockchain.

When forging a block, a node creates a Merkle tree[77] from the transactions in the order presented in the list. Only the Merkle root is added to the blockchain. As part of the validation, each node recreates this Merkle tree.

Nodes are able to index every anchor hash. However, to reduce disk size, most nodes should opt to extract the Merkle path of its own anchor transactions. This path forms the receipt that can be stored with the original data (like the event).

16.2 Authentication and authorization

Challenge/response authentication methods, like username and password verification, requires a centralized system. In a fully decentralized system, we rely on cryptographic signatures to provide authentication. While information isn't shared between the event chains, parties can still be identified across chains given the key pair used for signing.

In a broader sense, parties can sign any type of information this way. This is a similar use case as currently presented by PKI certificates. The reliance on Central Authorities to issue and revoke certificates has hindered the adoption of it as a replacement for challenge/response authentication.

With public blockchains, public/private key pairs can be created and used without the need for a central authority. A key pair forms a unique identity, which can be referenced via an address which is determined from the hash of the public key.

16.3 Certificates

A certificate transaction allows any identity to convey information about another identity by referencing its address. Unlike tokens, granting and revoking an account is fully under control of the issuer.

The certificate can be given a specific type chosen by the party that issued the certificate. While not required, it's recommended that a certificate is acknowledged by the recipient account before displayed to others.

16.4 Chain of trust

While a public address with a private key pair is a method of authentication, it doesn't provide a solution for authorization. Certificates can be used to specify relationships between identities.

This is a similar approach to the Web of Trust (WoT). The WoT has a number of drawbacks over PKI, which we do not have on our platform.

Establishing and revoking a relationship or marking an identity as compromised is simple, instant and irrevocable on the blockchain. Blockchain transactions are timestamped, which allows for verifying the existence of relationships on a certain point in time.

Rather than simply setting an identity, we set a specific relationship. The transaction doesn't confirm or deny any other information about the identity except the existence of the relationship, removing the need of physically meeting the other party.

In a given context, we only care about finding a chain of trust between two identities based on this relationship. This mimics the chain of trust as done with PKI validation but without the central authority. Rather than an absolute root certificate, the blockchain address of our organization or an organization we do business with functions as trust root.

16.5 Smart accounts

By default, any transaction for an account must be signed using the private key associated with the account. Waves introduced the concept of smart accounts, allowing anyone to customize this logic[78].

To do so, this logic can be scripted using a Non-Turing complete language. This script is only used to verify or deny a transaction for a specific account. It cannot trigger other transactions. As such, this type of smart contract does not prevent aggregation. A further limitation is placed on LTO smart accounts to ensure this.

LTO does not have data transactions, other transactions are not accessible from the script. You may validate whether a transaction has been signed by one or more other parties, enabling multi-signature. Rather than specifying these accounts directly, the requirement may reference a certificate instead.

There are other limitations one might place on an account as well. An account can be locked, only allow the transfer of tokens after a number of blocks, require a minimum number of tokens to remain on the account or only allow transferring tokens to specific accounts.

Anchoring transactions are an exception. They always need to be signed with the private key of the account. This logic is in place to apply future optimizations and can't be modified.

Using multi-signature is recommended when running a node. The account associated with the node typically holds a large number of tokens in order to stake and anchor. The private key to this account is known to the node. With multi-signature, obtaining that key won't give direct access to those tokens.

17 Summary blocks

Anchoring is a low-impact, non-disrupting method to bring an extra layer of security to the blockchain. We anticipate that other applications that do not use Live Contracts, will also use the anchoring feature.

One aspect of the blockchain that counteracts scalability is the fact that the blockchain will keep growing[79]. The size puts certain requirements on the hardware to keep a copy. It also puts a burden on new nodes that have to play back the whole chain. To reduce the growing speed of the chain, it uses summary blocks.

17.1 Key block size

Table 3 shows the structure of a key block on the blockchain. The global chain should be scalable up to $50\ million$ transactions per day. This is about five times the expected usage. The size of such a key block is determined by the block data and the transaction data (5).

$$Keyblock\ size = d + t \tag{5}$$

with:
$d =$ block data,
$t =$ transaction data.

Before calculating the expected blocksize, the following assumptions are made:

- 99.98% of the transactions are anchoring transactions (table 5). This is the main use of the global blockchain,

- The other 0.02% are issue certificate transactions (table 7),
- All other transactions are infrequent and can be neglected,
- All transactions are uniformly distributed over the blocks,
- On average one key block per minute is generated,
- Every day 1440 key blocks are created.

The block data is 277 bytes big (table 3). Under the assumptions made previously, the size of the transaction data can be calculated by equation (6).

$$Transaction\ data\ size = n \cdot (0.9998 \cdot a + 0.0002 \cdot c) \quad (6)$$

with:
$a =$ Size of an anchor transaction (table 5),
$c =$ Size of an issue certificate transaction (table 7),
$n =$ Amount of transaction per block.

This makes the total size of a key block about 3.8MB.

17.2 Growth without aggregation

With 3.8MB per block and 1440 blocks per day, the blockchain would grow 5.47GB per day / 2TB per year, if the blockchain would continuously run a full capacity.

The expected usage is on average $10\ million$ transactions, growing the blockchain 1.1GB per day. This results in about $3.65\ billion$ transaction or about 400GB per year.

For Bitcoin, with $340\ million$ transactions total[80], it takes about 7 days to synchronize from genesis, depending on network and hardware speed.

With billions of transactions, doing this naively could mean waiting weeks or even months for the global blockchain to synchronize.

One of the goals of aggregating transactions is to require only 20 minutes of synchronization per year, again depending on network and hardware speed. With 365 summery blocks in a year, a node should be able to process a summery block within 3 seconds.

17.3 Segregated witness

Segregated witness is a strategy employed in Bitcoin to reduce the data in a block[81] by separating a transaction into data that needs to be processed and data to verify the transaction in the block. This second part is called the witness data, containing signatures amongst other things.

Finality is the guarantee that blocks that are sufficiently deep will never be removed from the blockchain. Regardless of probabilistic finality or protocol finality, witness data is no longer useful if the block is guaranteed to not be reverted. Nodes are free to remove witness data for blocks that reached finality, saving disk space.

We've built on the logic of segregated witness for the concept of summary blocks.

17.4 Aggregation

Aggregation blocks are special blocks that are created every 1440 blocks (about once a day). They contain aggregated values of all the blocks since the previous summary block. When replaying the chain, only the summary blocks need to be applied to get close to the current state. After this, only the blocks created after the last summary block still have to be replayed. This decreases the replay time significantly.

The second to last summary block and all blocks before that are final. Nodes will not consider forks before that point, regardless of the longest chain. This means that only the transactions of the previous 500 to 1000 key blocks have to be stored. By removing transaction data from the key blocks after it has been stored in a summary block, key block size is reduced from 11.3MB to 277 bytes, making them negligible compared to the summary blocks.

17.5 Difference to pruning

At first glance, this approach has similarities to blockchain pruning, as you only maintain a limited set of transactions. The danger with pruning is in the introduction of a falsified state, threatening the immutability nature.

The danger comes from distributing the state of the blockchain. With segregated witness, transactions are applied without signature validation, relying on the concept of finality. Still, every node needs to apply all transactions from genesis to calculate the current state.

The actual transaction data is not used to calculate the block's signature but is rather stored as an attachment next to the block (table 2). As events without transactions, key blocks are part of the blockchain and can't be ignored. If transactions are applied without validation, aggregating them holds little risk.

17.6 Summary block size

Summary blocks contain all information about non-anchor transactions and an aggregated version of all the transaction fees and other token transfers. They are rather larger blocks, especially compared to the key blocks. To reduce the amount of memory used, the transaction fees and token transfer transactions get reduced to a balance change per participant (table (4)). Besides this aggregation, the summary will also contain the other transactions. To calculate the expected size of a summary block, the following assumptions are made:

- There is a total of 200000 participants,
- Every day 1 summary block is created,
- Based on the assumptions in the previous section we can assume that the balance change summary is the only significant part of a summary block.

Using these assumptions, the size of a summary block can be calculated. Using equation 7, this results in about 10.3MB.

$$Summary\ block\ size = Transaction\ summary = p \cdot e \quad (7)$$

with:
$p =$ Amount of participants in the last 1500 blocks,
$e =$ Size of a balance change summary entry (table 4.

17.7 Total size

The total size of the blockchain consists of two parts. A static part and a growing part. The static part consists of the last 1000 key blocks . Those will still contain the attachment data (5).

$$Static\ part = n \cdot k \quad (8)$$

with:
$n =$ Amount of keyblocks stored with transaction data,
$k =$ Size of a keyblock (5).

Using equation 8 results in the total size of the static part being about 11.3GB. Because only the last 1000 blocks are stored completely, which means including transactions, this size may differ slightly, but won't grow noticeably.

The growing part consists of the summary blocks (7) and what is left of the key blocks after the transaction data is removed. We'll define the size as growth per year using equation (9).

$$Growing\ part = n \cdot k + m \cdot s \tag{9}$$

with:
n = Number of keyblocks per year,
m = Number of summaryblocks per year,
k = Size of a keyblock without transaction data,
s = Size of a summaryblock (7).

Still following the previously made assumptions this result in the blockchain growing about 3.7GB per year.

17.8 History nodes

Nodes are not required to delete old transactions. By maintaining all transactions, history nodes are able to prove the correctness of the blockchain in case it's needed. History nodes are unable to pass a falsified history, as the blocks of the history node need to match those of other nodes. The network could rely on a relatively small amount of history nodes.

There is no on-chain benefit of running a history node. It doesn't increase the chance of forging new blocks. Running such a node must be done out of community interest or secondary income.

18 Network vulnerability

18.1 Importance inflation

A particular worry with PoI is the inflation of importance through dummy transactions. We can calculate the profit/loss from spam transactions as a formula of the maximum raffle factor;

- Raffle factor; r,
- Percentage of staked tokens; b_i,
- Cost of a transaction; c,
- Total transactions on network; n,
- Spam transactions; τ,
- Rewards; p,
- Profit/loss from spam; $\Delta p = p_{r_{max}} - p_{r=1}$.

$$p = (r \cdot b_i \cdot n \cdot c) - (\tau \cdot c) \tag{10}$$

$$r = 1, \tau = 0 \rightarrow p = b_i \cdot n \cdot c \tag{11}$$

$$r = r_{max}, \tau = b_i \cdot n \rightarrow (r_{max} - 1) \cdot b_i \cdot n \cdot c \tag{12}$$

This gives

$$\Delta p = ((r_{max} - 2) \cdot b_i \cdot n \cdot c) \tag{13}$$

Given

$$b_i > 0, n > 0, c > 0, \Delta p < 0 \rightarrow (r_{max} - 2) < 0 \tag{14}$$

$$r_{max} < 2 \tag{15}$$

Equation 15 proves that it's impossible to gain directly from spam transactions, with a maximum raffle factor of less than two.

A raffle factor close to two would make spam transactions nearly free. Increasing the importance of the network for little to no costs is undesirable, as it could aid an attacker trying to undermine the network with a 51% attack. The maximum raffle factor of 1.5 ensures a high cost of inflating importance.

18.2 Nothing at stake

The nothing at stake principle is the assumption that nodes will continue to build on any forks, rather than picking the longest chain, as there is no downside in doing so[82]. If all nodes would display such ill behavior, an attacker would only need a small percentage of tokens to force the network to switch to the other chain; a 1% attack.

Such a situation is a Tragedy Of The Commons[83]. All parties seek individual gain by abusing the system. But if everybody is doing it, nobody benefits. Instead, it only leads to undermining the network, causing a decrease in the value of the underlying token.

Waves Fair PoS has made it a lot more difficult for an unpublished orphaned branch to catch up with the main branch[74]. The possibility of profiting from such behavior with bad actors that only hold a small percentage of tokens is neglectable.

A bad actor would need to create and maintain an altered version of the node. The costs combined with the knowledge that there is little chance of benefiting from it, should be enough to disincentivize this behavior[84].

18.3 LPoS centralization

Projects that have implemented LPoS algorithm tend to lead to a high level of centralization.

This effect can be explained by the reward per token. A professional setup with a near 100% uptime will not miss a forging opportunity, yielding more reward with a set amount of tokens being staked. This draws token holders to lease to these nodes and has a reinforcing effect as reduced overhead allows for higher payouts to lessors.

Limiting the number of tokens per node is a flawed method enabling the opportunity for a Sybil attack[85]. In a permissionless reputation system, a single node can advert itself as multiple pseudonymous identities, circumventing this limitation.

The nature of our solution means a majority of transactions will be signed by a key pair associated with a node. The network will also consist of a relatively large number of nodes. This has no effect on PoS, while on PoI this gives the platform users an advantage over non-user token holders.

An additional measure is a limitation on leveraging on leased tokens; any node needs to own at least 10% of the tokens it stakes.

18.4 Denial of service attack

Due to the limited scalability, it's fairly easy to overload any public blockchain with too many transactions. Transaction fees are the primary defense against these sorts of attacks, but transactions still need to be verified to conclude there are insufficient funds. Even more, with a decent amount of funds, it's possible to overload the network with spam transactions as seen with Ethereum in July 2018.

Nodes have the option to automatically increase the transactions fees in case of such an attachment. Ideally, nodes gain as much from staking as is spend on transactions. As the spam tokens increase the rewards of transaction fees, this is used to automatically counter the attack.

18.5 SHA-2 vulnerability

In 2017 SHA-1 was proven to be vulnerable when the Google research lab managed to find a collision where two different documents resulted in the same hash[86]. If SHA-2 256bit would have a similar vulnerability this could be devastating for anchoring based on a Merkle tree.

If a collision is found, one can claim that the colliding document has been notarized. Even worse, given a Merkle root and a random hash, one might be able to generate a valid Merkle path. That would allow a hacker to verify any document. This would still not be a trivial task as for every branch in the tree, two hashes need to be combined that are exactly 32 bytes long.

While bruteforcing a specific SHA-1 would still require too many computations to achieve in a lifetime, the birthday paradox results in much fewer computations being required to find a collision. The birthday paradox also applies to the LTO public chain, as there are many Merkle roots, each with a maximum number of Merkle paths.

To overcome this, verification might fall back to history nodes in case the verification is disputed. However, this reduces the overall uses of anchoring nodes.

Instead a secondary Merkle tree might be added where the SHA-2 hash is hashed with another algorithm like SHA-3 or Blake2. Simply double hashing isn't useful when it's possible to falsify a Merkle, as only a vulnerability in the outer algorithm is required for an exploit. However, if there are two Merkle trees, both algorithms would need to be broken. Even then, a collision needs to be found for both trees of a single block, removing the birthday paradox advantage.

Part III. Platform

19 Architecture

19.1 Micro architecture

The LTO node is developed using the microservices architecture pattern. This means that all functionality within the node is split into microservices, with each service being responsible for only a small part of the entire node. There are several advantages to this pattern:

- Failure isolation, if a service fails it won't necessarily interfere with other services.
- Scalability, all the services within the node are decoupled and can therefore run on different machines. This makes it really suitable for horizontal scaling. The scaling is automated, which can be found in the description in section 24
- Flexibility, certain functionality flourishes better in certain programming languages. Each service can be developed in a different programming language.
- Code quality, by splitting the node into small and well-defined modules it becomes easier for developers to read and review. This leads to better code quality.

The microservices are grouped in a docker container. All these containers are run using the Kubernetes container orchestration platform. This will be described in section 24. Each microservice is designed to run independently. This means it has no shared dependencies, so each container has it's own database or event queue. Microservices are also designed to operate statelessly, so they are easily scalable.

19.2 Application layers and services

As described in section 19.1 the node is split up into several services. The services are grouped into 4 different layers. The node consists of the following layers:

- UI Layer: this consists of UI applications that interact with the application layer,
- Application layer: contains all the services that handle actions triggered by events from the event chain,
- Private chain layer: takes care of the decoupling of the node,
- Public chain layer: manages the public chain service. Our global public blockchain is optimized for storing hashes. Each node indexes all hashes, so they can be easily verified.

20 UI Layer

The UI layer contains two frontend applications which enable users to easily develop and debug their Live Contracts. First is the Chain Viewer. The chain viewer allows users to connect to a specific node and list all the chains. The user can only list and view chains of which he is a part. The second application is the playground application. The playground application lets users develop Live Contract scenarios. It visualizes the scenario in a state diagram and it contains other visualization and verification tools.

21 Application Layer

21.1 Web server

The web server application serves as a proxy between the frontend and the applications within the node. The web server performs two functions:

- Authentication of all requests to the services
- Proxies all requests to the correct service

21.2 Workflow engine

The actual creation and execution of the Live Contracts are done by the Workflow Service. If an event received by the Event Chain Service contains an action in the Live Contract it will be sent to the Workflow Service. The Workflow Service will execute the action which will lead to a state transition in the workflow and a new projection of the workflow. This projection is stored in a MongoDB database.

22 Private chain layer

The private chain layer decouples the node. Decoupling ensures a stable system even in case of bad connectivity or high load. The message queue is the communication layer for the private chain. The technology providing the message queue will be RabbitMQ. RabbitMQ is a lightweight message broker which is perfectly suited to deliver messages within the node but also to other nodes. RabbitMQ has a function called the Shovel. A Shovel will dynamically setup a connection with another RabbitMQ broker and exchanges messages. This mechanism is used to send events from one node to another.

Three services manage all inbound and outbound events.

22.1 EventChain service

The service which manages the private chains is the Event Chain service. This service processes all incoming events. During this processing of events the services take the following steps:

- Validate the incoming event(s), by checking if it's correctly signed and if the chain isn't broken.
- Validate if the chain matches the locally stored chain. If not, it performs conflict resolution where possible.
- If the event(s) are sent by the identity that belongs to this node it will execute the received event(s). Otherwise, it will only store them in the database.
- If a new identity is added from a different node, the whole chain is forwarded to this node.
- All the new events are forwarded to the related nodes.

For storage, the Event Chain service uses a MongoDB database.

22.2 Event enqueue service

The Event Enqueue service has the small task of putting events in the event queue. It does this both from services within the node (e.g. the Workflow Service and the Event Service) as well as from external users.

22.3 Event dispatch service

All messages in the event queue are handled by the Event Dispatch Service. It picks up all the messages and distributes them to the Event Service. If the Event Service processes the message it will be marked as handled. Otherwise it will be moved to the dead-letter queue.

23 Public chain layer

23.1 Anchor service

The Anchor service is at the heart of the public chain. The Anchor service will be a fork of the NXT platform extended with NG protocol. The Anchor service will be extended so it will not only be able to handle 'normal' transactions but also data transactions. These data transactions will be used to store hashes from events on the private chains as is described in section 16.1. All these hashes will be collected daily and merged deterministically into a Merkle Tree. This way the data transactions can be removed from storage to reduce the storage footprint but people are still able to verify whether the hash existed.

To be able to verify whether a certain hash is stored, all the hashes will be indexed. This way verification is much faster because you won't have to search through all the data transactions.

24 Container orchestration

Since the node is build up out of multiple microservices, a container orchestration platform is required to manage the running of the containers. Container orchestration platforms take care of a few tasks such as provisioning hosts, instantiating containers, restarting failed containers, scaling the cluster by adding or removing containers. Different container orchestration tools can be used for this purpose like Docker Swarm, Mesos, Nomad or Kubernetes. Initially, a configuration file will be included for Kubernetes. Each service will be configured to run in its own Pod with its own load balancer. This is done so individual services can scale independently of each other. Scaling of services is managed using the Horizontal Pod Autoscaler [87].

REFERENCES

[1] Hannah Ritchie Max Roser. *Technological Progress*. https://www.ft.com/content/cb56d86c-88d6-11e7-afd2-74b8ecd34d3b. Accessed: 03-06-2018. 2017.

[2] Christine Legner and Kristin Wende. "The challenges of inter-organizational business process design – a research agenda". In: (2007).

[3] Benjamin E. Hermalin and Michael L. Katz. "Moral Hazard and Verifiability: The Effects of Renegotiation in Agency". In: (1990).

[4] Audun Jøsang. "The right type of trust for distributed systems". In: *Proceedings of the 1996 workshop on New security paradigms*. ACM. 1996, pp. 119–131.

[5] Israel Z Ben-Shaul and Gail E Kaiser. "A paradigm for decentralized process modeling and its realization in the oz environment". In: *Proceedings of the 16th international conference on Software engineering*. IEEE Computer Society Press. 1994, pp. 179–188.

[6] Ray Fisman and Roberta Gatti. "Bargaining for bribes: The role of institutions". In: *International handbook on the economics of corruption* (2006), pp. 127–139.

[7] Jörg Becker, Michael Rosemann, and Christoph Von Uthmann. "Guidelines of business process modeling". In: *Business Process Management*. Springer, 2000, pp. 30–49.

[8] Manfred Reichert, Thomas Bauer, and Peter Dadam. "Enterprise-wide and cross-enterprise workflow management: Challenges and research issues for adaptive workflows". In: (1999).

[9] Vitalik Buterin. "Ethereum White Paper: A Next-Generation Smart Contract and Decentralized Application Platform". In: (2013).

[10] Vitalik Buterin and Karthik Gollapudi. *A Next-Generation Smart Contract and Decentralized Application Platform*. https://github.com/ethereum/wiki/wiki/White-Paper/f18902f4e7fb21dc92b37e8a0963eec4b3f4793a. Accessed: 22-05-2018.

[11] Ian Grigg. "The Ricardian Contract". In: (2004).

[12] Nick Sabo. "Formalizing and Securing Relationships on Public Networks". In: (1997).

[13] Telser. "A Theory of Self-Enforcing Agreements". In: (1980).

[14] David Joulfaian Douglas Holtz-Eakin and Harvey S. Rosen. "Sticking it out: entrepreneurial survival and liquidity constraints". In: (1993).

[15] *Toshi wallet now supports ERC20 tokens and ERC721 collectibles*. https://blog.toshi.org/toshi-wallet-now-supports-erc20-tokens-and-erc721-collectibles-e718775895aa. Accessed: 30-08-2018.

[16] *ERC-20 Token Standard*. https://eips.ethereum.org/EIPS/eip-20. Accessed: 30-08-2018.

[17] Daniel IA Cohen and Daniel IA Cohen. *Introduction to computer theory*. Vol. 2. Wiley New York, 1991.

[18] Marko Vukolić. "Rethinking permissioned blockchains". In: *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM. 2017, pp. 3–7.

[19] AbdulSalam Kalaji, Rob Mark Hierons, and Stephen Swift. "A search-based approach for automatic test generation from extended finite state machine (EFSM)". In: *Testing: Academic and Industrial Conference-Practice and Research Techniques, 2009. TAIC PART'09.* IEEE. 2009, pp. 131–132.

[20] M.G. Gouda, E.G. Manning, and Y.T. Yu. "On the Progress of Communication between Two Finite State Machines". In: (1984).

[21] G Pace and J Schapachnik. "Contracts for Interacting Two-Party Systems". In: (2012).

[22] Mark D. Flood and Oliver R Goodenough. "Contract as Automaton: The Computational Representation of Financial Agreements". In: (2015).

[23] Davide Basile, Pierpaolo Degano, and Gian-Luigi Ferrari. "Automata for Service Contracts". In: (2014).

[24] Pierpaolo Degano Davide Basile and Gian-Luigi Ferrari. "From Orchestration to Choreography through Contract Automata". In: (2014).

[25] Ian Ayres and Robert Gertner. "Filling Gaps in Incomplete Contracts: An Economic Theory of Default Rules". In: (1989).

[26] Jan L.G. Dietz. "Understanding and Modeling Business Processes with DEMO". In: (1999).

[27] YoungJoon Byun, Beverly A. Sanders, and Chang-Sup Keum. "Design Patterns of Communicating Extended Finite State Machines in SDL". In: (2001).

[28] Petri. "Kommunikation mit Automaten". In: (1962).

[29] Dennis Kafura. *Notes on Petri Nets*. http://people.cs.vt.edu/kafura/ComputationalThinking/Class-Notes/Petri-Net-Notes-Expanded.pdf. Accessed: 01-09-2018.

[30] Wil M.P. van der Aalst. "The Application of Petri Nets to Workflow Management". In: (1998).

[31] Jan Recker et al. "How good is bpmn really? Insights from theory and practice". In: (2006).

[32] Wil M.P. van der Aalst et al. "Life After BPEL?" In: (2005).

[33] Luciano García-Bañuelos et al. "Optimized Execution of Business Processes on Blockchain". In: (2017).

[34] Jan L.G. Dietz. "DEMO: Towards a discipline of organisation engineering". In: (1999).

[35] *JSONForms - React*. https://jsonforms.io/. Accessed: 30-08-2018.

[36] *JSONForm - Bootstrap 3*. https://github.com/jsonform/jsonform. Accessed: 30-08-2018.

[37] *Mozilla react-jsonschema-form*. https://github.com/mozilla-services/react-jsonschema-form. Accessed: 30-08-2018.

[38] *Angular Schema Form*. http://schemaform.io/. Accessed: 30-08-2018.

[39] *Open Document Format*. http://www.opendocumentformat.org/. Accessed: 30-08-2018.

[40] Sindhu Sajana and Sethumadhavan. "On Blockchain Applications: Hyperledger Fabric And Ethereum". In: (2018).

[41] Stephen A Cook. "The complexity of theorem-proving procedures". In: *Proceedings of the third annual ACM symposium on Theory of computing*. ACM. 1971, pp. 151–158.

[42] Daniel Brand and Pitro Zafiropulo. "On communicating finite-state machines". In: *Journal of the ACM (JACM)* 30.2 (1983), pp. 323–342.

[43] Robert M Hierons AbdulSalam Kalaji and Stephen Swift. "New approaches for passive testing using an Extended Finite State Machine specification". In: (2003).

[44] O Sury and R Edmonds. *Edwards-Curve Digital Security Algorithm (EdDSA) for DNSSEC*. Tech. rep. 2017.

[45] NIST. *Transition Plans for Key Establishment Schemes using Public Key Cryptography*. https://csrc.nist.gov/News/2017/Transition-Plans-for-Key-Establishment-Schemes. Accessed: 13-07-2018. 2017.

[46] Daniel J. Bernstein et al. "High-speed high-security signatures". In: *Journal of Cryptographic Engineering* 2.2 (2012), pp. 77–89. ISSN: 2190-8516. DOI: 10.1007/s13389-012-0027-1. URL: https://doi.org/10.1007/s13389-012-0027-1.

[47] Henri Gilbert and Helena Handschuh. "Security analysis of SHA-256 and sisters". In: *International workshop on selected areas in cryptography*. Springer. 2003, pp. 175–193.

[48] NIST. *NIST Policy on Hash Functions*. https://csrc.nist.gov/Projects/Hash-Functions/NIST-Policy-on-Hash-Functions. Accessed: 13-07-2018. 2015.

[49] Bruce Schneier and John Kelsey. "Cryptographic Support for Secure Logs on Untrusted Machines." In: *USENIX Security Symposium*. Vol. 98. 1998, pp. 53–62.

[50] Peter Bailis and Ali Ghodsi. "Eventual consistency today: Limitations, extensions, and beyond". In: *Queue* 11.3 (2013), p. 20.

[51] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[52] Miguel Castro and Barbara Liskov. *Byzantine fault tolerance*. US Patent 6,671,821. 2003.

[53] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf. Accessed: 17-05-2018.

[54] Aggelos Kiayias et al. "Ouroboros: A provably secure proof-of-stake blockchain protocol". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 357–388.

[55] POA Network. *Proof of Authority: consensus model with Identity at Stake.* https://medium.com/poa-network/proof-of-authority-consensus-model-with-identity-at-stake-d5bd15463256. Accessed: 17-05-2018.

[56] Git Documentation. *Git Branching - Rebasing*. https://git-scm.com/book/en/v2/Git-Branching-Rebasing. Accessed: 09-08-2018.

[57] Aaron van Wirdum. "Rejecting Today's Hard Fork, the Ethereum Classic Project Continues on the Original Chain: Here's Why". In: *Bitcoin Magazine* 20 (2016).

[58] European Parliament. *REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL: on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN. Accessed: 12-07-2018. 2016.

[59] Olly Jackson. "Is it possible to comply with GDPR using blockchain?" In: *International Financial Law Review* (2018).

[60] S Goldwasser, S Micali, and C Rackoff. "The knowledge complexity of interactive proof systems". In: (1989).

[61] *Blockchain costs per transaction*. https://www.blockchain.com/charts/cost-per-transaction. Accessed: 05-09-2018.

[62] Emanuel Palm. *Implications and Impact of Blockchain Transaction Pruning*. 2017.

[63] Wenting Li et al. "Towards scalable and private industrial blockchains". In: *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM. 2017, pp. 9–14.

[64] Serguei Popov. "A probabilistic analysis of the nxt forging algorithm". In: *Ledger* 1 (2016), pp. 69–83.

[65] Waves platform. *WAVES whitepaper*. https://blog.wavesplatform.com/waves-whitepaper-164dd6ca6a23. Accessed: 16-07-2018. 2016.

[66] *Chainpoint Node API: How to Create a Chainpoint Proof*. https://github.com/chainpoint/chainpoint-node/wiki/Chainpoint-Node-API:-How-to-Create-a-Chainpoint-Proof. Accessed: 05-09-2018.

[67] Gleb Kostarev. *Review of blockchain consensus mechanisms*. https://blog.wavesplatform.com/review-of-blockchain-consensus-mechanisms-f575afae38f2. Accessed: 13-07-2018. 2017.

[68] NEM. *NEM technical reference*. https://nem.io/wp-content/themes/nem/files/NEM_techRef.pdf. Accessed: 13-07-2018. 2018.

[69] LTO. "LTO Token Economy". In: (2018).

[70] Waves Platform. *Blockchain Leasing For Proof Of Stake*. https://blog.wavesplatform.com/blockchain-leasing-for-proof-of-stake-bac5335de049. Accessed: 13-07-2018. 2018.

[71] mthcl. "The math of Nxt forging". In: (2014).

[72] *Waves generators*. http://dev.pywaves.org/generators/. Accessed: 28-08-2018.

[73] *Nxt Blockchain Explorer*. https://nxtportal.org/monitor/. Accessed: 28-08-2018.

[74] Kofman Begicheva. "Fair Proof of Stake". In: (2018).

[75] *Waves-NG stress test: results in!* https://blog.wavesplatform.com/waves-ng-stress-test-results-in-44090f59bb15. Accessed: 05-09-2018.

[76] S. Haber and W.S. J Stornetta. "How to time-stamp a digital document". In: (1991).

[77] Ralph C. Merkle. "Method of providing digital signatures". U.S. pat. 4309569. Jan. 5, 1982.

[78] A. Begicheva and I. Smagin. "RIDE: a Smart Contract Language for Waves". Pat. 2018.

[79] Saifedean Ammous. "Blockchain Technology: What is it good for?" In: (2016).

[80] *Blockchain number of transaction*. https://www.blockchain.com/en/charts/n-transactions-total. Accessed: 05-09-2018.

[81] *BIP 141: Segregated Witness (Consensus layer)*. https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki. Accessed: 05-09-2018.

[82] *Problems Ethereum*. https://github.com/ethereum/wiki/wiki/Problems. Accessed: 30-08-2018.

[83] G Hardin. "The Tragedy of the Common". In: (1969).

[84] *Nothing considered a look at nothing at stake vulnerability for cryptocurrencies*. https://pivx.org/nothing-considered-a-look-at-nothing-at-stake-vulnerability-for-cryptocurrencies/. Accessed: 30-08-2018.

[85] John R Douceur. "The sybil attack". In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 251–260.

[86] Marc Stevens et al. "The first collision for full SHA-1". In: (2017).

[87] Kubernetes. *Kubernetes, Horizontal Pod Autoscaling*. https://github.com/kubernetes/community/blob/master/contributors/design-proposals/autoscaling/horizontal-pod-autoscaler.md. Accessed: 03-08-2018.

#	Field Name	Length
1	Version	1
2	Timestamp	8
3	Parent block signature	64
4	Consensus block length	4
5	Base target	8
6	Generation Signature	32
7	Transaction list Hash	32
8	Anchor Merkle root	32
9	Generator public key	32
10	Block's signature	64

TABLE 1: Key block structure

#	Field Name	Length
1	Amount of transactions (x)	4
2	Transaction #1 bytes	113
...	...	...
2 + (K - 1)	Transaction #K bytes	113

TABLE 2: Key block attachment

#	Field Name	Length
1	Version	1
2	Timestamp	8
3	Parent block signature	64
4	Consensus block length	4
5	Base target	8
6	Generation Signature	32
7	Transaction list Hash	32
8	Transaction #1 bytes	TODO
...	...	...
$8 + (K - 1)$	Transaction #K bytes	TODO
$9 + (K - 1)$	Balance change summary entry #1	40 (Table 4)
...	...	...
$9 + (K - 1) + (N - 1)$	Balance change summary entry #N	40 (Table 4)
$10 + (K - 1) + (N - 1)$	Generator public key	32
$11 + (K - 1) + (N - 1)$	Block's signature	64

TABLE 3: Summary Block structure

#	Field Name	Length
2	Wallet address	32
3	Balance change	8

TABLE 4: Balance summary entry

#	Field Name	Length
1	Transaction type	1
2	Anchor hash	32
3	Fee	8
4	Timestamp	8
5	Signature	64

TABLE 5: Anchor transactions structure

#	Field Name	Length
1	Transaction type	1
2	Sending address	32
3	Receiving address	32
4	Amount	8
5	Fee	8
6	Timestamp	8
7	Signature	64

TABLE 6: Transfer transaction structure

#	Field Name	Length
1	Transaction type	1
2	Id	32
3	Sending address	32
4	Receiving address	32
5	Expiration Date	8
6	Certificate Type	32
7	Fee	8
8	Timestamp	8
9	Signature	64

TABLE 7: Issue certificate transaction structure

#	Field Name	Length
1	Transaction type	1
2	Id	32
3	New expiration Date	8
4	Fee	8
5	Timestamp	8
6	Signature	64

TABLE 8: Update certificate transaction structure

#	Field Name	Length
1	Transaction type	1
2	Sending address	32
3	Receiving address	32
4	Amount	8
5	Fee	8
6	Timestamp	8
7	Signature	64

TABLE 9: Lease transaction structure

#	Field Name	Length
1	Transaction type	1
2	Sending address	32
3	Receiving address	32
4	Amount	8
5	Fee	8
6	Timestamp	8
7	Signature	64

TABLE 10: Cancel Lease transaction structure