The Performance, Interoperability and Integration of Distributed Ledger Technologies

Emanuel K. Palm

Dept. of Computer Science and Electrical Engineering
Luleå University of Technology
Luleå, Sweden

Supervisors:

Ulf Bodin, Olov Schelén and Jerker Delsing

To my beloved wife, Sofia.

Abstract

In the wake of the financial crisis of 2008, Bitcoin emerged as a radical new alternative to the fiat currencies of the traditional banking sector. Through the use of a novel kind of probabilistic consensus algorithm, Bitcoin proved it possible to guarantee the integrity of a digital currency by relying on network majority votes instead of trusted institutions. By showing that it was technically feasible to, at least to some extent, replace the entire banking sector with computers, many significant actors started asking what else this new technology could help automate. A subsequent, seemingly inevitable, wave of efforts produced a multitude of new distributed ledger systems, architectures and applications, all somehow attempting to leverage distributed consensus algorithms to replace trusted intermediaries, facilitating value ownership, transfer and regulation.

In this thesis, we scrutinize distributed ledger technologies in terms of how they could help facilitate the digitization of contractual cooperation, especially in the context of the supply chain and manufacturing industries. Concretely, we consider them from three distinct technical perspectives, (1) *performance*, (2) *interoperability* and (3) *integration*. Voting systems, with or without probabilistic mechanisms, require significant time and resources to operate, for which reason it becomes relevant to investigate how the costs of running those systems can be mitigated. In particular, we consider how a blockchain, a form of distributed ledger, can be pruned to in order to reduce disk space requirements. Furthermore, no technical system part of a larger business is an island, but will have to be able to interoperate with other systems to maximize the opportunity for automation. For this reason, we also consider how transparent message translation between systems could be facilitated, as well as presenting a formalism for expressing the syntactic structure of message payloads. Finally, we propose a concrete architecture, the *Exchange Network*, that models contractual interactions as negotiations about token exchanges rather than as function invocations and state machine transitions, which we argue lowers the barrier to compatibility with conventional legal and business practices.

Even if no more trusted institutions could be replaced by any forthcoming distributed ledger technologies, we believe contractual interactions becoming more digital would lead to an increased opportunity for using computers to monitor, assist or even directly participate in the negotiation, management and tracking of business agreements, which we see as more than enough to warrant the cost of further developing of the technology. Such computer involvement may not just save time and reduce costs, but could also enable new kinds of computer-driven economies. In the long run, this may enable new levels of resource optimization, and not just within large organizations, but also smaller companies, or even the homes of families and individuals.

Contents

Acknowledgments

This body of scientific effort is not just the work of my own hands, but also of a great number of people who have assisted or directed me in my work. Firstly, I would like to recognize that this thesis would not exist without the funds provided through the *Productive 4.0* project,[1] as well as the people who secured the funding for EISLAB at Luleå University of Technology and were involved in recruiting me. Secondly, I thank my supervisors, Ulf Bodin, Olov Schelén and Jerker Delsing, each of which has provided me with indispensable support and guidance in learning the scientific process, mindset and culture. Thank you for believing in my ideas and helping realize them. Thirdly, I want to express my gratitude for my closest coworkers, Fernando Ramirez, Jaime Garcia, Cristina Paniagua, Jacob Nilsson, Kiarnmayee Katyayani Kolluru, Simon Jonsson Lahdenperä, Lara Lorna Jimenez, Joakim Nilsson, Sergio Martin Del Campo Barra, Niklas Karvonen, as well as all other collegues I've had at LTU. You made my time as a Ph.D. student a rewarding and enjoyable experience. Fourthly, I would like to acknowledge the positive influce of the partners I had the pleasure of working with during my research studies. These include, but are not limited to, Oscar Carlsson of Midroc; Kiran Shekhar, Noah Winneberger and Till Witt of NXP; Caroline Berg von Linde, Johan Hörmark, Christian Lagerkvist and Jamie Walters of SEB; as well as Richard Hedman of Volvo Group Trucks Operations. Thank you for helping me better understand the needs and wants of the industry, and also for your ever so witty jokes. Finally, I want to express my deepest appreciation and admiration for my wife, who has helped me in more ways than I can account for. Truly, this work would not have been possible without your loving support.

"The Lord *is my strength and song, and he is become my salvation ..."*
(Exodus 15:2)

Luleå, June 2019
Emanuel Palm

[1]EU ARTEMIS JU grant agreement number 737459.

Part I

Chapter 1
Introduction

The last few decades has seen more and more aspects of human life entering into the digital domain. Voice communication, television, banking, governmental services and social networks are just a few examples. This development is largely driven by its economical advantages. When information becomes directly accessible to computers, it can be analyzed or managed automatically, which improves the effectiveness of, or reduces the need for, human labor. Not all domains of human work are straightforward to digitize, however. It is, for example, not a coincidence that we use the word *computer* to refer to our electronic computing machines. The job of the human computer, someone tasked with executing given calculations, was one of the first to be automated by the machine. A computer machine *is* an automatable calculation device, which should make it very apparent why it could replace the human computer early in its history. In order to make the computing machine able to automate more kinds of human labor, however, it needed to be endowed with more fundamental capabilities. Computer networking, speech-to-text, vision, among many other such examples, were later created and can today be built upon to enable the automation of increasingly complex use cases.

One particular computer capability emerged from curious circumstances toward the end of 2008 and the beginning of 2009. In the wake of the global financial crisis, the pseudonym Natoshi Sakamoto published a paper [1] and the source code of a novel kind of computer software. The software, *Bitcoin*, was created as an alternative to the fiat currencies of the traditional banking sector, which at this point many considered to have lost its credibility. Through the use of a novel kind of probabilistic consensus algorithm, Bitcoin proved it possible to guarantee the integrity of a digital currency by relying on a substitute for network majority votes, *proof-of-work*, instead of on trusted institutions. By showing that it was technically feasible to, at least to some extent, replace the entire banking sector with voting computers, many significant actors, banks included, started asking what else this new technology could help automate. A subsequent, and perhaps inevitable, wave of initiatives attracted enough funding to produce a multitude of other distributed ledger systems, architectures and applications, all somehow attempting to leverage consensus algorithms to replace trusted intermediaries, that hitherto had been facilitating value ownership, transfer and regulation, among other things.

1.1 Motivation

The technological capability introduced with Bitcoin, often referred to as *"the blockchain"*, allows stakeholders with conflicting interests to maintain and append transactions to a shared immutable ledger. Perhaps a bit unintuitively, this capability could have major implications not just for conventional banks or digital currencies, but also for industries, which may become able to integrate their value chains across stakeholder boundaries without any mediators [2] [3]. In other words, distributed ledger technologies may enable cooperating parties to digitize and automate contract handling, asset transfer, identity handling and other forms of contractual interaction. This could serve to improve the flexibility and effectiveness of contemporary manufacturing processes, without the cost in money and loss of control typically incurred when entrusting a party as mediator. Eventually, the technology may help facilitate economies largely dominated by machine agents, autonomously buying and selling, producing and cooperating, all in the service of their owners.

However, before proceeding to describe how these opportunities relate to the objective of this thesis, we want to stop and consider the structures that incentivized us to pursue them. In particular, the research in this thesis was funded via the *Productive 4.0* project,[1] which is also known by the more formal name *EU ARTEMIS JU grant agreement number 737459*. The project is funded in part by the European Union via ECSEL JU,[2] and in part by the 109 organizations participating in the project. By implication, our efforts were shaped both by the overarching ambition of the project, which is to further digitize and interconnect contemporary industrial processes, and by the partners that decided to cooperate with us, which all have their own reasons for deciding to participate in such a research project. Concretely, the companies most actively contributing to our work were Volvo Trucks, SEB, NXP and Midroc. The names of the individuals concretely making those contributions are listed in the Acknowledgments section earlier in the thesis. While the needs and insights of those companies and individuals certainly have affected our contributions, we have strived to only pursue generally applicable results.

1.2 Problem Description

The primary objective of this thesis is to investigate the requirements for meaningfully digitizing contractual cooperation, with particular emphasis on the potential utility of distributed ledger technologies. As fulfilling this objective in its entirety is way beyond any reasonable scope of a thesis such as this, we divide it into three distinct perspectives, (1) performance, (2) interoperability and (3) integration, and then address a limited number of issues associated with each such. Before summarizing these perspectives as three research questions, we first describe them and how they relate to the research papers we list in Chapter 4. The research questions are revisited in Chapter 5, where we consider the extent to which we answer them.

[1]See `https://productive40.eu`.
[2]See `https://www.ecsel.eu`.

1.2.1 Perspectives

1. **Performance**. Computers tend to work orders of magnitude faster than humans, which is one significant reason they have become such important instruments of automation. However, a computer's *practical* speed of operation depends on many factors, among which is the complexity of the task it is set to execute. Distributed ledger applications tend to require comparatively high numbers of messages to be sent between peers to have them reach consensus, as well as requiring significant storage space to store proposed or finalized transactions [4]. Reducing the amount of required disk space is particularly important if wanting to have less capable, or very long-lived, machines participate in distributed ledger networks, which may be of particular interest in industrial contexts. We approach these issues in two ways. Firstly, we consider the implications of shrinking a blockchain, a form of distributed ledger, by pruning it of less relevant data, and, secondly, we consider alternatives to the consensus mechanisms typically used by these systems. The first way is the main theme of Paper A, while the second is considered as part of Paper C.

2. **Interoperability**. At this point in history, industrial production systems tend to be staggeringly complex [5], often involving large numbers of machines, systems and processes, which may have been accumulated and upgraded over decades. While distributed ledger technologies could help improve flexibility and effectiveness of these systems, they must communicate with their computer components to be of any practical utility. One ideal could be to have some kind of means able to translate any system interface into another such compatible with some other system, a kind of universal adapter for computer communications. We contribute to the formulation of such a means by outlining a conceptual method for message payload translation in Chapter 3.3, building on previous efforts on message protocol translation by H. Derhamy et al. [6], and then present a concrete way of modelling the basic structures of that method in Paper B.

3. **Integration**. Manufacturing systems, power plants, supply chains, as well as other forms of industrial enterprises, consist not only of mechanical and computer parts. Those parts belong to larger corporate structures, shaped by customer demands, business visions, resource planning systems, contractors, environmental regulations, and so on. Distributed ledger systems tend to diverge from other technologies in that they operate across both (1) organizational and (2) structural boundaries, joining different companies and many of their systems, of both computers and humans, into larger systems of systems. As using distributed ledger technologies may require significant changes to business practices, integrating them becomes especially challenging. In Papers C and D we present an architecture that could be implemented on top of a distributed ledger solution, in which inter-organizational interactions are modeled as negotiations about ownerships. We consider how implementing that architecture affects cooperation governance, privacy and performance. In Paper D in particular, we investigate an implementation strategy we argue lower the barriers to compatibility with conventional legal and business practices.

1.2.2 Research Questions

Q1 *What possibilities exist for making distributed ledgers require less disk space, even if assuming new data will be added indefinitely?*

Q2 *What would be required to facilitate seamless integration between legacy systems and distributed ledger technologies?*

Q3 *How can distributed ledger systems be fitted into the cooperational structures already existing between industrial stakeholders?*

1.3 Methodology

In a sense, the scientific method is both an ideal and a journey. An ideal, since the outcome it is designed for, objective and applicable knowledge, is hard to acquire. A journey, since the road from a scientific objective to a scientific achievement it is full of haphazard roadblocks and meaningful learning experiences.

My work started with a thorough introduction to the aims and accomplishments associated with the Productive 4.0 project and the Arrowhead Framework [7], as well as being tasked with investigating the potential seen in the blockchain concepts [8] [9]. As my Master's thesis was about pruning transactions from blockchains [10], it was considered relevant enough for me to rewrite into a conference paper. Around the same time, I started to meet with representatives from companies part of the Productive 4.0 project to discuss their visions and wants related to distributed ledger technologies. It took about a year of discussions, ideas and arguments before it started to become clear what kind of system these partners would benefit from. Before then, I finished a configuration system with message translation capabilities as part of a course, and was directed to write a paper about it. Having written two papers and documentation for a conceptual Arrowhead system, I created a limited implementation of the systems discussed with our partners. After finishing the implementation, I was able to write two papers about it before having to focus on writing this thesis.

How does this story relate to the scientific method? All of my efforts were coordinated together with my supervisors, which means that even though all nuances of scientific valuation within our discipline were not known to me, my supervisors were actively ensuring the quality of my work. Later, I learned that I was directed according to the philosophy of *Experimental Computer Science and Engineering*, which *"involves the creation of, or the experimentation with or on, computational artifacts"* [11]. Even though I never set out to explicitly adhere to any formalized methodology, there are, however, significant parallels between the way my research was conducted, *Design Research* [12], and *Case Study Research* [13]. The former is about identifying business problems and generalising their solutions, while the latter is about investigating existing scenarios and proposing improvements. In my case, the business problems and scenarios were given, together with abstract visions regarding their solution. However, it is hard for me to evaluate if strict adherence to either of these would have yielded stronger results.

1.4 Scope

Distributed ledger technologies tend to be of significant complexity, involving the use of distributed consensus algorithms, computer language evaluators, cryptography, gossip algorithms, and so on. Rather than being concerned mainly with any one constituent of a distributed ledger system, we focus on the technical requirements for making such a system become a meaningful component of a larger enterprise. In other words, we try to generally understand the technical conditions required to digitize inter-organizational cooperation, while also making concrete proposals for new constituents where existing ones are perceived as insufficient. With the exception of Paper A, this implies that we consider performance only theoretically, when we consider it at all. At this stage, it also means that the technologies we concretely propose are intended to demonstrate how certain problems could be solved differently, not how they should be solved in real-world scenarios.

1.5 Outline

This compilation thesis is organized into two major parts. Part I, to which this outline belongs, contains a description of the context that shaped and directed our concrete research contributions, which are contained in Part II.

The rest of Part I is organized as follows. In Chapter 2, we briefly describe the current state-of-the-art in distributed ledger technology, including descriptions of how the technology operates and common variants of it. In Chapter 3, we extrapolate current trends and our contributions into a road-map to a future where machines are able to make financial decisions and form their own economies. Further, in Chapter 4 we describe how our research papers fit into the context in this thesis, while, finally, Chapter 5 ends the thesis with a discussion, our conclusions, ethical considerations and our thoughts on what ought to be the focus of the distributed ledger community in the next few years.

Chapter 2

Distributed Ledger Technologies

A *distributed ledger* is a log of events, maintained and updated by the members of a distributed ledger network. The name comes from the accounting field, where ledgers of currency transactions are used to record the movement of money between accounts. They can be used for many other things than for tracking account balances, however. Ledgers can record asset ownership transfers, vehicle inspections, assignment updates, among many other things. While being able to record such events within a business is interesting enough, a *distributed* ledger allows businesses and other entities to reliably record interactions *between* themselves. Through clever use of cryptography and voting, distributed ledger technologies can be used to ensure no practical opportunity exists for any collaborating party to tamper with the events they record, which means that they can be used as evidence in different kinds of regulation frameworks. There is, however, currently no one way to implement these kinds of ledgers that suits all relevant types of use cases. That, taken together with the large interest in the technology, has resulted in a plethora of very distinct kinds of systems, with different approaches to programmatic verification, compatibility with legal institutions, consensus mechanisms, and so on.

2.1 Overview

In this chapter, we briefly consider the technological developments that lead to the making of the Bitcoin cryptocurrency, the explosion of derivatives that came after it, and the technical motivations behind their creation. We also make an attempt to categorize and compare a handful of popular or otherwise relevant distributed ledger solutions. Finally, we close the chapter with a description of the primary constituents of a distributed ledger system, how those constituents can be designed, and how they maintain the properties of the system they constitute.

The chapter is not meant to be exhaustive. It should, however, provide interested readers without strong knowledge on the topic an adequate introduction. If more depth on the subject is desired, we advice looking at the references we use in this chapter, as they should represent significant, or otherwise relevant, works within the field.

2.2 History

While possible to begin a history like this at many different dates and times, we start a few years after 1990. At this point, the Internet was taking the step from research labs and early adopters into the world of the general public.[1] The *World Wide Web* had recently been created by Tim Berners-Lee at CERN, and was now starting to find its way to the homes of the growing group of people owning personal computers. More investors started to gravitate towards the Internet, which would eventually lead to the burst of the dotcom bubble at the end of the decade. During this time of new ideas and increasingly inflated expectations, efforts at digitizing contracts and money started to gain traction.

Nick Szabo, a young computer scientist, was working on his idea of a *"computerized transaction protocol that executes the terms of a contract"*, or the *"smart contract"* [14] [15]. Around the same time, Ian Grigg and Gary Howland were working on their Ricardo payment platform, which, notably, involved *"a method to identify and describe issues of financial instruments as contracts"*, producing so-called *"Ricardian contracts"* [16] [17]. Both the smart and the Ricardian contracts were concrete attempts to create digital counter-parts to the conventional contract, but differed, however, in their generality, relation to traditional legal authorities, and focus on automated regulation.[2]

During the same general period, in 1992, the *cypherpunk* movement was founded, dedicated to using cryptography to protect the privacy of individuals from governments, corporations and other large organizations [20]. The members of the movement directly created, or otherwise influenced the creation of, *HashCash*, *B-money*, *bit gold*, as well as other experimental solutions for representing money in digital networks [21]. The many minor achievements of the movement culminated with the introduction of Bitcoin later in 2008 [1], which combined Szabo's smart contracts, immutable logs, a novel variant of Byzantine fault tolerance, as well as cryptographic identities, into one useful application. Through its subsequent and spectacular rise to popularity, Bitcoin proved it viable to run a global payment network without a central authority, and, consequently, also helped cement the technologies it built upon as reliable and acceptable in the public mind.

However, with Bitcoin's rise to prominence, it became increasingly apparent that the system has a number of significant limitations. For example, it only supports a limited form of smart contracts, *Scripts* [22], it provides neither strong anonymity or strong identity guarantees [23], it has proven to have significant operational costs [24], as well as having high latency and low transaction throughput [25] [26] [27], all of which make it unfit for many otherwise compelling use cases. Consequently, a plethora of alternative solutions have been built in order to cater for different categories of use cases. Significant examples include *Ethereum* [28], which provides extended smart contract capabilities, *ZeroCash* [29], which facilitates stronger anonymity guarantees, *Hyperledger Fabric* [30] [31], which may only be used within groups of well-known participants, *Lightning Network* [32], which extends Bitcoin with high-frequency payment channels, among many other.

[1]See `https://www.internetsociety.org/history` for an excellent description of this history.

[2]Other relevant efforts predating the 2008 Bitcoin launch include *OASIS LegalXML* from 2001 [18] and *RuleML* from 2005 [19], both of which are attempts to represent legal documents in a machine-verifiable form. These examples are likely to represent only a small fraction of all that could be mentioned.

2.3 Classification

At the time of writing, the number of distributed ledger technologies in existance has since long been too unwieldy to adequately consider all of them in a context like this. We, therefore, focus on well-known or otherwise relevant examples with particular bearing on our work. Concretely, we briefly consider Bitcoin [1], Ethereum [28], Hyperledger Fabric [31] and R3 Corda [33]. These solutions represent two major ends of the distributed ledger spectrum, the *permissionless* and the *permissioned*, as well as four distinct strategies at regulating ledger updates. A summary of their properties is outlined in Table 2.1.

Table 2.1: Significant properties of four distributed ledger solutions, presented in terms of the constituents introduced in Section 2.4. Note that some of the differences between these systems are not visible. For example, even though Bitcoin and Ethereum use different, albeit similar, consensus mechanisms, their properties seem identical. The reason for this is that we are concerned with highlighting distinctions that impact how these solutions must and can be used, which rules out details about performance improvements, state data structures, and so on.

	Permissionless		*Permissioned*	
	Bitcoin	**Ethereum**	**H. Fabric**	**R3 Corda**
Proofs				
– Identity	Public Key	Public Key	Certificate	Certificate
– Authorship	Signature	Signature	Signature	Signature
– Validity	Proof-of-Work	Proof-of-Work*	Vote, by Peers	Vote, by Notaries
Consensus				
– Visibility	Global	Global	Channel Only	Peer Only
– Resolution	Probabilistic	Probabilistic	Definitive	Definitive
– Throughput	Lower	Lower	Higher	Higher
Contracts				
– Validation	Yes, via Scripts	Yes, via Contracts	Yes, via Policies	Yes, via Flows
– Embedding	Yes, in UTXOs	Yes, in Auto. Agents	No	No
– Legal Integration	No	No	No	Yes, via Legal Prose

*We ignore that Ethereum also can operate using *Proof-of-Stake* or *Proof-of-Authority* to simplify our comparison.

2.3.1 Permissionless and Permissioned

Distributed ledger solutions such as Bitcoin and Ethereum operate without their users explicitly creating any accounts. Rather, each user is represented by a public key [34], often referred to as an *address*, created by each user in isolation. Each possible address *is* an account, and access to the corresponding private key is what allows the account to be controlled. As no central authority regulates account creation, these systems are often referred to as being *permissionless*. Bitcoin and Ethereum *can be* and are permissionless because they rely on the kind of consensus algorithm first introduced by S. Nakamoto in [1], which does not assume complete knowledge about all system participants. As identities are anonymous but activities public, users are said to be *pseudo-anonymous*.

When systems such as Hyperledger Fabric and R3 Corda were designed, this lack of control over users was regarded as a problem. Therefore, they require that each user is represented by a *certificate*, which contains both a public key and details about the entity represented by that key. Those certificates must, in addition, somehow be established to be authentic. Consequently, these systems are referred to as being *permissioned*, as permission is required by an arbitrary registrar to join any of their networks. Also, as solutions such as Fabric and Corda were designed to be used, primarily, within smaller groups of well-known participants, consensus algorithms like PBFT [35] or Raft [36] may be adequate, even if they can only be used practically with smaller number of peers. An advantage of using such consensus algorithms is that they can guarantee voting outcomes definitely, while Nakamoto algorithms only makes such guarantees probabilistically [37].

2.3.2 Regulation Frameworks

Distributed ledger solutions tend to exist in order to guarantee that shared logs of events are correct according to some arbitrary standards of correctness. A *regulation framework* is whatever system is used to define and enforce such standards of correctness. Each of the four solutions we compare has its own distinct framework, and additional proposals of interest do exist, such as Alex Norta's *Self-Aware Contracts* [38] and the *Definition Bank* approach we describe briefly in Papers C and D. We proceed, however, to describe only the approaches of the four solutions listed in Table 2.1, which should give a good glimpse into the range of possibilities that exists when designing these kinds of systems.

- **Bitcoin**. While designed primarily as a payment platform, Bitcoin does provide a user-programmable regulation framework. In particular, it requires that a limited Forth-like programming language, referred to as *Script* [22], be used to formulate the conditions for spending so-called *Unspent Transaction Outputs*, or UTXOs, which are what hold the currency owned by the platform's users. The language must also be used to provide the data required to fulfill those conditions when attempting to spend the funds of one or more UTXOs [39]. A basic UTXO condition would be to require the signature of a certain private key, which would lock its funds to a single user. Albeit not Turing-complete, the language can still be used to facilitate multi-signature accounts, Lightning Network payment channels [32], and more.

- **Ethereum**. In addition to a currency, *Ether*, the Ethereum platform provides the *Ethereum Virtual Machine* (EVM), as well as the possibility of creating special users that are controlled by programs defined in its Turing-complete byte-code format. These special users, referred to as *contracts* or *autonomous agents* [28], execute their programs when transactions are sent to them, which may include funds, program function arguments, as well as other things. While being able to support advanced kinds of collaboration, such as ERC20 tokens [40], *The DAO* and games like *King of the Ether Throne*, the platform has been plagued by different kinds of exploits [41] [42]. These vulnerabilities could be seen as a consequence of the wide range of possibilities the EVM provides, and the fact that contracts cannot be updated after being deployed, even if critical bugs are discovered.

- **Hyperledger Fabric**. Rather than requiring that all network maintainers execute the exact same version of each contract, as does Ethereum, Hyperledger Fabric forces each maintainer to deploy its contracts independently [43]. Consequently, different variants and versions of contracts may co-exist, which solves the problem of not being able to update deployed contracts. Concretely, Fabric contracts, or *chaincodes*, are containerized applications, interacted with via message passing [31]. Each successful chaincode invocation produces a *read-write set*, which must be acceptable according to an *endorsement policy* enforced within a group of peers, or *channel*. Such policies may require that read-write sets are signed by certain peers, as well as being acceptable according to any custom validation rules. If approved, the read-write set is eventually appended to the replicated ledger maintained within the channel. Also, while no explicit means officially exist for integration with legal institutions, there are efforts, such as the Accord project [44], aimed at facilitating such capabilities. Lastly, while being able to implement a currency system, Fabric does not natively provide an equivalent to Bitcoins or Ethers.

- **R3 Corda**. Just as Fabric, R3 Corda does not provide a native currency or embed its contracts, or *flows*, into its ledger, again implying that such flows can be updated when circumstances change or bugs are discovered. While Ethereum and Fabric model their contracts as a state machines mutated by invoking designated methods, Corda transactions apply functions to *state objects*, which may then be transformed into one or more new such objects. Each state object can only be used in a successful transaction a single time, which enables Corda to use a different kind of consensus model [33]. Concretely, transactions are normally only sent between pairs of peers, allowing each pair of peers to construct their own unique state data structures without having to reveal sensitive information to other parties. When state objects are used to change asset ownerships, however, their hashes can be presented to so-called *notary pools*, which can attest whether or not that hash has been presented to them before or not. Given that the pool of notaries can be trusted, which reach agreement through a distributed consensus algorithm, double-spending and similar forms of misconduct can be prevented, even if the notaries are never shown the actual state objects they vote about. Also, Corda allows state objects to refer to or include *legal prose*, which is any kind of suitable legal document. We are, however, unaware if such legal prose has ever been tried fruitfully in a court of law.

Whether or not a given solution should include a native currency, embed its contracts directly into its ledger, or refer to legal prose, are all examples of trade-offs that can be made to cater for certain kinds of use cases. Bitcoin and Ethereum are designed to support ad-hoc transactions or collaborations between users all over the world, even when having little knowledge of each other. Fabric and Corda, on the other hand, are created to help automate collaborations performed between parties that already have knowledge of each other and existing incentives to collaborate. Having gone through our descriptions of the just mentioned systems, it should be apparent that these assumptions motivate different design decisions, which in turn result in distinct system properties.

2.4 Characterization

What is a distributed ledger solution? It is a multi-node application that maintains a shared ledger of transactions, where each transaction represents a commitment of its author. Concretely, each ledger *transaction* modifies a *state*, which is a data structure that determines a current set of rights and obligations associated with the users of the solution. The concrete type of data structure is not important, however. It could consist of accounts and balances, key/value maps, relational database tables, and so on. A simple example of such a state data structure is given in Figure 2.1.

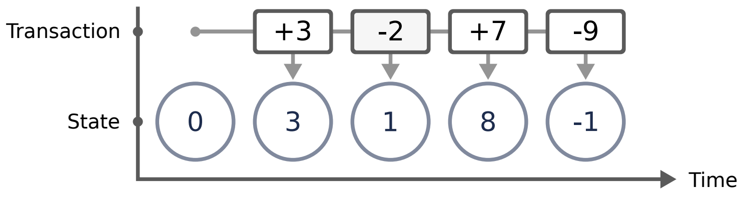


Figure 2.1: A naive distributed ledger whose state is altered by executing transactions. Each transaction carries either an addition or a subtraction, while new states are created by adding the previous state to a new transaction. The first transaction, represented by a dot, is sometimes called the genesis transaction. In this example it contains nothing.

As the state of a distributed ledger solution determines something as critical as the rights and obligations of the users of the system, it is paramount to ensure that is behaves predictably at all times. This is guaranteed via a **system-of-proof**, which relies on a **consensus mechanism**, which uses a **regulation framework**, as shown in Figure 2.2.

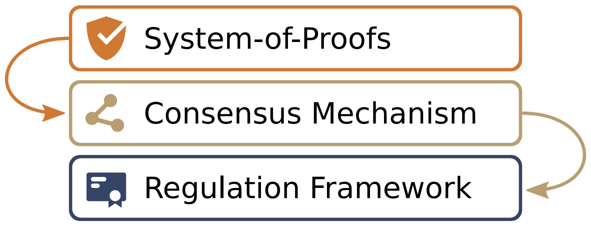


*Figure 2.2: The three abstract constituents of a distributed ledger system. A **system-of-proof** depends on a **consensus mechanism** in order to establish ledger correctness, which in turn relies on a **regulation framework** to determine how to vote during the consensus procedure.*

The *system-of-proof* guarantees that each transaction, as well as the ledger as a whole, can be trusted to be correct. The *consensus mechanism* enables the ledger maintainers to coordinate the appending of new transactions. Finally, the *regulation framework* provides the rules necessary to determine whether or not individual transactions express valid commitments.

2.4.1 System-of-Proofs

For a distributed ledger solution to be of any practical value, any parties using it most be able to rely on that it is correct. Concretely, this means that (1) each user that is able to influence the ledger can be reliably identified, (2) each transaction can be verified to have remained unaltered since its creation by a specific user, and (3) all ledger transactions follow the rules of a regulation framework, as established via a consensus mechanism.

Identity

Ensuring parties can be reliably identified is typically accomplished through the use of public key cryptography, as in all the distributed ledger systems we mentioned up until this point [1] [28] [31] [33]. Each user creates its own public/private key pair, submits the public key to a registrar together with any additional information, if at all required, and then uses its private key to prove its identity as needed [34].

Authorship

Public key cryptography can, apart from identifying users, also be used to establish that a given fact has been *signed*, or endorsed, by a specific user. A *signature* is the combination of (1) a plaintext, (2) a public key, (3) a hashing function and (4) the encrypted hash of the plaintext, where the private key corresponding to the mentioned public key was used to encrypt the hash [34]. If the plaintext is altered, any other user can notice it by, (1) hashing the plaintext using the hashing function, (2) decrypting the signature into a hash using the public key and (3) comparing the two hashes, which then will not match.

An interesting consequence of using public key signatures is that *hash pointers* can be used to refer to material beyond the signed plaintext. Such pointers are simply the hashes of other plaintexts, or any other kind of document. If those other documents are modified, their hashes will no longer match those embedded in the signed plaintext. Consequently, the signature of the original plaintext not only confirms that a certain user endorsed its contents, but also the the user endorses any contents that plaintext refers to. An example of such a signed plaintext is given in Figure 2.3.

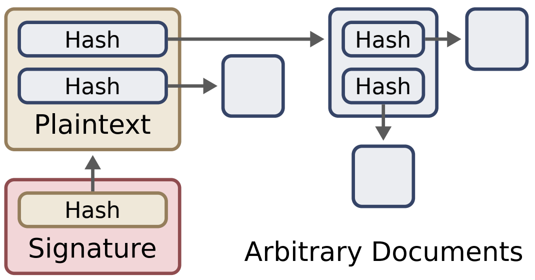


Figure 2.3: A hashed plaintext, where the hash of that plaintext is signed using the private key of an arbitrary user. As the signed plaintext contains additional hash pointers, also the documents referred to can be verified to be unaltered since the signing of the plaintext.

While a hash cannot be turned into its original plaintext, an entity given (1) a set of relevant plaintexts and (2) a signature, becomes able to determine whether or not those plaintexts are referred to by the signature. In the context of distributed ledger solutions, this can be used to reliably refer to contracts, transactions, or external data. One particularly interesting use of hash pointers is to guarantee that the *order* of some set of transactions has not been tampered with. This can be accomplished by having every transaction include the hash of its direct predecessor, as in Figure 2.4.

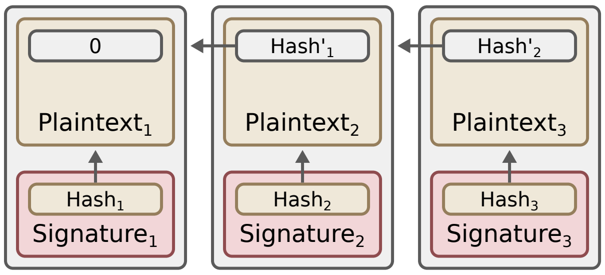


Figure 2.4: A series of signed plaintexts, each referring to a previous such via a hash pointer, except for the first. Note that each back reference is the hash of both a plaintext and a signature. A result of this schema is that each new plaintext becomes an endorsement of all previous such.

In blockchain systems, transactions are typically grouped together into *blocks*, each of which contains a hash pointer to the block directly proceeding it. This grouping into blocks is, however, primarily a means to optimize the performance. Individual transactions can be arranged in the same way. Bitcoin [1], Ethereum [28] and Hyperledger Fabric [31] are examples of systems grouping their transactions into blocks, while R3 Corda [33] is one example of a system not grouping transactions.

Validity

Ensuring that transactions refer to credible identities and can be proved to be unaltered does not in and of itself guarantee that they contain meaningful data. That is established through each ledger maintainer controlling each transaction in relation to a regulation framework, and then voting about it within a group of peers.

2.4.2 Consensus Mechanism

The consensus mechanism allows an effective majority of maintainers to decide what transactions to include in their ledger or ledgers. Using a majority of maintainers rather than a trustee spreads reliance from a single party to a group of parties, which can be an effective strategy to circumvent certain middle-men. Currently, there are many concrete consensus mechanisms in use by various distributed ledger solutions [45] [46]. In order to reduce scope, we generalize by categorizing them into three groups, which are (1) *request-response consensus*, (2) *explicit voting* and (3) *substitute voting*.

Request-Response Consensus

While perhaps not typically thought of as a consensus mechanism, the request-response messaging paradigm, often used in distributed computer systems, could be seen as a ballot of two voters. For example, some voter A formulates and votes for a proposal p, includes it in a message $m_{p,A \to B}$ and sends it to B. If B rejects $m_{p,A \to B}$, the ballot could be though of as ending in a tie, which means no action is taken. If B accepts $m_{p,A \to B}$, on the other hand, all votes were in favor of p, which is then executed.

When only two parties collaborate, this consensus model may suffice. If the parties desire to have proof of each proposal acceptance, signatures and hashes can be used as explained in Section 2.4.1. This kind of consensus is supported by R3 Corda [33], and is something we explore as parts of Papers C and D. In particular, it does not require any data to be revealed to third parties that would otherwise have to take part in the ballot, which makes its use appealing in contexts where privacy is important.

Explicit Voting

With explicit voting, we refer to systems in which votes are counted rather than being represented by some kind of *weight*, as we consider in the next section. We have already mentioned PBFT [35] and Raft [36], but other alternatives exist [45]. Some of these solutions are *Byzantine Fault Tolerant* (BFT), which means that they keep working even if a certain minority of voters actively try to manipulate the ballots, while others are referred to as being *Crash Fault Tolerant* (CFT), which means that some voters may crash, but none of them are expected to be malicious or act arbitrarily [47]. PBFT is an example of the former kind, while Raft is an example of the latter.

Explicit voting algorithms typically require that all voters are known before-hand. This makes them suitable for permissioned distributed ledger solutions, as the complete set of ledger maintainers are always known. Hyperledger Fabric is one example of a system that uses this kind of voting procedure [31]. However, as these algorithms typically require that all voters count all votes, or at least enough to be sure of an effective majority, the number of messages required to reach consensus per participant tends to increase with each new maintainer. Transaction throughput is, consequently, lowered by an increase of voters, as can be seen for the CFT algorithm Paxos in [48] and [49], which is claimed to have performance characteristics similar to Raft in [36].

Substitute Voting

The idea of substitute voting is to find a way to estimate popularity without organizing a ballot and counting votes. While it may seem a bit unintuitive, it is not difficult to find real-world analogies. For example, if wanting to know whether or not a crowd of people favor pizza or sushi for lunch, they could be asked to sceam “yes” as loud as they can as each alternative is named. Whichever alternative resulted in the loudest noise would be considered the majority winner. Another example could be to look at the tare on doors to find the most popular entrances and exists of a building. While estimates such as these may be highly inaccurate, they could be adequate for certain use cases.

The Bitcoin consensus algorithm [1] relies on an analogous kind of voting substitute, sometimes referred to as *weight*. The history of transactions with the highest weight will be considered authoritative all honest nodes. To understand how Bitcoin determines this weight, we first need to explain the three major constituents of its consensus procedure, which are (1) *transaction dissemination*, (2) *mining* and (3) *block election*.

1. **Transaction dissemination**. When any Bitcoin user wishes to transfer any of its funds, it must create a transaction and send copies of it to as many *miners* as reasonably possible. A miner is a user that actively participates in the creation of new blocks, which are batches of transactions that are accepted or reject collectively. Each such block, significantly, refers to an earlier block of transaction via a hash, which means that the blocks form chains of blocks, or *blockchains*.

2. **Mining**. Each miner collects and validates incoming transactions while also trying to produce a *proof-of-work*. A proof-of-work is the solution to a cryptographic puzzle that can only be solved by trying random solutions. The puzzle is always formulated in terms of the hash of the most recent authoritative block. The first Bitcoin node to find the next puzzle solution becomes the legitimate creator of the next block and can claim a reward in the Bitcoin currency.

3. **Block election**. A miner ends its attempt to produce a proof-of-work when it either (A) finds a solution, or (B) gets a block with a solution from another miner. In the case of the former, the miner assembles its own block from (1) the transactions it collected, (2) the proof-of-work it found and (3) the hash of block preceding the created one, and then spreads the block to other nodes. It then continues by trying to find a proof-of-work for whatever block is now the latest authoritative such.

   A consequence of this procedure is that individual nodes may receive blocks created by different miners in isolation. The occurrence is referred to as a *fork*, as the result is the existence of two or more valid *branches* of blocks, as depicted in Figure 2.5. The consensus protocol stipulates that the block with the highest weight is selected, where weight is calculated from the number of transactions, proofs-of-works, and other details, included in each block *and all previous blocks it refers to.*

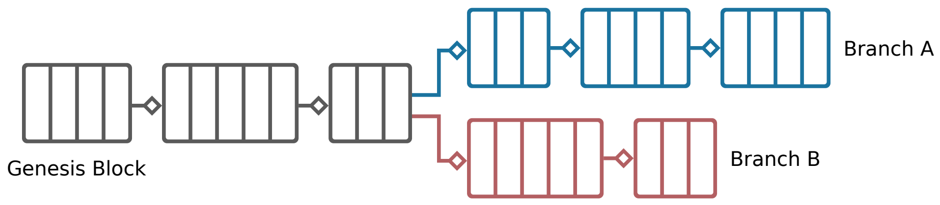


Figure 2.5: Chained blocks of transactions, with a fork after block 3. According to the Nakamoto consensus protocol [1], miners must only build on the chain with the highest known weight. If assumed that branch A is heavier, then the two blocks only part of branch B must be ignored.

How does this procedure produce a substitute for voting? In effect, each miner votes on the chain of blocks it believes in by trying to find a proof-of-work for its next block. In other words, the act of finding a proof-of-work *is* the voting, while the weight observable in a chain of blocks is proof of how the votes have ended in the past. There are several alternatives to both proof-of-work as weight and the use of Nakamoto consensus when larger numbers of users need to reach consensus, some of which are described in [46] [50].

However, this kind of voting is not without its drawbacks. We have already implied that it may seem like a transaction has been accepted by the ledger maintainers for a while, only for the maintainers to forget it by changing to a different branch of blocks. Even if the probability of such a change of branch to happen decreases over time, the possibility remains real [37]. Another problem arises when individual miners, or pools of miners, become able to try significantly more proof-of-work solutions per second than the rest of the maintainers, as considered in [51].

2.4.3 Regulation Framework

The last primary constituent, and what could be considered the foundational component of a distributed ledger solution, is the *regulation framework*. Its primary function is to dictate what may and what may not be recorded in the ledger maintained by the system in question. This to ensure that the ledger provides a meaningful description of the rights and obligations of its maintainers. More specifically, such a ledger consists of *transactions* that may be *executed* in sequence to produce a *state data structure*.

Transactions

We started our characterization of distributed ledger solutions by claiming that their transactions represent commitments. While there may be many ways of modeling such commitments, we do so here by assuming that transactions are function invocations.[3] For example, a transaction may express that A wants to transfer x amount of currency to B. In this case, the commitment is to provide B with currency. Another example could be that A wants to give up ownership of item y in exchange for item z owned by B. In this case, two authors commit to exchanging items with each other. To ensure that transactions are legitimate commitments of their authors, digital signatures are typically used. In the first example, where A transfers money to B, the signature of A would likely suffice. In the second example, however, where two users exchange ownerships, both signatures would likely be required to facilitate the same kind of confidence.

Concretely, a transaction could be said to consist of (1) a set of signatures, (2) the name of an invoked function and (3) any arguments, apart from any other details required by consensus algorithms, for time tracking, and so on. In the case of our first example, the function name could be "`transferTo`" and the arguments could be x and B. In the second example, the function could be "`exchange`" and the arguments y and z.

[3]In Papers C and D, we instead refer to transactions as *finalized exchanges* and consider them to always cause so-called *tokens* to change owners. Even though the function-and-arguments abstraction is often used by distributed ledger solutions, such as [28] [30] [33], its use is by no means mandatory.

Transaction Execution

When a transaction is executed, the function named in the transaction is looked up and provided with the signatures and arguments in the transaction. If the function invocation fails, which it only does if the commitment the transaction describes is unsound, the transaction is considered invalid and should not be part of any ledger maintained by the solution. If the invocation is successful, however, the transaction will eventually cause a state data structure to be updated. The exact details regarding when transactions are executed and when their results cause a state change depends on the implementation of the distributed ledger solution in question. It also varies whether or not the users of the solution are allowed to define their own functions, and what those functions then are capable of. While some form of programmability seems to be the norm, via smart contracts or otherwise, Cryptonite [52], for example, provides only a fixed set of functions.

State Data Structures

When a function invocation is successful, it causes a state data structure to be updated. The data structure could be thought of as a means to determine the current set of *rights and obligations* of all users of a distributed ledger solution. A *right* could, for example, be ownership of and amount of currency and the possibility of transferring it to other users. An *obligation* could, on the other hand, be a commitment to drive any paying user between certain locations. The concrete form of the state data structure depends on what is considered the most straightforward way for to fulfill its purpose. It could, for example, be a set of accounts and balances or key/value stores.

2.5 Summary

The distributed ledger technologies we see today can trace their roots back to efforts such as Nick Szabo's smart contracts [15] and the early work of the cypherpunk movement [21]. These efforts eventually culminated with the introduction of Bitcoin in 2008 [1], which soon became the subject of a global hype [8]. Bitcoin's raise to popularity proved the viability of the technology used to construct it, which in turn created significant interest in adapting the technology for new kinds of use cases. As a result, two categories of distributed ledger systems arose, the *permissionless*, with examples such as Bitcoin and Ethereum [28], and the *permissioned*, which includes examples such as Hyperledger Fabric [31] and R3 Corda [33]. In the former kind of system, variants of the Nakamoto consensus algorithm [1] dominate due to them facilitating larger number of maintainers and not requiring any one node to have complete knowledge of the users of the system. In the latter kind of system, traditional distributed consensus algorithms, such as PBFT [35], are used as they provide stronger consensus guarantees and provide higher throughput [26] [48] [49]. Irrespective of category, a distributed ledger solution always consists of (1) a *system-of-proof*, (2) a *consensus mechanism* and (3) a *regulation framework*, allowing it to (1) prove the correctness of its ledger, (2) coordinate the creation of new ledger entries and (3) validate ledger entries and map them to user rights and obligations.

Chapter 3

The Road to Machine Economies

If reliable distributed ledger solutions can be constructed that overcome all current performance, interoperability and integration issues, what concrete types of applications would that facilitate? What would then become the most relevant research topics, and how would their results shape cultures, economies and power structures? These questions are all relevant to ask, and there is no shortage of speculation about their answers [8] [9] [53] [54] [55] [56]. However, each such speculation is formulated from a unique perspective, which means that they both compete and enrich each other. We believe serious investigators do well to consult many kinds of predictions and perspectives to improve their chances of making relevant preparations for future trends and disruptions.

3.1 Overview

In this section, we afford ourselves to speculate freely about the future of our research contributions and wider research area. We begin by considering the three main topics of the thesis, which are the (1) performance, (2) interoperability and (3) integration of distributed ledger technologies. We then end by extrapolating the trends we observe and try to make predictions about how these will shape research and societies in the decades to come. In summary, we predict that the vast majority of concrete financial and contractual decisions will be made autonomously by machines, but that those machines will act according to higher-level policies formulated by humans.

We recognize that it is hard to be accurate when making these kinds of predictions, as individual details may alter the trajectories of, what could otherwise be thought of, straightforward chains of events. However, these kinds of predictions can trigger fruitful discussions about how downsides of the projected developments can be mitigated, help motivate research funding, as well as enable people to make other kinds of preparations for upcoming shifts in cultural, economic or political structures. It should be had in mind, however, that there is a significant risk that our predictions will turn out to be incorrect. If serious about wanting to understand future developments, many more sources and experts should be consulted.

3.2 Performance

Permissionless distributed ledger solutions, such as Bitcoin or Ethereum, tend to have significant limits to throughput and latency [25] [26]. For example, Bitcoin and Ethereum can only handle up to 7 and 20 transactions per second, respectively. This in contrast to Visa, an electronic funds transfer provider, which, according to [26], is able to handle up to about 10 000 transactions per second. Furthermore, a Bitcoin transaction requires about 10 minutes or more to be processed [25], while we believe most people using Visa cards are used to it taking no more than a few seconds. At the other end of the spectrum, permissioned solutions have a network size problem [26]. The kind of consensus algorithms these systems currently tend to use cannot realistically scale to hundreds of maintainers, which means that they are confined to use cases where only a limited number of maintainers are required. Additionally, when the rate of created transactions is high, both of these categories of systems have a storage capacity problem.

What would happen if these problems ceased to be relevant? We could, for example, assume that higher network bandwidths, faster chips, new kinds of algorithms, or denser storage solutions become available. Another contributor could be improvements to the ledger pruning algorithm we present in Paper A, which would allow storage space to be used more efficiently. However these problems are mitigated, we believe the result would be that the technology could become ubiquitous. Everything from smart bolts to toilet paper dispensers could participate in distributed ledger economies. If we assume that the solution we imagine would be straightforward to use for most people, it could become feasible to charge very small fees for activities that today often have to be funded in other ways, like visiting websites, listening to music, using public restrooms, etc. It could also mean that it becomes viable to have Internet-connected devices buy resources from each other, such as gas or electricity, as a way to minimize resource waste.

What industries would be disrupted by such developments? Most significantly, we believe it would be the financial industry [57]. If money can be transferred cheaply without the assistance of a bank, those banks would loose an important source of revenue. Also, if legal tender can be transferred without a central bank, or if central banks adopt distributed ledger technologies, banks would likely need to adapt to major regulatory changes. However, we do not believe the need for the other services banks provide is going to disappear, such as loans, liquidity forecastings, financial instruments, and so on. Other potential disruptions, such as new forms of insurance, would likely need more than only improvements to distributed ledger performance.

The advertising industry could be another subject of such disruption. Cheap and easy payments could popularize alternatives to watching advertisements, which would reduce the influence of Facebook [58], Google [59] and other companies primarily living off selling advertising space, or at least force them to extend their revenue models. Other industries, like the transportation, manufacturing and e-commerce industries, would likely benefit from transitioning to distributed ledger payment systems, in terms of lower costs and more room for digitization. However, if the technology cannot be integrated more effectively than today, we do not expect these industries to face any related disruptions.

3.3 Interoperability

As we already covered in Section 1.2.1, one major barrier to the adoption of distributed ledger technologies is the effort required to integrate them with existing systems. If a means could be identified that would allow seamless interoperation between legacy systems and distributed ledger solutions, perhaps in the way we propose in Section 4.3, it could significantly lower the time and costs associated with the adoption and use of the technology. This would in turn lead to lower costs of testing, deploying and upgrading distributed ledger solutions, which would speed up the rate of innovation and lower the barriers to using the technology fruitfully. While perhaps not a direct vehicle of disruption in and of itself, access to low-cost interoperation may lead to disruption solely by making more kinds of companies able to use the technology to distinguish themselves from their competitors. It could also make it more feasible for home users to use distributed ledger solutions, helping them buy and sell surplus solar power, share interesting snippets captured by home surveillance cameras, and so on.

3.4 Integration

While advances in the performance and interoperability of distributed ledger technologies may be required to make them economically feasible, advances in integration are required to make them practically useful in more settings. As far as we can tell, the only type of application of distributed ledger technologies that has been commercially successful is the management of currencies, whether it be Bitcoins [1] or ERC20 tokens [40]. Other projects, such as the DAO, an organization where financial decisions must be approved via a vote, have failed more or less spectacularly [41]. Why is this the case? We believe it comes down to two major factors, (1) immaturity and (2) isolation. With the former, we mean that problems such as managing bugs, or knowing what kinds of collaborations Etherum is suitable for, do not have well-understood solutions. With the latter, we mean that distributed ledger technologies are hard to integrate with the technologies and business practices prevalent in most markets this day. In particular, integration with legal institutions is one important kind of integration that is still missing [60].

What would happen if these problems ceased to be relevant? We could assume that legal frameworks are developed that make most national courts of law accept certain distributed ledger technologies and consider the data they generate as proofs. Perhaps would some of those systems be based on our Exchange Network architecture, which we present in Papers C and D. Initially, we believe the technology would be regarded primarily as a way to digitize existing structures between collaborating companies, such as between manufacturers and suppliers, rather than as a tool for innovation. Paper contracts would be replaced with digital such, with little to no variations between the original contracts and their digital counter-parts. As computers are faster than humans, however, early innovators could try to drastically decrease the scope of agreements and increase the frequencies at which they are made, which could increase process granularity and efficiency.

In the longer term, we believe the most significant impacts of improved integration will be that it (1) lowers the costs of collaboration and (2) facilitates unprecedented degrees of micromanagement. Firstly, as collaborations takes less time to set up, operate, follow up and terminate, corporations are given larger freedom to change collaborators when their incentives change. This may lower the barriers to market entry, which in turn could leave more room for competition. Secondly, when contracts no longer are written primarily for humans to follow, but computers, they can become staggeringly complex by today's standards. This has implications not just for agreements between companies, but also on what kinds of rules law-makers can expect organizations to follow. For example, rather than having a couple of criteria for calculating income taxes, hundreds of criteria could be used without it increasing costs significantly for the companies that would have to follow them, if they have computers that can manage the calculations for them.

What kinds of disruption could such developments be expected to cause? More than any other, we believe that legal institutions and enterprises would be disrupted. When legal institutions work out new best-practices, laws and standards, the rest of the legal system will have to follow, even if the old practices remain accepted. For example, a litigation where all proofs are digital could become significantly less expensive than such involving traditional paper proofs, as computers would be able to assist more than previously possible in verifying documents, receipts and legal circumstances. This means that new competences will be in demand, which may lead to new kinds of legal firms, lawyers and other experts outcompeting the contemporary ones. Another, albeit related, form of disruption may occur within business-to-business markets. If a company is too slow at adopting the new technologies, it may become less in demand as it cannot offer the same kinds of contractual flexibility, granularity and frequency as its competitors.

3.5 Further Down the Road

Given that all performance, interoperability and integration issues are solved, what will then be the most pressing research questions? The three categories of issues we mentioned are all about *infrastructure*, which is another way of saying that they are about taking cooperational processes from the physical domain into the digital. When those processes are fully and adequately digital, the next step is to make them autonomous. Research from the Artificial Intelligence community, like that on multi-agent systems summarized in [61], could likely be improved and adapted for this particular kind of use cases. The result would be that computers become able to manage more and more aspects of our economies, according to higher-level policies formulated by humans. As they can do so faster and more accurately than humans possible could, we expect that the vast majority of financial interactions eventually will be executed by computers. Trying to predict how such machine-dominated economies would affect society is hard, however, as these would develop alongside other technological innovations, cultural trends, and political changes. Perhaps we will see that most individuals use computer assistants to budget and manage their economies, or it becoming ubiquitous to own household robots that also produce, buy and sell goods and services with neighbors? The future will tell.

Chapter 4
Contributions

The purpose of this thesis is twofold, (1) to provide a tangible artifact that can be used by the scientific community to assess the academic competence of myself, Emanuel Palm, and (2) to help interested readers gain a deeper understanding about the contexts and observations that lead my supervisors I to write the papers included in the second part of the thesis. By implication, the primary contributions we, my supervisors and I, make to the scientific body of knowledge about distributed ledger technologies it not this thesis itself, but the four papers it contains.

4.1 Overview

In this section, we go through each of the four papers included in Part II, in the order they are presented there. The purpose of this review is to provide motivations, summaries and to help clarify how I, Emanuel Palm, contributed to each of those papers. The papers correlate with the perspectives and research questions introduced in Chapter 1 as depicted in Figure 4.1.

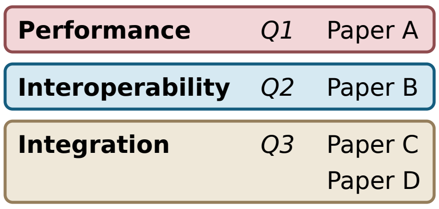

Performance *Q1* Paper A
Interoperability *Q2* Paper B
Integration *Q3* Paper C
Paper D

Figure 4.1: An overview of how the four papers reviewed in this section relate to the research perspectives and questions in Section 1.2.

At the time of writing, Papers A and B have been accepted and presented at academic conferences, Paper C have been accepted but not yet presented at such a conference, while Paper D has been submitted to a journal, but has not yet been accepted or rejected. All of Papers A, B and C either have been or are to be published in conference journals.

4.2 Paper A

Title: Selective Blockchain Transaction Pruning and State Derivability

Authors: Emanuel Palm, Olov Schelén, Ulf Bodin

Published in: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT)

Motivation: The paper is a rewrite of my Master's thesis [10]. The rewrite was regarded as a good way for me to get a first paper published and to deepen my understanding of distributed ledger technologies.

Summary: We introduce a method for blockchain maintainers to independently select and remove blockchain transactions, while still being able to preserve the derivability of certain ledger states. An implementation of the approach, which we also demonstrate, is able to reduce the size of its blockchain by about 84.49%.

Author contributions: I wrote all parts of the paper, but received significant feedback from Olov Schelén and Ulf Bodin. Also Jerker Delsing helped with suggestions regarding its presentation. The extension of Hyperledger Fabric and benchmarking mentioned in the paper were originally developed and gathered for my Master's thesis [10]. However, the paper adds to the original thesis by introducing more formalisms and a new discussion.

4.3 Paper B

Title: Syntactic Translation of Message Payloads Between At Least Partially Equivalent Encodings

Authors: Emanuel Palm, Cristina Paniagua, Ulf Bodin, Olov Schelén

Presented at: 2019 IEEE International Conference on Industrial Technology (ICIT)

Motivation: I had made a distributed configuration system as part of a course on the Arrowhead Framework [7] with translation capabilities, and it was deemed to be interesting enough to be reworked into a paper. As the kind of translation has direct relevance to the work of Derhamy et al. [6], it is presented as a complimentary extension of his work. In particular, Derhamy et al. provide a system for message *protocol* translation, while we introduce a system for message *payload* translation.

Summary: We present a theoretical method for translating message payloads in transit between endpoints. The method involves representing and analyzing payload syntaxes with the aim of identifying concrete translations that can be performed without risk of syntactic data loss. While only able to translate payloads that are *syntactically equivalent*, as defined in the paper, we see it as a first step towards *syntactic transformational translations* and *semantic translations*.

Author contributions: I am the primary author of the second and third sections, while Cristina Paniagua is the primary author of the first and the last. Even though the paper covers what was my own original ideas, Cristina Paniagua, Ulf Bodin and Olov Schelén all provided indispensable feedback, especially in terms of how to present the work, understanding its utility and refining its formalisms.

4.4 Paper C

Title: The Exchange Network: An Architecture for the Negotiation of Non-Repudiable Token Exchanges

Authors: Emanuel Palm, Olov Schelén, Ulf Bodin, Richard Hedman

Accepted to: 2019 IEEE International Conference on Industrial Informatics (INDIN)

Motivation: We were tasked with digitizing a supply chain use case as part of the Productive 4.0 project, and we concluded that its requirements would be best satisfied by a negotiation framework. However, no such framework we could find was able provide guarantees analogous to those of distributed ledger solutions, for which reason we decided to produce our own architecture and implementation.

Summary: We present the *Exchange Network*, a general-purpose and implementation-independent architecture for digital negotiation and non-repudiable exchanges of *tokens*, which are symbolic representations of arbitrary values. We consider the implications of implementing the architecture in three different ways, as well as demonstrating the feasibility of the architecture by outlining our own implementation of it and also by describing a supply-chain scenario inspired by one transportation process at Volvo Trucks.

Author contributions: I made the evaluated implementation and wrote Sections I, II, III, IV and VI, while Ulf Bodin wrote Section V together with Richard Hedman and Olov Schelen wrote Section VII. The ideas in the paper have been discussed and reevaluated so many times that it is hard to know what ideas to attribute to whom. Generally, the basic ideas are my own, while many improvements and insights came via the other authors.

4.5 Paper D

Title: Approaching Non-Disruptive Distributed Ledger Technologies

Authors: Emanuel Palm, Olov Schelén, Ulf Bodin

Submitted to: IEEE Access (Open Call)

Motivation: As we believe the particular implementation of the *Exchange Network* architecture we present in Paper C provides a compelling design pattern for digitizing business-to-business interactions, we decided to write a more extensive paper about it and how it is less disruptive than the state-of-the-art.

Summary: We propose a design for distributed ledger solutions based on the *Exchange Network* architecture that lacks *distributed consensus algorithms* and *code-as-contracts*, both of which tend to make such solutions break with prevailing legal and business practices. Concretely, we characterize the current cooperational paradigm, outline six requirements of adherence, present our design, as well as considering both how our own solution and how R3 Corda could fulfill our requirements.

Author contributions: I wrote all parts of the paper, but received significant feedback from both Olov Schelén and Ulf Bodin. In particular, they gave indispensable comments regarding the scope, aim and presentation of the paper.

4.6 Other Work

During the period the four papers presented in this chapter were written, I was engaged as a teacher's assistant in three courses, implemented one and wrote documentation for two experimental Arrowhead Framework [7] systems, completed five graduate courses, presented on project and employee workshops, and went on many meetings. Several collaborations on papers were planned, but none of them ended up coming to fruition before this thesis was submitted.

Being a doctoral student is a work of many assignments, as well as a struggle to stay up to date with relevant research subjects and coming up with new insights and solutions. Many of the skills I have acquired did not leave any tangible artifacts. However, my observation is that I have become a more proficient writer and presenter, as well as having learned much about finding and assessing the quality of relevant research.

Chapter 5

Reflections

At this point, we have briefly described the state-of-the-art of distributed ledger technologies, outlined our expectations on the future development on and impacts of the technology, as well as how we ourselves have contributed to the advancement of the art.

5.1 Overview

Here, we present our answers to the research questions listed in Section 1.2.2, our views on developments in our research area from an ethical perspective, as well as a description of how we will continue to work with the technology in the future.

5.2 Conclusions

Q1 *What possibilities exist for making distributed ledgers require less disk space, even if assuming new data will be added indefinitely?*

The primary strategy we investigate in Paper A is *ledger transaction pruning*. The strategy becomes possible when transactions only need to be executed a limited and well-known number of times, after which they can be removed. When this strategy can be applied depends on whether the distributed ledger solution of concern relies on either (1) a *deterministic*, or (2) a *probabilistic* consensus algorithm. While the former kind of algorithm allows for executed transactions to be removed immediately, the latter one requires waiting until the probability of having to unwind such executed transactions is sufficiently low [37]. As the kind of algorithm can never be absolutely sure about whether another, more probable, history of transactions will be presented, room must be left for a sufficiently large number of its transactions, from the most recent and backwards, to be replaced. Such replacements, or *resets*, require that replaced transactions have their executions reversed before the new ones can be executed.

In the case of it being relevant to be able to determine if a given transaction belongs to a set of pruned, but authoritative, transactions, chains of hashes can be kept for the purpose. However, if those hashes are kept indefinitely, pruning merely reduces the rate of growth rather than limiting it.

Apart from pruning historic transactions deemed of less relevance, we believe there could be plenty of room for compressing transactions using tailor-made compression algorithms, which could substitute public keys, common scripts, as well as other data, with less expansive placeholders. The strategy will never, however, be able to completely limit ledger growth, as that will always require transactions to be fully removed.

Q2 *What would be required to facilitate seamless integration between legacy systems and distributed ledger technologies?*

In Paper B, we propose a translation strategy where syntactically equivalent message payloads are converted at the syntax level. The strategy requires (A) that specifications be written for each syntax and (B) that a mapping between equivalent structures in each syntax is created. However, this kind of translation is not seamless, even if combined with the protocol translations capabilities we mention in our paper. Firstly, non-equivalent payloads can not be translated and, secondly, translated payloads are not adapted to the semantic expectations of their receivers. To mitigate this, we propose two extensions to our model, a *transformational* and a *semantic*.

1. **Transformational Translation**. To be able to translate between *any* syntax trees, not just equivalent such, a consistent and reversible syntax transformation schema needs to be applied during translation. As a first step, such transformation schemes could be defined via specifications written by humans.

2. **Semantic Translation**. In order for translation of messages between services not designed to work together, the *semantic* gap between their message formats need to be bridged, not the syntactic one. By this we mean that conversions between, for example, CBOR and JSON will not be enough. Also the fields of the messages will need to be adapted, rearranged or complemented. This requires a robust way of reasoning about and describing the semantic expectations of each service of concern. We believe that two such semantic descriptions, taken from whatever services are to communicate with each other, could be used to generate a syntax transformation schema. However, services include and require slightly different sets of data fields, which may lead to syntax transformation schemes ending up being incomplete. Completing them could require that data is queried from multiple services, or that valid assumptions can be made about default values.

We believe the journey we started with Paper B has the potential to lead to truly seamless translation in the future, but much work remains. The approach we propose may end up requiring many kinds of specifications to be written in order to translate properly, which may not seem very seamless. There may, however, exist ways to infer many of those specifications, perhaps by parsing source code or analyzing messages.

Q3 *How can distributed ledger systems be fitted into the cooperational structures already existing between industrial stakeholders?*

Firstly, the collaborative needs of those industrial stakeholders must be clearly understood. Secondly, those needs must be compared to the capabilities and limits of available distributed ledger technologies. Thirdly, a solution must be identified that meets as many as possible of those needs, and that is able to integrate with other systems and processes that can meet the needs that cannot be directly addressed.

1. **The collaborative needs of the industry**. As part of Paper D, we outline a characterization of contractual cooperation and turn it into a list of six relevant requirements. Those requirements are being able to (1) prove what has been agreed upon, (2) renegotiate terms when circumstances change, (3) handle deviations in a court of law or via some other adjudicator, (4) interpret terms consistently, (5) identify other parties reliably, as well as (6) be able to conceal agreements and interactions from competitors.

2. **Relevant capabilities and limits of distributed ledger technologies**. As we consider in more detail in Chapter 2, a distributed ledger solution is a system where two or more parties maintain and replicate a ledger of transactions, where each added transaction is verified to be correct by a regulation framework. When all the transactions are executed, they produce a state data structure from which the rights and obligations of the system users can be determined. How the capabilities of these solutions tend to match the cooperational requirements we listed above is considered at great length in Paper D. In particular, we know of no solution that directly facilitates contract renegotiation or provides a proven system of integration with courts of law or other adjudicators. While some distributed ledger solutions are able to substitute trusted third parties with automated voting procedures, such substitutes are typically very constrained in terms of what they can consider as evidence, as well as the power they have to compel misbehaving users to make reparations. Also, only some solutions allow for contracts and transactions to be revealed exclusively to those of immediate concern.

3. **An adequately useful solution**. In both of Papers C and D, we outline an architecture and implementation strategies that can be used to build industrial collaboration systems. In particular, we propose that many common scenarios are most adequately served by a consensus procedure akin to that of R3 Corda [33], which provides good opportunity for preserving privacy, as well as by replacing the executable-code-as-contracts model, which is used by virtually all distributed ledger solutions, with a model in which digital ownerships are negotiated about and exchanged. We argue that a system about defining and exchanging *tokens*, with their ownership and transfer implications defined in conventional legal language, provides a lower barrier to adoption by legal institutions and experts than does systems about programmatic specifications and finite state machine transitions.

5.3 Ethical Considerations

No research is without its implications on wider society, and that especially if it affects resource production and distribution, trading and finance, or other areas of geopolitical significance. When talking about distributed ledger technologies, there are two themes we believe especially relevant to discuss from an ethical perspective, which are (1) inflated expectations on the technology and (2) its potential to disrupt societal power structures. We discuss them here in turn.

5.3.1 Inflated Expectations

For anyone who has followed the discussion around and evolution of blockchains during the last years, it will unlikely come as a surprise that the technology shows signs of being subject of a hype. In other words, the expectations on the technology [9] do not match the capabilities it has proven to have [62]. While large interest in a technology creates opportunities for it to be further developed, via funding of scientific projects and other initiatives, it is our observation that it also inflates the expectations of those paying for, leading, providing feedback and following up on those initiatives.

Perhaps contrary to intuition, this can create an incentive for those tasked with developing the technology to focus their work on non-essential improvements. If the researchers or developers insist that the technology is far less capable than advertised, they compromise the prestige and agendas of those investing in it, as well as risking conflict with others better served by a status quo. If, on the other hand, the researchers or developers can show that they have been able to improve some aspect of the technology, even if it has little real-world consequence, they add to the prestige and momentum of their funders. At the same time, however, they also perpetuate the idea of the inflated expectations being valid.

Even though this may come as no surprise to those with experience with financing and leading research projects, we would like to emphasize the importance of creating and upholding cultures where projects are allowed to fail and research outcomes may be very different from what was initially expected.

5.3.2 Disruption of Power Structures

In Chapter 3, we considered how distributed ledger technologies, given the right kinds of improvements, could disrupt both financial and legal institutions. This is a historically interesting kind of disruption, as both of the financial and legal functions are integral to how modern governments control their economies. We suspect that if governments cannot effectively make the transition to the new distributed ledger paradigm, when and if it does usher in, they risk being out-competed by private actors providing the same kinds of services. We make no judgement regarding whether such a development would be good or bad, but note that it could lead significant societal turmoil if governments neither expect or welcome such developments.

5.4 Future Work

While some potential ideas for future efforts have already been presented in this chapter, we here reiterate a few of them, as well as presenting new ones. We categorize the ideas according to the three perspectives we first presented in Chapter 1.

1. **Performance**. While we have presented a method for pruning transactions to reduce disk space, we see potential for other methods for further improvements. For example, in cases where it is relevant to be able to determine whether given transactions were part of a ledger, but has been pruned, Merkle trees may be used as suggested in [1]. However, in cases were some degree of inaccuracy is acceptable, there could be room for instead using something like bloom filters [63], which would further reduce disk requirements. Other promising research topics could include extending our selective transaction pruning approach to more kinds of ledgers, handle more transaction selection algorithms, or operate more efficiently in terms of reduced execution time, compute power, electricity or hard drive tare.

2. **Interoperability**. In Section 5.2.Q2, we describe two extensions to the translation model we present in Paper B, one transformational and one semantic. As those extensions require certain specifications to be written, it could become interesting to investigate whether those specifications could be generated automatically. Another relevant line of research could be to determine if code could be generated from our specifications that could be deployed directly on communicating devices, which would remove the need for an intercepting translation service.

3. **Integration**. In Papers C and D, we describe the four different components of our *Exchange Network* architecture. We believe there are several things that could be done to make the architecture automate more aspects of managing contractual relationships. For example, tracking, analyzing and sharing data about what other parties can be trusted, perhaps as suggested in [64] [65], automatically sharing contractual definitions and other data as required by accepted agreements, buying and selling private data, methods for using conventional contracts in Exchange Networks, or higher-level negotiation, such as about how to negotiate, about the details of a contract, or about contractual exceptions.

While there are plenty of subjects that could be investigated in the future, what we will pursue in the future will largely depend on the wants and needs of whatever project will fund our research. It is not inconceivable that a line of questions we haven't considered until this point will be the primary subject of our coming efforts.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, accessed 2019-02-08. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[2] J. Al-Jaroodi and N. Mohamed, "Industrial applications of blockchain," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan 2019, pp. 0550–0555. [Online]. Available: https://doi.org/10.1109/CCWC.2019.8666530

[3] L. D. Xu, E. L. Xu *et al.*, "Industry 4.0: state of the art and future trends," *International Journal of Production Research*, vol. 56, no. 8, 2018. [Online]. Available: https://doi.org/10.1080/00207543.2018.1444806

[4] Z. Zheng, S. Xie *et al.*, "Blockchain challenges and opportunities: a survey," *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, 2018. [Online]. Available: https://doi.org/10.1504/IJWGS.2018.095647

[5] K. Efthymiou, D. Mourtzis, A. Pagoropoulos, N. Papakostas, and G. Chryssolouris, "Manufacturing systems complexity analysis methods review," *International Journal of Computer Integrated Manufacturing*, vol. 29, no. 9, pp. 1025–1044, 2016. [Online]. Available: https://doi.org/10.1080/0951192X.2015.1130245

[6] H. Derhamy, J. Eliasson, and J. Delsing, "IoT interoperability – on-demand and low latency transparent multiprotocol translator," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1754–1763, Oct 2017. [Online]. Available: https://doi.org/10.1109/JIOT.2017.2697718

[7] J. Delsing, *IoT Automation: Arrowhead Framework.* CRC Press, 2017.

[8] M. E. Peck, "Blockchains: How they work and why they'll change the world," *IEEE Spectrum*, vol. 54, no. 10, pp. 26–35, 2017.

[9] M. Swan, *Blockchain: Blueprint for a New Economy.* O'Reilly, 2015.

[10] E. Palm, "Implications and impact of blockchain transaction pruning," 2017. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-64986

[11] National Research Council, *Academic Careers for Experimental Computer Scientists and Engineers.* Washington, DC: The National Academies Press, 1994. [Online]. Available: https://doi.org/10.17226/2236

[12] A. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *Management Information Systems Quarterly*, vol. 28, pp. 75–105, Mars 2004. [Online]. Available: https://doi.org/10.1007/s11576-006-0028-8

[13] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, p. 131, Dec 2008. [Online]. Available: https://doi.org/10.1007/s10664-008-9102-8

[14] N. Szabo. (1994) Smart contracts. Accessed 2019-05-20. [Online]. Available: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html

[15] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997. [Online]. Available: https://doi.org/10.5210/fm.v2i9.548

[16] I. Grigg, "Financial cryptography in 7 layers," in *Financial Cryptography*, Y. Frankel, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 332–348. [Online]. Available: https://doi.org/10.1007/3-540-45472-1_23

[17] I. Grigg, "The ricardian contract," in *IEEE International Workshop on Electronic Contracting*, July 2004, pp. 25–31. [Online]. Available: https://doi.org/10.1109/WEC.2004.1319505

[18] L. L. Leff, "Automated reasoning with legal XML documents," in *Proceedings of the 8th International Conference on Artificial Intelligence and Law*, ser. ICAIL '01. New York, NY, USA: ACM, 2001, pp. 215–216. [Online]. Available: https://doi.org/10.1145/383535.383560

[19] G. Governatori, "Representing business contracts in RuleML," *International Journal of Cooperative Information Systems*, vol. 14, no. 02n03, 2005. [Online]. Available: https://doi.org/10.1142/S0218843005001092

[20] M. E. Peck, "Bitcoin: The cryptoanarchists' answer to cash," *IEEE Spectrum*, vol. 49, no. 6, pp. 50–56, 2012.

[21] A. Narayanan and J. Clark, "Bitcoin's academic pedigree," *Communications of the ACM*, vol. 60, no. 12, pp. 36–45, 2017. [Online]. Available: https://doi.org/10.1145/3132259

[22] R. Klomp and A. Bracciali, "On symbolic verification of bitcoin's script language," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology.* Springer International, 2018, pp. 38–56. [Online]. Available: https://doi.org/10.1007/978-3-030-00305-0_3

[23] P. Reynolds and A. S. M. Irwin, "Tracking digital footprints: anonymity within the bitcoin system," *Journal of Money Laundering Control*, vol. 20, no. 2, pp. 172–189, 2017. [Online]. Available: https://doi.org/10.1108/JMLC-07-2016-0027

[24] K. J. O'Dwyer and D. Malone, "Bitcoin mining and its energy footprint," 2014. [Online]. Available: https://doi.org/10.1049/cp.2014.0699

[25] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer, "On scaling decentralized blockchains," in *Financial Cryptography and Data Security*, J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 106–125. [Online]. Available: https://doi.org/10.1007/978-3-662-53357-4_8

[26] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *International workshop on open problems in network security*. Springer, 2015, pp. 112–125. [Online]. Available: https://doi.org/10.1007/978-3-319-39028-4_9

[27] A. Chauhan, O. P. Malviya, M. Verma, and T. S. Mor, "Blockchain and scalability," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, July 2018, pp. 122–128. [Online]. Available: https://doi.org/10.1109/QRS-C.2018.00034

[28] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014, accessed 2019-05-20. [Online]. Available: https://gavwood.com/paper.pdf

[29] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*, May 2014, pp. 459–474. [Online]. Available: https://doi.org/10.1109/SP.2014.36

[30] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016, accessed 2019-05-20. [Online]. Available: https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf

[31] E. Androulaki, A. Barger *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, pp. 30:1–30:15. [Online]. Available: https://doi.org/10.1145/3190508.3190538

[32] J. Poon and T. Dryja. (2016) The bitcoin lightning network: Scalable off-chain instant payments. Version 0.5.9.2. Accessed 2019-05-20. [Online]. Available: https://lightning.network/lightning-network-paper.pdf

[33] M. Hearn. (2016) Corda: A distributed ledger. Accessed 2018-05-21. [Online]. Available: https://docs.corda.net/_static/corda-technical-whitepaper.pdf

[34] A. Salomaa, *Public-key cryptography*, 2nd ed., ser. Texts in Theoretical Computer Science. Springer Science & Business Media, 1996. [Online]. Available: https://doi.org/10.1007/978-3-662-03269-5

[35] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186, accessed 2019-05-22. [Online]. Available: http://pmg.csail.mit.edu/papers/osdi99.pdf

[36] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. USENIX Association, 2014, pp. 305–319, accessed 2019-05-22. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[37] K. Saito and H. Yamada, "What's so different about blockchain? – blockchain is a probabilistic state machine," in *Distributed Computing Systems Workshops (ICDCSW), 2016 IEEE 36th International Conference on*. IEEE, 2016, pp. 168–175. [Online]. Available: https://doi.org/10.1109/ICDCSW.2016.28

[38] A. Norta, "Self-aware smart contracts with legal relevance," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8. [Online]. Available: https://doi.org/10.1109/IJCNN.2018.8489235

[39] A. M. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies*, 2nd ed. "O'Reilly Media, Inc.", June 2017.

[40] F. Vogelsteller and V. Buterin, "ERC-20 token standard," Ethereum Improvement Proposal, Ethereum Foundation, EIP 20, 2015, accessed 2019-05-23. [Online]. Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

[41] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (SoK)," in *Principles of Security and Trust*, M. Maffei and M. Ryan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 164–186. [Online]. Available: https://doi.org/10.1007/978-3-662-54455-6_8

[42] J. Krupp and C. Rossow, "teEther: Gnawing at Ethereum to automatically exploit smart contracts," in *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 2018, pp. 1317–1333, accessed 2019-05-23. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/krupp

[43] M. Sykes, M. Parzygnat *et al.* (2019) Hyperledger fabric: Chaincode for operators. Revision `98f61f76`. Accessed 2019-05-23. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/chaincode4noah.html

[44] Accord Project LLC. The smart legal contract stack. Accessed 2019-02-25. [Online]. Available: https://www.accordproject.org

[45] C. Cachin and M. Vukolić, "Blockchain consensus protocols in the wild," *arXiv*, no. arXiv:1707.01873, 2017, accessed 2019-05-29. [Online]. Available: http://arxiv.org/abs/1707.01873

[46] L. Bach, B. Mihaljevic, and M. Zagar, "Comparative analysis of blockchain consensus algorithms," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO).* IEEE, May 2018, pp. 1545–1550. [Online]. Available: https://doi.org/10.23919/MIPRO.2018.8400278

[47] M. Raynal, *Fault-Tolerant Message-Passing Distributed Systems: An Algorithmic Approach.* Springer International Publishing, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-94141-7

[48] W. J. Bolosky, D. Bradshaw, R. B. Haagens, N. P. Kusters, and P. Li, "Paxos replicated state machines as the basis of a high-performance data store," in *Symposium on Networked Systems Design and Implementation (NSDI).* USENIX Association, 2011, pp. 141–154, accessed 2019-05-30. [Online]. Available: https://www.usenix.org/legacy/events/nsdi11/tech/nsdi11_proceedings.pdf

[49] G. Vieira and L. E. Buzato, "The performance of paxos and fast paxos," *arXiv*, no. arXiv:1308.1358, 2013, accessed 2019-05-30. [Online]. Available: http://arxiv.org/abs/1308.1358

[50] N. Chaudhry and M. M. Yousaf, "Consensus algorithms in blockchain: Comparative analysis, challenges and opportunities," in *2018 12th International Conference on Open Source Systems and Technologies (ICOSST).* IEEE, 2018, pp. 54–63. [Online]. Available: https://doi.org/10.1109/ICOSST.2018.8632190

[51] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *Communications of the ACM*, vol. 61, no. 7, pp. 95–102, 2018. [Online]. Available: https://doi.org/10.1145/3212998

[52] J. D. Bruce. (2014) The mini-blockchain scheme. Accessed 2019-03-20. [Online]. Available: http://cryptonite.info/files/mbc-scheme-rev3.pdf

[53] H. Kantur and C. Bamuleseyo, "How smart contracts can change the insurance industry: Benefits and challenges of using blockchain technology," 2018, accessed 2019-05-31. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:hj:diva-39899

[54] M. Iansiti and K. R. Lakhani, "The truth about blockchain," *Harvard Business Review*, vol. 95, no. 1, pp. 118–127, 2017, accessed 2019-05-31. [Online]. Available: https://hbr.org/2017/01/the-truth-about-blockchain

[55] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *2017 IEEE International Congress on Big Data (BigData Congress).* IEEE, June 2017, pp. 557–564. [Online]. Available: https://doi.org/10.1109/BigDataCongress.2017.85

[56] M. Mettler, “Blockchain technology in healthcare: The revolution starts here,” in *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom).* IEEE, September 2016, pp. 1–3. [Online]. Available: https://doi.org/10.1109/HealthCom.2016.7749510

[57] K. Fanning and D. P. Centers, “Blockchain and its coming impact on financial services,” *Journal of Corporate Accounting & Finance*, vol. 27, no. 5, pp. 53–57, 2016. [Online]. Available: https://doi.org/10.1002/jcaf.22179

[58] Facebook Inc. (2019) Facebook reports fourth quarter and full year 2018 results. Accessed 2019-06-03. [Online]. Available: https://s21.q4cdn.com/399680738/files/doc_financials/2018/Q4/Q4-2018-Earnings-Release.pdf

[59] Alphabet Inc. (2019) Alphabet announces fourth quarter and fiscal year 2018 results. Accessed 2019-05-31. [Online]. Available: https://abc.xyz/investor/static/pdf/2018Q4_alphabet_earnings_release.pdf

[60] M. Giancaspro, “Is a ‘smart contract’ really a smart idea? insights from a legal perspective,” *Computer Law & Security Review*, vol. 33, no. 6, 2017. [Online]. Available: https://doi.org/10.1016/j.clsr.2017.05.007

[61] G. Weiss, *Multiagent Systems.* MIT Press, 2013.

[62] H. P. Levy. (2018, October) The reality of blockchain. Accessed 2019-06-14. [Online]. Available: https://www.gartner.com/smarterwithgartner/the-reality-of-blockchain

[63] M. Mitzenmacher, “Compressed bloom filters,” *IEEE/ACM Transactions on Networking (TON)*, vol. 10, no. 5, pp. 604–612, 2002. [Online]. Available: https://doi.org/10.1109/TNET.2002.803864

[64] G. Caronni, “Walking the web of trust,” in *Proceedings IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000).* IEEE, 2000, pp. 153–158. [Online]. Available: https://doi.org/10.1109/ENABL.2000.883720

[65] T. D. Huynh, N. R. Jennings, and N. R. Shadbolt, “An integrated trust and reputation model for open multi-agent systems,” *Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 2, pp. 119–154, September 2006. [Online]. Available: https://doi.org/10.1007/s10458-005-6825-4

Part II

Paper A

Selective Blockchain Transaction Pruning and State Derivability

Authors:
Emanuel Palm, Olov Schelén, Ulf Bodin

Reformatted version of paper originally published in:
Crypto Valley Conference on Blockchain Technology (CVCBT), 2018

Selective Blockchain Transaction Pruning and State Derivability

Emanuel Palm, Olov Schelén, Ulf Bodin

Abstract

Distributed ledger technologies, such as blockchain systems, have in recent years emerged as promising platforms for machine-to-machine commerce and other forms of multi-stakeholder applications. However, despite the potential demonstrated by projects such as Bitcoin, Ethereum, and Hyperledger Fabric, the disk space typically required to host a copy of a ledger may be prohibitively large for many categories of devices. In this paper, we introduce an approach for reducing ledger size in blockchain systems, based on arbitrary pruning predicate functions, allowing each network participant to independently select and remove any already applied transactions. We also show that if only pruning certain ledger transactions, the ability to derive an unmodified state data structure from the remaining transactions is maintained. The approach is validated through a supply chain use case utilizing a modified version of Hyperledger Fabric, in which ledger size is reduced by about 84.49% via selective transaction pruning.

1 Introduction

The release of Bitcoin at the end of 2008 [1] marked the beginning of a major technological trend, which has grown to encompass a plethora of derivatives and alternatives [2] [3]. These systems, commonly referred to as *Distributed Ledger Technologies* (DLTs), remove the need for using trusted middle-men by instead relying on networks of voting computers, sharing immutable histories of transactions that each participant can verify to be correct. Bitcoin famously hosts a distributed electronic cash platform where there is no central authority [1], and many other use cases have since been proposed. These include political election systems [4], self-driving cars buying their own fuel [2], and *smart factory* [5] machines autonomously buying and selling material and goods within and between factory plants. Use cases typically have in common that they require a shared notion of identity, ownership, and transfer of ownership within a network of machines controlled by different stakeholders, each with its own incentive to maintain the system.

Although distributed ledgers could be a key enabler for emergent technologies such as autonomous machine-to-machine commerce, the storage space typically needed to host a ledger might be prohibitively large for many categories of devices. At the time of writing, the Bitcoin blockchain ledger required about 134 gigabytes of memory, growing steadily at a pace of about 52 gigabytes per year [6], while the ledger of Ethereum [7], another popular blockchain system, required almost 395 gigabytes of memory [8]. Both of these systems are *permissionless* (i.e. open to public participation) meaning that any accepted activity by any participant will likely lead to an increase of ledger size. On the other hand, *permissioned* systems, such as R3 Corda [9], Quorum [10], or Hyperledger Fabric [11], assume that ledgers only be replicated within closed groups of known participants. This may lead to ledgers being smaller, but could also mean that many ledgers are maintained at the same time. In [12], a system is presented where each network stakeholder can maintain private blockchains with any subsets of other network participants in order to protect sensitive interactions.

The problem of ever-growing distributed ledgers is already well-known. Bitcoin and Ethereum allow the use of light clients [1] [7] that only carry limited ledger and state subsets, but are unable to participate in the consensus process. At the cost of more disk usage, Bitcoin Core [13] and related efforts [14] [15] [16] retain the ability to validate transactions by first constructing a complete state data structure, and then pruning all but the n last ledger blocks. These approaches are, however, lacking in that they (1) do not facilitate fine-grained control of what transactions to keep, and (2) do not allow useful data structures to be derived from partially pruned blocks.

In this paper, a pruning strategy applied to Hyperledger Fabric [17] version 0.6 is presented. The strategy is similar to the disk space reclaiming procedure proposed in the original Bitcoin paper [1], with the significant difference that transactions of interest can be left unpruned, and the ability to rebuild a valid state data structure can be maintained if certain conditions apply. Pruning is regulated through the use of arbitrary predicate functions, which can be specified such that only transactions cancelling each other out, or are fully superseded by later trancations, are deleted. For example, certain money transfers may sum up to zero, or, an ownership statement may fully replace one or more previous such. We show that if only removing certain transactions with the mentioned properties, the state data structures derivable from any blocks of interest can be made to remain unaltered.

The contributions of significance presented in this paper are: (1) A strategy for pruning blockchains relying on arbitrary predicate functions for transaction selection. (2) The identifying and outlining of significant implications of pruning, including how it affects expected memory growth and its impact on being able to build unmodified state data structures from pruned blocks. And finally, (3) a supply chain use case facilitated by a modified version of Hyperledger Fabric [17] version 0.6 that serves to validate the presented approach and to exemplify its impact on ledger storage requirements.

The paper is organized as follows: Section 2 describes related efforts in more detail. Section 3 outlines a general strategy for selective blockchain transaction pruning, as well as methods for predicting blockchain growth. Section 4 presents a modification to Hyperledger Fabric making it able to prune its ledger, together with a use case and benchmark results. Section 5 discusses related topics and suggestions for future research. Finally, Section 6 concludes the paper.

2 Related Work

In the original Bitcoin white paper [1], two approaches for reducing ledger size were presented, namely *Reclaiming Disk Space* (RDS) and *Simplified Payment Verification* (SPV). Even though other approaches have been developed, such as for Bitcoin Core [13] or Cryptonite [14], all such known to the authors are generally similar to those originally suggested. For this reason, the names of the original approaches are used here to represent the current state-of-the-art of ledger size reduction.

2.1 Reclaiming Disk Space

The key idea of the RDS approach is to remove some transactions, from the oldest and onward, after applying them to a state data structure [1]. To avoid losing the ability to verify that given blocks or transactions are part of the pruned past, block hashes and Merkle Tree roots [18] may be saved.

Systems relying on probabilistic consensus [19] are required to allow already accepted blocks of transactions to be replaced if a chain of such with sufficient *weight* is presented [1]. It is, therefore, needful to refrain from pruning the n most recent blocks. n is chosen to represent a *weight* (e.g. proof-of-work [1]) that is large enough to avoid the practical possibility for an $m > n$ block reorganization to occur, where m is the number of known main blocks that are superseded by a reorganization. An $m > n$ reorganization leads to a given network participant no longer being able to assume its state to be relevant, meaning the state must be acquired again from its network.

While this approach leaves room for keeping transactions of interest, no strategy for choosing such is described in any work known to the authors. Our solution, on the other hand, gives fine-grained control over what transactions to keep, which we also show can be used to maintain particular system properties, such as being able to rebuild unmodified state data structures from partially pruned blocks.

2.2 Simplified Payment Verification

Using SPV means that a network participant gives up the ability to verify the validity of new transactions by itself, and instead delegates that responsibility to other nodes of its network [1]. Giving this ability up means that only the portions of a blockchain that are of interest to the owner of a given SPV node need to be saved.

In Bitcoin Core [13], SPV nodes download all block headers, but only transactions of interest. As block headers contain Merkle Tree [18] root hashes, any retrieved transaction can be proved to belong to any of the known block headers. Transactions and blocks are requested by providing a number of regular nodes with filters [20], which are used to decide what transactions to relay. Given that a majority of the connected regular nodes convey all desired transactions, a valid partial state can be built from them. Transactions of interest can be selected up to the granularity of instances of addresses, keys or script hashes within individual transactions.

Requesting other nodes to send certain transactions is, however, not the same as leaving out transactions while pruning. Also, the procedure facilitated by Bitcoin Core does not allow filters to consider any particular state of the data structure built by applying all transactions, only the transactions themselves. Our solution allows arbitrary information to be considered while determining if an individual transaction is to be pruned, such as derived states, blocks, significant timestamps, or any other data of interest. Even though it could require more storage space, our solution does not require giving up the ability to participate in the consensus process.

2.3 Other Approaches

Not all distributed ledger systems store their transactions in blocks, or even in a sequential history. Systems such as Swirlds [21], IOTA [22], or R3 Corda [9] use graph-like ledgers rather than chains of blocks, which could yield significantly different pruning implications. Nodes in R3 Corda, for example, can allegedly prune transactions of liquid asset transfers (e.g. money transfers) by requesting an asset issuer to re-issue a given asset, effectively aggregating relevant previous transactions into a single new transaction. Whether or not the pruning strategy presented in this paper could be applied to these graph-like ledgers is not considered, but is left as an open topic for future research.

3 Selective Transaction Pruning

Selective transaction pruning is to be understood as the practice of blockchain network participants independently removing transactions that neither contribute with important system properties or are of particular interest. An important system property could be protection against not having any main blocks after a reorganization, while interest depends on the information needs of a given network participant. In a supply chain scenario, transactions related to lost, damaged or late goods could be particularly interesting to shipping companies, terminal stations, etc, as they might be needed when negotiating with partners or insurance agencies. Computers on delivery trucks or container ships may only find it relevant to carry transactions related to transported goods, while statisticians may want to refrain from pruning anything from their servers to allow for future data analysis.

In order to clarify what general properties are maintained by a blockchain, and how these are changed as transactions are pruned, this section begins with a general anatomy of the blockchain data structure. It is followed by descriptions of how significant blockchain properties may be maintained, the definition of a selective transaction pruning algorithm, and a description of how the presented algorithm can be used to only remove transactions that cancel each other out or have been superseded. Finally, a method for predicting blockchain memory requirements is presented.

Table 1: Significant parts of a general pruned blockchain.

Name	F*	(Invariant)	Description
Indicators			
Block Height	h	$(1 \leq h)$	Authoritative blocks part of a blockchain. $h = 1$ implies the existence of only a genesis block.
Block Position	i	$(1 \leq i \leq h)$	The index of the ith block. The genesis block is represented by $i = 1$.
Transaction Position	j	$(1 \leq j \leq u_i)$	The jth transaction of the ith transactions set, containing u_i transactions.
Reorganization Height	m	$(1 \leq m \leq h)$	Number of main blocks superseded by a reorganization.
Guard Height	n	$(0 \leq n \leq h)$	Number of consequtive blocks, beginning with the most recent, used to protect against reorganizations.
Primitives			
Block	b_i		The ith block.
Header	H_i		Header of ith block, containing block predecessor hash, checksum of block transactions, etc.
Transaction	$t_{i,j}$		The jth event description out of u_i belonging to the ith block.
Transaction Set	$t_{i,*}$		All transactions belonging to the ith block.
State	s_i		A data structure built by applying every $t_{i,j} \in t_{i,*} \in b_i \in \{b_1, \ldots, b_i\}$.
Significant Blocks			
Genesis Block	b_1		The initial block.
Current Block	b_h		The most recent block of a currently authoritative chain of blocks.
Main Blocks	b_*		All blocks, pruned or not, part of a currently authoritative chain of blocks.
Guard Blocks	$b_{*,guard}$		n consequtive blocks, beginning with the most recent, used to protect against reorganizations.
Free Blocks	$b_{*,free}$		$h - n$ blocks thay may or may not contain pruned transactions.
Significant States			
Current State	s_h		The hth state, representing the application of all blockchain transactions.
Derivable State	s'_i		Underived s_i that could be constructed from only local data.
Retrievable State	s''_i		Underived s_i that might be constructible from network peer data.

*F = Formal name.

3.1 Blockchain Anatomy

Primitives

A blockchain is an ordered sequence of *transactions* grouped into *blocks*. Each transaction describes a change to a *state*, which could be any kind of data structure, such as a table of monetary accounts or a key/value store. Each block includes a *header*, which contains enough information to identify any preceding blocks and to determine if these blocks are unmodified. Concretely, the header could include the hash of the preceding block and a hash of the transactions included in the block itself (e.g., [1] [7] [14]).

Significant Blocks

Every blockchain contain two generally significant blocks, the *genesis block* and the *current block*. The genesis block is the first block in its chain, and does sometimes contain special information that dictates how a blockchain can be used. For example, Ethereum uses the genesis block to set an initial mining difficulty and can use it to preallocate funds to given accounts [7]. The current block is the most recent block in its chain, and is typically special since it must have a derived state, or the given node owning the block will not be able to participate in the process of validating new transactions. If a blockchain is part of a system relying on probabilistic consensus [19], such as Bitcoin [1] or Ethereum [7], room must be given for block reorganizations. The currently authoritative chain of blocks is typically referred to as *main blocks*. If a system with probabilistic consensus supports transaction pruning, the n most recent blocks may serve as *guard blocks*, meaning they act as protecting against $m > n$ reorganizations by not containing pruned transactions, where m is the number of main blocks superseded by the reorganization. Any block not being a guard is a *free block*, and could be pruned if desired by its owner. Figure 1 depicts a blockchain with n guard blocks and $h - n$ free blocks. In blockchain systems using non-probabilistic consensus algorithms (e.g. PBFT [23]), such as Quorum [10] or Hyperledger Fabric [11], it becomes unnecessary to make these distinctions. Technically, all blocks are both authoritative and free, as they cannot possibly be superseded by reorganizations.

Significant States

A state data structure, or state, is constructed by applying all transactions in a given block and all of its preceding blocks. Each block could be considered having an associated state, even if that state does not currently exist. Currently existing states are here referred to as being *derived*, while non-existing states are considered being either *derivable* or *retrievable*. The difference in being derivable or retrievable lies in whether or not enough information is had by a network participant to construct a given state, or if that information has to be acquired from other participants. It becomes relevant to talk about retrievable states when a given blockchain system supports pruning, as removing transactions could imply that some states seize to be derivable. Lastly, the state derived from the current block is referred to as the *current state*.

Table 2: Logical descriptions of significant blockchain properties. See Table 1 for definitions.

Expression	Description
Block Predicates	
$Unmodified(b_i) \Longleftarrow Successor(b_{i+1}, b_i) \vee Unmodified(t_{i,*})$	The block b_i is known to be unmodified if either b_{i+1} can be proved to be the successor of b_i, or if the order and state of the transactions in b_i can be proved to be unmodified.
$Successor(b_i, b_{i-1}) \Longleftarrow hash(b_{i-1}) = H_{i,predecessor}$	The block b_i is provably the successor of b_{i-1} if the hash of the predecessor b_{i-1} is equal to the *predecessor* field of the header of b_i.
$Valid(b_i) \Longleftarrow apply(s_{i-1}; t_{i,*}) = s_i$	The block b_i is provably valid if applying all of its transactions to the derived state of the preceding block yields a valid state s_i.
Transaction Predicates	
$Unmodified(t_{i,j}) \Longleftarrow Unmodified(t_{i,*}) \Longleftarrow hash(t_{i,*} \cdot H_{i,\complement checksum}) = H_{i,checksum}$	The transaction $t_{i,j}$ is provably the jth transaction of the block b_i if the hash of all block transactions and relevant block header fields (here assumed to be all fields but the checksum) equals the checksum field of the header.
$Valid(t_{i,j}) \Longleftarrow apply(s_{i-1}; t_{i,j}) \neq \varnothing$	The transaction $t_{i,j}$ is provably valid if applying it to the derived state of its preceding block yields any valid state.
State Predicates	
$Derivable(s'_i) \Longleftarrow apply(s_{i-1}; t_{i,*}) = s_i \vee unwind(s_{i+1}; t_{i+1,*}) = s_i$	s_i can be constructed if all transactions in b_i can be applied to the state derived from the block preceding it, or if the transactions in the successor block b_{i+1} can be used to undo their changes to the successor state s_{i+1}.
$Retrievable(s''_i)$	The state s_i can be retrieved if either blocks, state and blocks, or just the state in question can be retrieved as needed from other network participants, and whatever data is acquired can be trusted to be correct.
Auxiliary Functions	
$apply(s_{i-1}; t_{i,*}) = \begin{cases} s_i, & \text{if } Valid(s_i) \\ \varnothing, & \text{otherwise} \end{cases}$	Represents the application of the given set of transactions $t_{i,*}$, where each transaction is ensured not to modify the given state s_{i-1} such that it describes a non-permitted outcome.
$hash(x) = y$	A function that takes whatever arguments given, combines them and turns them into a checksum. $\cdot$ denotes an argument combination function.
$unwind(s_i; t_{i,*}) = s_{i-1}$	Represents the undoing of the modifications made to the given state s_i by the given set of transactions $t_{i,*}$.

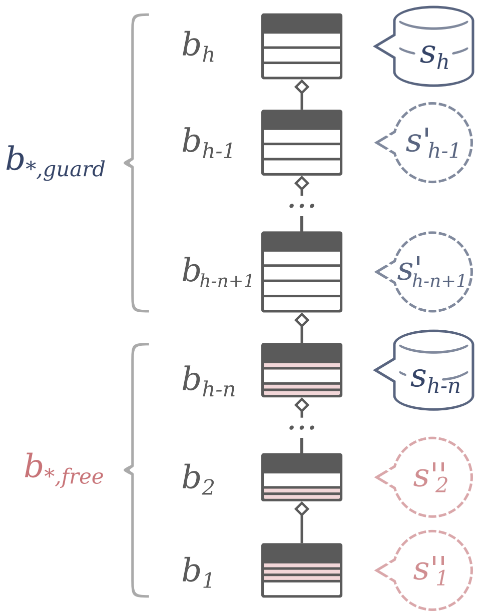


Figure 1: A blockchain of h blocks. The latest n blocks serve as guard against reorganizations. See Table 1 for definitions.

3.2 Pruning and Property Changes

Removing data from a blockchain may free up computer memory, but it could also lead to other property changes. Table 2 lists properties that may be maintained by a blockchain as logical predicates. It should be noted when reviewing the table that pruned blocks could lead to the loss of those properties.

A Property Preserving Hashing Procedure

To be able to prove that a block is unmodified, or that one particular block is the successor of another, requires the use of a hashing function $hash(x) = y$, as shown in Table 2. Since the hashing function is expected to change its output y with the slightest alteration of x, pruning a block will normally lead to its hash changing, consequently making those properties unprovable. This could, however, be mitigated by using a hashing procedure that accounts for missing transactions. One such procedure combines stored hashes of pruned transactions with calculated hashes of the remaining such. Given that $\cdot$ is an arbitrary combination operator, and the definitions in Table 2, the procedure could be defined formally as:

$$\begin{aligned} h_i &= hash(H_{i,\complement checksum} \cdot f(t_{i,*})) \\ f(t_{i,*}) &= g(t_{i,1}) \cdot g(t_{i,2}) \cdot \ldots \cdot g(t_{i,u}) \\ g(t_{i,j}) &= \begin{cases} t_{i,j,saved\ hash}, & \text{if } Pruned(t_{i,j}) \\ hash(t_{i,j}), & \text{otherwise} \end{cases} \end{aligned} \tag{1}$$

The same procedure would be used both to prove a given block is unmodified, and that it is the predecessor of its successor. An alternative procedure using Merkle trees [18] [1] could likely be used to reduce the number of stored hashes, but its definition is left as a topic for future research.

3.3 Selective Pruning Algorithm

The proposed selective pruning algorithm, illustrated in Figure 2, operates in three phases, namely *preparation*, *marking* and, lastly, *sweeping*. The reader should note that the algorithm is described in terms of these phases not because they are strictly required to occur in sequence, but rather because it makes it straightforward to describe the algorithm.

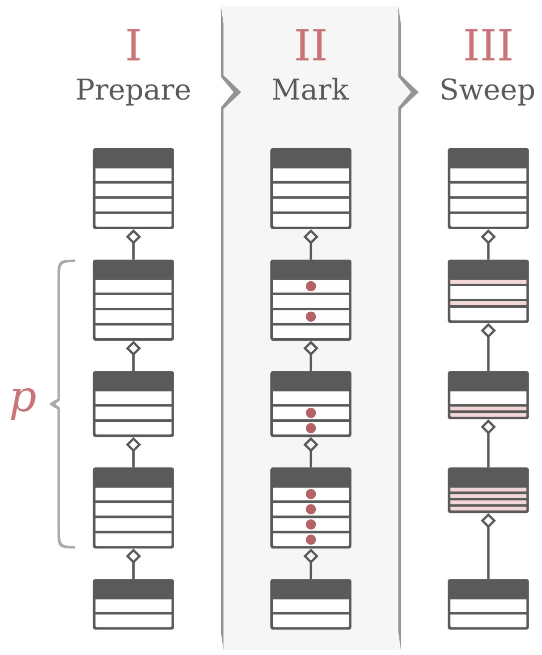


Figure 2: The three phases of the selective pruning algorithm.

I. Preparation

The objective of the preparation phase is to identify the set of blocks that will be pruned p, and to identify and assemble any data d that will be required when considering individual transactions for pruning. That data could include one or more state data structures, which may have to be derived to become available. It could also include actual blocks, or external data not available through the pruned blockchain.

II. Marking

After p and d have been identified and assembled, every transaction $t_{i,j} \in b_i \in p$ is tested using a predicate function T, provided by the initiator of the pruning algorithm. T must satisfy:

$$T : \langle t_{i,j}; d \rangle \mapsto \{true, false\} \tag{2}$$

The position (i, j) of each transaction where T yielded *true* is stored in a set R, which keeps track of all transactions pending removal. An example of a function satisfying T is given in Section 4.3. The formulation and analysis of other T-functions is left as a topic for future research.

III. Sweeping

Lastly, each transaction identified by R is removed. Additional measures may be required to ensure that memory is practically freed when the phase is over, such as compacting blocks to make room for additional such.

3.4 Selective Pruning and Maintaining Derivability

The blockchain properties presented in Section 3.2 and Table 2 imply that removing any transaction from a block leads to the state associated with that block to no longer be derivable, at least in its original form. There are, however, special cases when this is not true. Transactions may be categorized as being *generally significant*, *universally insignificant* or *retroactively insignificant* in relation to state derivability, where transactions in the latter two categorized may be pruned subject to certain conditions. The categories are defined below.

Significant Transactions

If the removal of a particular transaction results in the state associated with its block, or any succeeding block, no longer being derivable in its original form, the transaction in question is to be regarded as generally significant to state derivability. Given the definitions in Tables 1 and 2, it may be expressed formally as:

$$\begin{aligned} Significant(t_{i,j}) \impliedby\ & (Removed(t_{i,j}) \implies \\ & \exists s'_x\ (x \geq i\ \wedge\ \neg Derivable(s'_x))) \end{aligned} \tag{3}$$

For example, consider transaction $t_{1,1}$ in Figure 3. If it is removed, then $[a : 1]$ would no longer be part of s'_1, and as a consequence $[a : 11]$ would no longer be associated with s'_2. As neither s'_1 or s'_2 would be derivable in their original forms, the transaction cannot be removed without impacting the derivability of any state.

Universally Insignificant Transactions

If removing one or more associated transactions in the same block does not lead to the state associated with the block becoming different, then those transactions are together considerable as universally insignificant to state derivability. As each state builds upon its predecessor state, if a set of transaction removed from the same block does not effect the state derived from that block, no subsequent state is affected either. Given the definitions in Tables 1 and 2, and that $t_{i,J} \subset t_{i,*} \in b_i$, it may be described formally as:

$$Insignificant_u(t_{i,J}) \Longleftarrow (Removed(t_{i,J}) \Longrightarrow Derivable(s'_i)) \tag{4}$$

An example can be taken from Figure 3, where $t_{2,3}$ has no impact on c, as $3 \ (mod\ 5) = 3$. If $t_{2,3}$ is removed, then s'_2 remains unchanged, making the transaction universally insignificant to state derivability. To give an example where $t_{i,J}$ contains more than one transaction, consider b_3 in Figure 3. The block contains $add(a, 5)$, $add(a, 3)$ and $sub(a, 8)$, which if all applied leads to no change of a. As these transactions effectively cancel each other out, they become insignificant only if considered together.

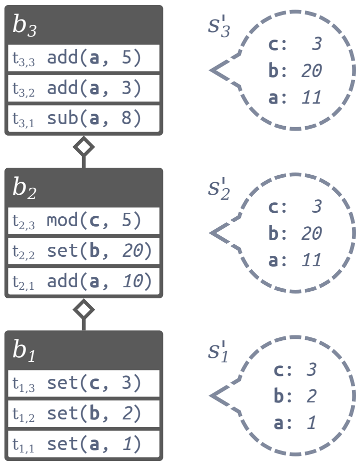


Figure 3: A blockchain of 3 blocks, where each block has a derivable state. Transactions set or modify arbitrary integers associated with alphabetic keys. See Table 1 for definitions.

Retroactively Insignificant Transactions

Given the existence of a set of states that no longer need to be derivable R', if removing one or more associated transactions from any blocks directly related to any member of R' does not lead to any state not in R' becoming different, then those transactions are together considerable as being retroactively insignificant to state derivability. Considering the definitions in Tables 1 and 2, and that $t_{i,J} \subset t_{i,*} \in b_i$, it may be defined as:

$$\begin{gathered} Insignificant_r(t_{i,J}, R') \Longleftarrow (Removed(t_{i,J}) \Longrightarrow \\ \forall s'_x \ (x < i \ \vee \ s'_x \in R' \ \vee \ Derivable(s'_x))) \end{gathered} \tag{5}$$

Consider transaction $t_{1,2}$ in Figure 3. If removed, then s'_1 ceases to be derivable in its original form, but s'_2 remains unaltered. The reason for this is that $t_{1,2}$ is completely superseded by $t_{2,2}$, which overwrites the same b first set by $t_{1,2}$. Therefore, if $s'_1 \in R'$, then $t_{1,2}$ can be removed.

3.5 Predicting Blockchain Growth

Given a blockchain of h blocks, where the mean size of a block is μ_b, the mean size of a block header is μ_H, the mean size of a transaction is μ_t, the mean number of transactions per block is μ_u and ϵ represents any sources of error, then the size S of a blockchain may be defined as:

$$\begin{aligned} S &= h\mu_b + \epsilon \\ \mu_b &= \mu_H + \mu_t\mu_u + \epsilon \end{aligned} \tag{6}$$

If the mean block size is not known and cannot be estimated, Equation 6 may also be formulated with μ_b substituted as:

$$S = h(\mu_H + \mu_t\mu_u) + \epsilon \tag{7}$$

It should be noted that any of the variables in Equation 6 could be expressed as functions over one or more variables, such as time. Typical sources of error ϵ could include platform or storage media restrictions, such as page or block sizes, whether or not blocks are stored in multiple file system files, etc.

Example 1

Some blockchain currently consist of 40000 blocks, and is expected to grow with a rate of 120 blocks per day, where the mean size of a block is known to be constant at 0.30 MB. Given that d is the number of days elapsed since the current time, $h = (40\ 000 + 120d)$, and $\epsilon = 0$, then its size in MB is:

$$S = h\mu_b + \epsilon = (40\ 000 + 120d)0.30 + 0 = 12\ 000 + 36d$$

Accounting for Pruned Blocks

Given that n is the number of unpruned blocks in some blockchain, that $\mu_{p(b)}$, $\mu_{p(H)}$, $\mu_{p(t)}$ and $\mu_{p(u)}$ are the mean sizes of pruned blocks, pruned headers, pruned transactions and average number of pruned transactions per block, respectively, then pruned blockchain size $\tilde{S}$ could be calculated using:

$$\begin{aligned} \tilde{S} &= n\mu_b + (h-n)\mu_{p(b)} + \epsilon \mid h \geq n \\ \mu_b &= \mu_H + \mu_t\mu_u + \epsilon \\ \mu_{p(b)} &= \mu_{p(H)} + \mu_t(\mu_u - \mu_{p(u)}) + \mu_{p(t)}\mu_{p(u)} + \epsilon \end{aligned} \tag{8}$$

Example 2

A blockchain is expected to grow with a pace of 1 block every two minutes, or about 262800 block per year. The mean sizes of both unpruned and pruned blocks are known to be constant at 2.30 MB and 0.50 MB, respectively. Equation 8 is used to calculate the expected size of the blockchain after 19 years. Given that the last $n = 1\ 000$ blocks will need to be kept unpruned, $h = 262\ 800 \cdot 19$, and $\epsilon = 0$, then the blockchain size in MB is expected to be:

$$\begin{aligned}\tilde{S} &= n\mu_b + (h - n)\mu_{p(b)} + \epsilon \\ &= 1\ 000 \cdot 2.30 + (262\ 800 \cdot 19 - 1000)0.5 + 0 \\ &= 2\ 498\ 400\end{aligned}$$

If, on the other hand, the maintainers of the blockchain would have refrained from pruning any blocks, its size in MB would have been:

$$\begin{aligned}S &= h\mu_b + \epsilon \\ &= (262\ 800 \cdot 19)2.30 + 0 \\ &= 11\ 484\ 360\end{aligned}$$

The size reduction gained from pruning is in this scenario $1 - (2\ 498\ 400 \div 11\ 484\ 360) \approx 78.25\%$.

Blockchain Growth Linearity

Pruning a blockchain may yield significant savings in storage space requirements, but can only be used to limit the size of a blockchain to a desired threshold if the mean size of a pruned block $\mu_{p(b)}$ can be or approaches zero. Figure 4 depicts five different growth trajectories, calculated using Equation 8, for blockchains where pruning leads to different changes in mean block size. Only one trajectory, where $\mu_{p(b)} = 0$, represents a fixed storage requirement relative to the number of unpruned blocks n. Hence, blockchain growth may be assumed to always be linear over time, pruning or not, unless the size of a pruned block is effectively zero. Whether or not it is reasonable to remove entire blocks, including their headers, would depend on the expectation that any of the information contained in a block will be useful in the future.

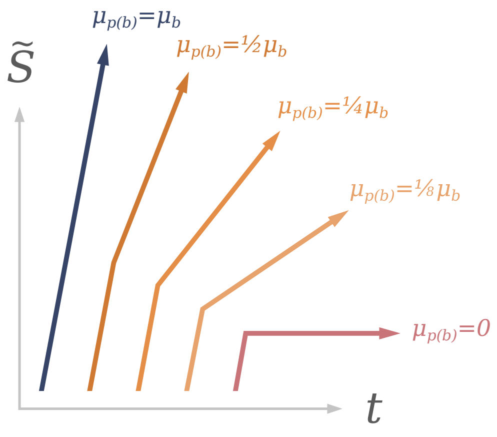


Figure 4: The impact of pruning on blockchain size over time. μ_b *and* $\mu_{p(b)}$ *are assumed to be constants. Each line represents a growing blockchain where* n *blocks are kept unpruned.*

4 Selective Transaction Pruning in Hyperledger Fabric

This section presents the application of the theory presented in Section 3 on Hyperledger Fabric. It begins with a brief overview of the system for readers not already familiar with it and continues with a presentation of the modifications made to it for it to support selective transaction pruning. Finally, it ends with a description of an asset-delivery use case followed by benchmark results, which are presented to verify that selective transaction pruning is possible and yields predictable results.

4.1 Overview of Hyperledger Fabric

Hyperleger Fabric [11] is a blockchain system developed and maintained by Hyperledger [24], a project hosted by the Linux Foundation aimed at creating and maintaining open source blockchain systems for enterprise applications. The system is significantly permissioned and relies on non-probabilistic consensus. The transactions of its blockchain describe invocations of chaincode functions, where a chaincode is a form of containerized application serving as a smart contract [25]. Each valid chaincode invocation may result in the key/value store associated with the system running the chaincode being updated.

4.2 The Pruning Extension

Hyperledger Fabric was significantly extended to support *selective transaction pruning* and *blockchain size analysis*. To allow the abilities in Table 2 associated with hashing to be preserved to some extent, the block *hashing procedure* was also modified. The remainder of this subsection is dedicated to outlining how these features were implemented. The reader should, however, first note that when the presented features were developed, the more stable version of Fabric was deemed to be version 0.6 [17]. This means that details mentioned below may not be true for more recent versions of Fabric.

Selective Transaction Pruning

The pruning algorithm described in Section 3.3 requires a provided predicate function for the purpose of determining which transactions to prune. Hyperledger Fabric allows its blockchain to be modified only via chaincodes, which are deployed independently by the maintainers of the nodes participating in the transaction validation process. Chaincodes do only have to be functionally equivalent for the system to operate, meaning that each maintainer could use its own implementation. To allow each participant to also provide its own pruning predicate function, the chaincode messaging protocol and procedures were extended for the purpose. If provided, the predicate function is called exclusively with transactions generated or validated using the same chaincode, meaning there is no way for one pruning predicate function to decide whether transactions generated by other chaincodes are to be removed. When invoked via an added REST [26] endpoint, the implemented pruning procedure proceeds as follows:

I. Prepare: An effective read-only copy of the most recently assembled state data structure is created and used as d. All blocks, from the most recent to the oldest, used to derive the copied state are used as p.

II. Mark: All non-pruned transactions in p are provided together with d, one at a time, to the pruning predicate function associated with the chaincodes first used to generate them, if such exists. Each transaction provided to a pruning predicate function returning *true* is marked for pruning.

III. Sweep: Each block is deserialized. Marked transactions are hashed, have their contents removed, their types changed to `PRUNED`, and their `metadata` fields set to their hashes. Finally, the block in question is serialized and saved over its previous version. When no more blocks to prune remain, d is removed.

Two things should be noted by the reader about the presented selective pruning procedure. Firstly, Hyperledger Fabric uses a non-probabilistic consensus algorithm. Therefore, no transactions already used to construct a state data structure are technically required to participate in the validation of new blocks (see Section 3.1). Secondly, as pruning predicate functions have arbitrary implementations, guarantees about being able to derive useful state data structures from pruned blocks depend on those implementations. An example of such an implementation is given in Section 4.3.

Blockchain Size Analysis

Measuring the impact of pruning requires a way for block size metrics to be collected. As blocks are stored in serialized form in a database, Hyperledger Fabric was extended to gather block byte sizes by iterating through blocks and checking the number of bytes used to represent them. In order to also measure the sizes of headers and transactions, blocks are deserialized and their transactions serialized and measured, one by one. To calculate the size of each block header, the size sum of the transactions belonging to the same block is subtracted from the size of the entire block. Given that $S_{i,b}$ is the byte size of the ith block, $S_{i,t,*}$ is the byte size of the transactions of the same block, u_i is the number of transactions it contains, $S_{i,H}$ is the block header byte size, and, finally, $\psi(x)$ is a function that serializes and determines the byte size of x, the measurements may be expressed formally as:

$$\begin{aligned} S_{i,b} =& \psi(b_i) \\ S_{i,t,*} =& \sum_{j=1}^{u_i} \psi(t_{i,j}) \\ S_{i,H} =& S_{i,b} - S_{i,t,*} \end{aligned} \tag{9}$$

As Google Protocol Buffers [27] is used as serialization format, there is some byte overhead associated with designating consecutive transactions as members of a collection. This collection overhead varies with the number of transactions in the collection and affects calculated sizes of block headers. To make the collected metrics accessible from outside Hyperledger Fabric, existing REST [26] endpoints were modified such that blocks and transactions are served together with information about their sizes. It should be noted that the reported sizes do not account for any other storage space than that of the blocks themselves. Database indexes and other overhead is not accounted for.

Property Preserving Hashing Procedure

The Block hashing procedure used by Hyperledger Fabric only entails feeding a serialized block to a hash function. If a block is pruned, its serialized form becomes different, which means that it yields a new output if provided again to the hashing procedure. Pruned blocks cannot be proved to be unmodified or to be the predecessors of their successors unless the hashing procedure is modified as described in Section 3.2. As it was assumed to be meaningful to retain these properties even if transactions are pruned, the procedure described by Equation 1 was implemented. Pruned transactions are replaced with only their hashes and a `PRUNED` type indicator. As Hyperledger Fabric blocks hold a `nonHashData` header field, the implementation ignores this field.

4.3 Use Case: Asset Delivery Network

One proposed use case for systems such as Hyperledger Fabric is supply chain management [12]. Supply chains may cross political, geographical or cultural boundaries, and could require the cooperation of stakeholders with conflicting interests. Blockchain technology could be a way to manage fairness where trusted middle-men or other kinds of arbiters are hard to agree on. To verify that selective transaction pruning could be used to reduce ledger size and preserve significant transactions, a naive supply chain scenario was formulated with the intent of reflecting the interactions of different stakeholders in an asset delivery network. No attempt is made to account for subjects of contention, such as false claims about delivery times or asset locations. The scenario, depicted in Figure 5, includes 18 *sites* connected via unidirectional *routes*, through which *assets* are delivered via different kinds of media, such as shipping or trucking. All significant delivery events are registered with a member of a Hyperledger Fabric cluster through a chaincode written for the purpose.

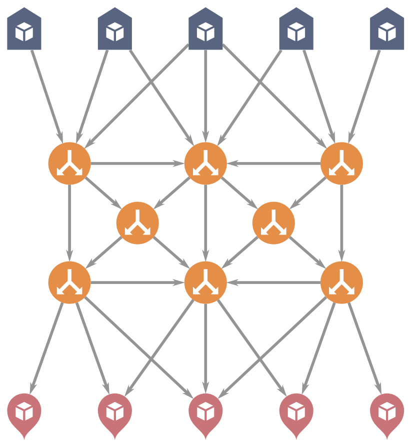


Figure 5: A delivery network of 5 warehouses, 8 distribution centers and 5 delivery points connected via 33 unidirectional routes, allowing assets to be transferred via 387 different paths.

Algorithm 1 Pruning predicate function associated with use case chaincode. The function only returns *true* if *transaction* affects key/value pairs not present in *state*. This will be the case only if *state* represents a point in time after which the asset concerned by *transaction* was successfully delivered and removed.

```
function IsPrunable($transaction$, $state$)
    $keys \leftarrow$ ResolveAffectedKeys($transaction$)
    $values \leftarrow$ GetStateValuesByKeys($state$, $keys$)
    return IsEmpty($values$)
end function
```

Chaincode

The chaincode written to support the use case allows assets to be added to warehouses, forwarded from sites to routes, and received from routes to distribution centers or delivery points. When assets are received the opportunity is given to the receiving party to claim the asset has been lost. Assets successfully received at a delivery point may be removed. Chaincode functions also exist for adding sites and routes, as well as for querying the state of the delivery network. Finally, a pruning predicate function was defined as part of the chaincode. The function is implemented such that the state directly associated with the last main block remains fully derivable, as defined in Section 3.4, which is facilitated by the implementation outlined in Algorithm 1.

Simulation

An external application was written to simulate the behavior of the stakeholders of the delivery network. The application is started with an asset cardinality and a seed for its pseudo-random number generator. When initiated, the application operates in delivery rounds. Each round, assets are generated, forwarded, lost or removed. A bounded random number of assets are generated and assigned to a random path through the delivery network. Assets currently in transit are forwarded, from site to route or from route to site, via their previously assigned paths. Each time an asset is received at a site it has a risk of being lost, determined by a probability property configured for each route. The first delivery round after an asset has been successfully received at a delivery point, it is removed. When no more assets remain to be generated or moved, the application is terminated.

Benchmark

The simulation application was executed with the instruction to move 1000 assets through the scenario delivery network, out of which 120 were randomly selected to be lost at random routes of their delivery paths. When the simulation completed, relevant size metrics were collected, and the one modified Hyperledger Fabric peer used to maintain the blockchain was instructed to prune it. After pruning, size metrics were collected again and then compiled into the statistics available in Table 3. Some of the statistics are also illustrated in Figure 6. Significantly, the size of the entire blockchain was reduced from about 9.51 MB to 1.47 MB, which is a size reduction of circa 84.49%. The reader may note that the average block header size decreased from 277.36 B to 253.22 B, despite that no header values of any block were altered in any way. This reduction is related to the way the header size is calculated, described in Section 4.2.

Table 3: Asset delivery simulation statistics. Fractional numbers are rounded to two decimal places.

	Pruned		Original	
Block Size				
Mean	$\mu_{p(b)} = 3\ 470.28$ B	$\sigma_{p(b)} = 1\ 705.41$ B	$\mu_b = 22\ 376.50$ B	$\sigma_b = 4\ 862.87$ B
Extremes	$\max_{p(b)} = 27\ 407$ B	†$\min_{p(b)} = 219$ B	$\max_b = 28\ 357$ B	†$\min_b = 918$ B
Total	$\tilde{S} = 1\ 474\ 868$ B		$S = 9\ 510\ 013$ B	
Header Size				
Mean	$\mu_{p(H)} = 253.22$ B	$\sigma_{p(H)} = 24.71$ B	$\mu_H = 277.36$ B	$\sigma_H = 29.56$ B
Extremes	$\max_{p(H)} = 313$ B	†$\min_{p(H)} = 151$ B	$\max_H = 313$ B	†$\min_H = 152$ B
Total	$\tilde{S}_H = 107\ 619$ B		$S_H = 117\ 877$ B	
TX Size				
Mean	$\mu_{p(t)} = 123.80$ B	$\sigma_{p(t)} = 201.63$ B	$\mu_t = 850.43$ B	$\sigma_t = 12.59$ B
Extremes	$\max_{p(t)} = 892$ B	$\min_{p(b)} = 68$ B	$\max_t = 895$ B	$\min_t = 815$ B
Total	$\tilde{S}_t = 1\ 367\ 249$ B		$S_t = 9\ 392\ 136$ B	
Avg. TXs/Block	$\mu_{p(u)} = 25.99$	$\sigma_{p(u)} = 5.48$	$\mu_u = 25.99$	$\sigma_u = 5.48$
Cardinalities				
Blocks	$h = 425$		$h = 425$	
Transactions	unpruned = 768	pruned = 10 258	unpruned = 11 044	pruned = 0
Assets	delivered = 880	lost = 120	delivered = 880	lost = 120

TX = Transaction.

† The always empty genesis block, which in this case is 16 B, is not considered.

Projection

Assume that blocks are generated at a rate of 1 block per unit of time t, that the blockchain is expected to keep growing with constant block mean size, and that the blockchain is pruned every time a new block is added. Given the definitions in Tables 1 and 3, Equation 8, $n = 0$, $h = t$, and $\epsilon = 0$, then could the byte size of the blockchain $\tilde{S}$ be:

$$\begin{aligned}\tilde{S} &= n\mu_b + (h - n)\mu_{p(b)} + \epsilon \\ &= 0 \cdot 22\ 376.50 + (t - 0)3\ 470.28 + 0 \\ &= 3\ 470.28t\end{aligned}$$

At $t = 425$ would $\tilde{S} = 3\ 470.28 \cdot 425 = 1\ 474\ 869$, which is approximately the same as $\tilde{S}$ in Table 3. If $n = 100$ blocks were kept unpruned at $t = 425$, then would the size have been:

$$\begin{aligned}\tilde{S} &= n\mu_b + (h - n)\mu_{p(b)} + \epsilon \\ &= 100 \cdot 22\ 376.50 + (425 - 100)3\ 470.28 + 0 \\ &= 3\ 365\ 491\end{aligned}$$

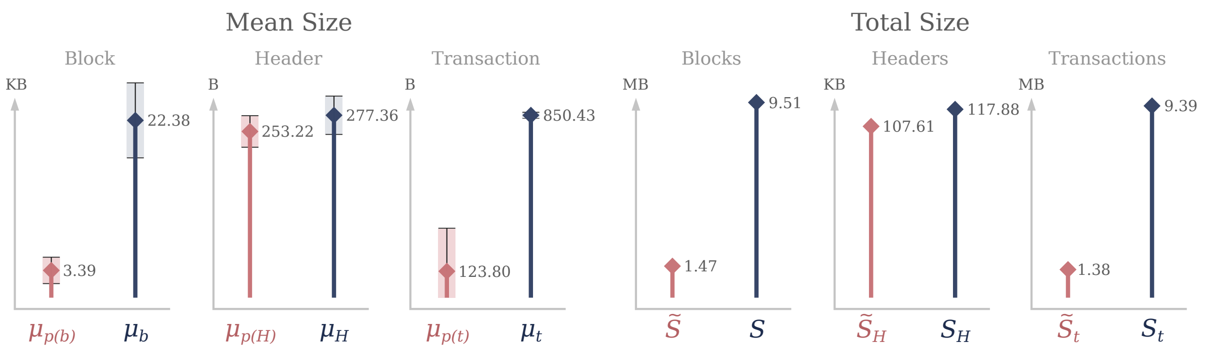


Figure 6: Plotted asset delivery simulation statistics. All numbers are rounded to two decimal places.

5 Discussion

Limiting Blockchain Growth

It was shown in Section 3.5 that pruning can only reduce, not effectually limit, the rate of blockchain growth unless pruned blocks can be removed completely. If this is possible in a given scenario depends on whether older blocks contain useful data, in the context of consensus or otherwise, as explained in Section 3.1.

The Ratio of Prunable Transactions

The blockchain produced by the scenario presented in Section 4.3 could be reduced in size by about 84.49% largely because circa 92.88% of transactions could be pruned. This ratio is high because (1) most of the transactions were subjectively decided to not contain interesting information, and (2) most transactions dealt with information having limited lifespans. Assets were introduced, moved through the sites and routes of the delivery network, and finally removed if nothing exceptional happened. This implies that the gains of pruning could vary greatly with different kinds of use cases.

Identifying Prunable Transactions

In Section 3.4 conditions for pruning transactions are presented, and in Section 4.3 an example of a working pruning predicate function implementation is given. The example implementation is only able to identify transactions that deal with data that has later been removed, but any set of eligible transactions that effectively cancel each other out, or are fully superseded by later transactions, could be prunable. What would function implementations look like for pruning other—potentially highly complicated—sets of transactions? Could such an implementation be identified that is guaranteed to find all prunable transactions modifying, for example, a key/value store? Further work on these and related questions could lead to selective transaction pruning becoming more generally applicable.

Sharing Pruned Blocks

What would happen if a blockchain network participant, perhaps in the process of rejoining its network, was provided with a pruned chain of blocks? Through the use of a hashing procedure similar to that presented in Section 4.2, the received blocks could be decided to contain only valid remaining transactions. There would, however, be no way for the receiver to know whether transactions it deems significant have been pruned, meaning that it cannot be decided whether the state data structures derivable from the received blocks are useful or not. This could perhaps be mitigated by having network participants collectively agree on which states are significant, via state hashes or otherwise. After applying pruned blocks, network participants could determine if the significant states are unmodified. The feasibility of such an approach could be a topic for future research.

Irrevocably Lost Transactions

Even though a blockchain network may continue to function if some transactions are removed by all participants, it could be undesirable that information can be irrevocably lost without any chance for it to be detected. A solution could include assigning different portions of the past to special historian nodes. Another could be to make pruned transactions have a random chance of becoming forever protected from pruning, which could be tuned to make the event of permanent transaction loss unlikely.

Pruning Performance

In this paper, the only performance metric of concern has been that of disk space. It could be expected, however, that the pruning algorithm presented in Section 3.3 may use significant computer resources while being executed, such as primary memory or processor time. As the algorithm requires the consideration of every transaction in every considered block, it could be assumed to have something reminiscent of a linear relationship between the number of considered transactions and processing time. The implementation presented in Section 4.2 is able to execute while also participating in the process of accepting new blocks and updating its current state. If the algorithm cannot be modified to yield better than linear performance characteristics, then there might be additional ways to avoid degrading the performance of more critical systems tasks, such as the consensus process.

6 Conclusions

It is shown in Sections 3 and 4 that selective transaction pruning is possible, theoretically and practically. The anatomy of a general blockchain is presented, as well as descriptions of how such maintains significant properties, a selective pruning algorithm, conditions for selective transaction pruning to not affect significant state derivability, and methods for predicting blockchain growth. A modified version of Hyperledger Fabric [17] is used to demonstrate that a blockchain with transactions from an artificial supply chain scenario could be reduced in size with 84.49% by pruning 92.88% of its transactions. It is our conclusion that selective transaction pruning is a generally viable approach to limiting blockchain growth while keeping transactions of interest, and that it could fruitfully be applied in any context where the benefit of freeing up memory outweighs the gains of having a complete blockchain.

Acknowledgements

Dr. Johan Kristiansson, formerly at Ericsson Research in Luleå, provided indispensable guidance. Also, this work was funded via the *Productive 4.0* project (EU ARTEMIS JU grant agreement no. 737459).

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] M. E. Peck, "Blockchains: How they work and why they'll change the world," *IEEE Spectrum*, vol. 54, no. 10, pp. 26–35, 2017.

[3] T. Bocek, B. B. Rodrigues, T. Strasser, and B. Stiller, "Blockchains everywhere – a use-case of blockchains in the pharma supply-chain," in *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on.* IEEE, 2017, pp. 772–777.

[4] M. Swan, *Blockchain: Blueprint for a new economy.* O'Reilly Media, Inc., 2015.

[5] E. Hofmann and M. Rüsch, "Industry 4.0 and the current status as well as future prospects on logistics," *Computers in Industry*, vol. 89, pp. 23–34, 2017.

[6] Saint Bitts LLC. (2017) Blockchain size. Accessed 2017-11-22. [Online]. Available: https://charts.bitcoin.com/chart/blockchain-size

[7] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.

[8] EtherScan. (2017) Ethereum ChainData size growth (FULL Sync). Accessed 2017-11-22. [Online]. Available: https://etherscan.io/chart/chaindatasizefull

[9] M. Hearn. (2016) Corda: A distributed ledger. Accessed 2018-02-15. [Online]. Available: https://docs.corda.net/_static/corda-technical-whitepaper.pdf

[10] JP Morgan Chase. (2016) Quorum white paper. Accessed 2018-02-15. [Online]. Available: https://github.com/jpmorganchase/quorum-docs/blob/master/Quorum Whitepaper v0.1.pdf

[11] E. Androulaki, A. Barger *et al.*, “Hyperledger fabric: A distributed operating system for permissioned blockchains,” in *Proceedings of the Thirteenth EuroSys Conference.* ACM, 2018, pp. 30:1–30:15. [Online]. Available: http://doi.acm.org/10.1145/3190508.3190538

[12] H. Wu, Z. Li, B. King, Z. Ben Miled, J. Wassick, and J. Tazelaar, “A distributed ledger for supply chain physical distribution visibility,” *Information*, vol. 8, no. 4, 2017, accessed 2018-01-17. [Online]. Available: http://dx.doi.org/10.3390/info8040137

[13] Bitcoin Project. (2017) Bitcoin core. Accessed 2017-11-23. [Online]. Available: https://bitcoin.org/en/bitcoin-core

[14] J. D. Bruce. (2014) The mini-blockchain scheme. Accessed 2017-11-23. [Online]. Available: http://cryptonite.info/files/mbc-scheme-rev3.pdf

[15] R. Dennis, G. Owenson, and B. Aziz, “A temporal blockchain: a formal analysis,” in *Collaboration Technologies and Systems (CTS), 2016 International Conference on.* IEEE, 2016, pp. 430–437.

[16] A. Chepurnoy, M. Larangeira, and A. Ojiganov, “Rollerchain, a blockchain with safely pruneable full blocks,” *arXiv*, 2016, accessed 2018-04-20. [Online]. Available: https://arxiv.org/abs/1603.07926v3

[17] C. Cachin, “Architecture of the hyperledger blockchain fabric,” in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[18] R. C. Merkle, “A certified digital signature,” in *Conference on the Theory and Application of Cryptology.* Springer, 1989, pp. 218–238.

[19] K. Saito and H. Yamada, “What’s so different about blockchain? – blockchain is a probabilistic state machine,” in *Distributed Computing Systems Workshops (ICDCSW), 2016 IEEE 36th International Conference on.* IEEE, 2016, pp. 168–175.

[20] M. Hearn and M. Corallo, “Connection bloom filtering,” Bitcoin Improvement Proposal, Bitcoin Project, BIP 0037, 2012, accessed 2017-12-01. [Online]. Available: https://github.com/bitcoin/bips/blob/ master/bip-0037.mediawiki

[21] L. Baird. (2016) The swirdlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. Accessed 2017-11-27. [Online]. Available: http://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf

[22] S. Popov. (2017, October) The tangle. Accessed 2017-11-27. [Online]. Available: https://iota.org/IOTA_Whitepaper.pdf

[23] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[24] Linux Foundation. (2017) Hyperledger. Accessed 2018-01-26. [Online]. Available: http://hyperledger.org

[25] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[26] R. T. Fielding and R. N. Taylor, *Architectural styles and the design of network-based software architectures.* University of California, Irvine Doctoral dissertation, 2000, vol. 7.

[27] Google. (2018) Protocol buffers. Accessed 2018-01-30. [Online]. Available: https://developers.google.com/protocol-buffers

Paper B

Syntactic Translation of Message Payloads Between At Least Partially Equivalent Encodings

Authors:
Emanuel Palm, Cristina Paniagua, Ulf Bodin, Olov Schelén

Reformatted version of paper accepted for publication in:
IEEE International Conference on Industrial Technology (ICIT), 2019

Syntactic Translation of Message Payloads Between At Least Partially Equivalent Encodings

Emanuel Palm, Cristina Paniagua, Ulf Bodin, Olov Schelén

Abstract

Recent years have seen a surge of interest in using IoT systems for an increasingly diverse set of applications, with use cases ranging from medicine to mining. Due to the disparate needs of these applications, vendors are adopting a growing number of messaging protocols, encodings and semantics, which result in poor interoperability unless systems are explicitly designed to work together. Key efforts, such as Industry 4.0, put heavy emphasis on being able to compose arbitrary IoT systems to create emergent applications, which makes mitigating this barrier to interoperability a significant objective. In this paper, we present a theoretical method for translating message payloads in transit between endpoints, complementing previous work on protocol translation. The method involves representing and analyzing encoding syntaxes with the aim of identifying the concrete translations that can be performed without risk of syntactic data loss. While the method does not facilitate translation between all possible encodings or semantics, we believe that it could be extended to enable such translation.

1 Introduction

Improved IoT device interoperability has become an increasingly important ambition during the last few decades, motivating research within both industry and academia. For instance, the upcoming *Industrial IoT* (IIoT) paradigm, including efforts such as *Industry 4.0* [1], put heavy emphasis on making IoT devices work together to create emergent applications [2] [3]. Factory plant owners are expected to be able to buy sensors, actuators, vehicles and other machinery that can work together with little integration effort. Consequently, finding a means to dynamically facilitate device interoperability becomes a paramount objective. We contribute to this effort by presenting a theoretical method for translating message payloads in transit between interacting endpoints, fitting into systems such as the one depicted in Figure 1.

A significant ambition of this work is to provide message payload translation capabilities to systems such as the Arrowhead Framework [4], a SOA-based IoT framework that supports the creation of scalable cloud-based automation systems. This kind of framework facilitates device service discovery and orchestration at runtime,without any need for human intervention. A would-be Arrowhead translator service could be thought of as an intermediary situated between two communicating systems, as is shown in Figure 1. As work has already been done on creating a multiprotocol translation system [5] for Arrowhead, the translation method we describe in this paper could be thought of as a complement to existing systems only supporting protocol translation.

At the time of writing, the list of message encodings in common use includes XML [6], ASN.1 [7], JSON [8], CBOR [9], Protocol Buffers [10] and many others. The length of this list could be attributed to the diversification of connected computing devices observed during the last few decades. As dimensions and costs of computers have been decreasing, new paradigms—such as the *Internet of Things* (IoT), *Big Data* and *Machine Learning*—have spurred the development of a growing variety of computing hardware, ranging from low-power wearable gadgets to quantum computers. Even though the use of many encodings may be important for optimizing the utility of these devices, it may become a stumbling block when the need to interoperate arises.

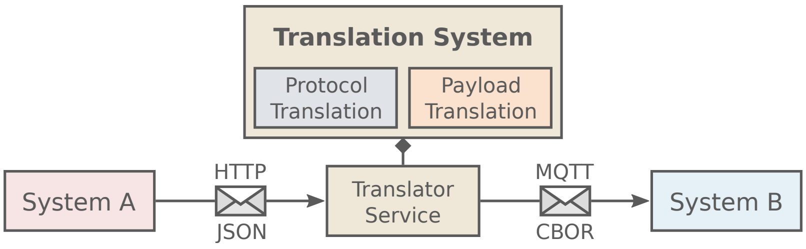


Figure 1: A conceptual translation system producing a transient service for protocol and payload translation at runtime, which allows two otherwise incompatible systems to communicate.

Previous interoperability efforts of relevance include (1) *protocol translation* [5, 11, 12], (2) *encoding libraries* supporting multiple concrete encodings [13, 14, 15] and (3) *ontological translation* [16, 17, 18]). For instance, Derhamy *et al.* present a multiprotocol translator useful within the Arrowhead IoT framework in [5]. Other protocol-related solutions include translation agents, protocol gateways [19] and adapters [20]. These efforts are not, however, concerned with message payloads, only with message protocols. Additionally, while there are several software libraries that can translate message payloads between encodings, none of these formally prove their translations to be lossless, which is an important focus of this work. Libraries of this kind include Jackson [14], Serde [13], Json.NET [15] and many others. Finally, ontological translation is concerned exclusively with message encodings describing effective arrays of triples, while our model can work with any kind of encoding.

In this paper, we present a theoretical method for translating encoding syntaxes. The method is useful for formulating *intersection encodings* that allow for encoded messages to be translated between multiple encodings with intersecting syntaxes without risk of syntactic information loss. In particular, the method is described in terms of representations and validation functions, differing from traditional encoding specifications in that they are concerned with abstract structures and elements instead of strings of bits.

2 Problem Description

The purpose of this work is to provide a rigorous approach to reasoning about and preventing information loss during syntactic encoding-to-encoding translation. However, before presenting such an approach, we first provide our definitions of encoding, translation and lossless translation and describe when a translation is not lossless.

2.1 Message Encodings

We define *encoding* as a set of rules followed to convert *interpreted messages* to and from binary strings. To convert such a message into a string, or to *encode* it, one is required to (1) construct a syntax tree representing the original message, (2) convert the syntax tree into a string of lexemes, and then, finally, (3) turn the lexemes into a binary string.[1] The result can then be *decoded* back into the original message by following the same steps in reverse order as depicted in Figure 2.

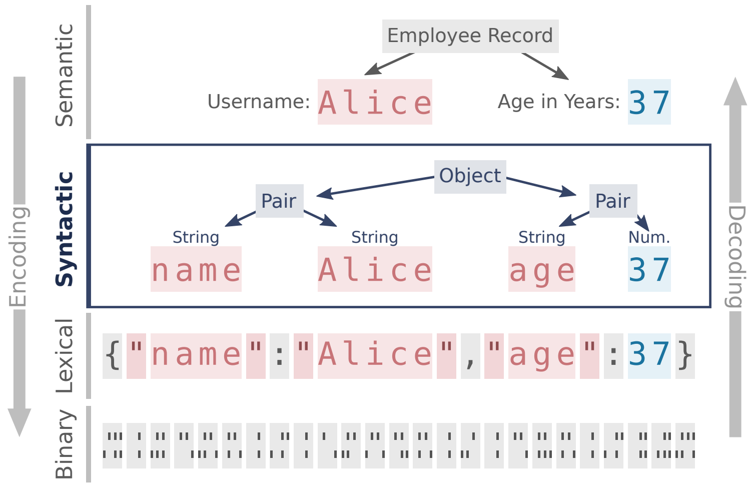


Figure 2: Interpretational levels of a JSON message. At the syntactic level, the types and values of the individual parts of the message can be identified, but their context is unknown.

2.2 Message Translation

Message translation is the process of transforming one encoded message into another with a different encoding, preserving some level of meaning associated with the original message. A translator could be described as a function accepting a string c_a adhering to encoding a and returning either string c_b encoded with b or an error ϵ, such as

$$f_{a,b} : c_a \mapsto \{c_b, \epsilon\} \tag{1}$$

[1] We use the term *syntax tree* exclusively for referring to abstract syntax trees. Concrete syntax trees, or parse trees, are not strictly necessary for either encoding or decoding; hence we do not consider them here.

A translator $f_{a,b}$ can operate at any of the levels depicted in Figure 2, each providing different knowledge about the string being translated. A *syntactic translator* $f_{\Sigma a,b}$ knows only of a syntax tree V_a constructed from its input string c_a, as well as the specifications of its source and target encodings a and b. Facts not recorded in V_a, such as knowledge of certain structures being equivalent, canceling each other out or having insignificant ordering, cannot be acted upon. Specifically, $f_{\Sigma a,b}$ converts V_a it constructed from c_a into an equivalent form V_b in the syntax of the target encoding b, and then encodes V_b into string c_b as shown in Figure 3.

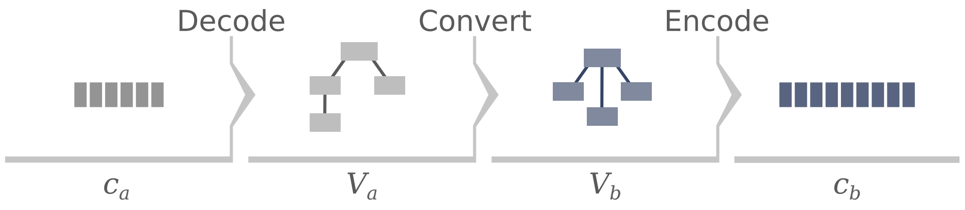


Figure 3: Syntactic translation is the process of decoding a string, converting its syntax tree, and encoding the converted tree.

In the rest of this paper, we assume that syntactic translation from string c_a to string c_b can be performed if a syntax tree V_a can be converted into another tree V_b expressed with the syntax of the desired target encoding. As we are only concerned with syntactic translation, none of the other kinds are given any further treatment.

2.3 Syntactically Lossless Message Translation

For translation from syntax tree V_a into syntax tree V_b to be considered syntactically lossless, V_a and V_b must express the same structural information. This requirement means that if V_a holds an array of three integers, then V_b must also contain an array of the same three integers, even though V_a and V_b being formulated with different encoding type systems. Specifically, a syntactic translation from V_a to V_b using $f_{\Sigma a,b}$ is syntactically lossless only if there exists a translator $f_{\Sigma b,a}$ such that

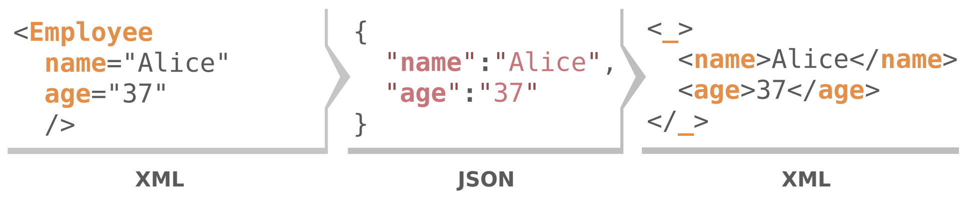
$$f_{\Sigma b,a}(f_{\Sigma a,b}(V_a)) = V_a \tag{2}$$

In other words, if the original syntax tree can be recovered from the translated tree, all original data exist in the translated tree, and translation is lossless.

Note that syntax trees rather than strings are being compared. The input c_a and output c'_a of a lossless translation could differ even if their syntax trees are identical. Many encodings allow for the same syntactic structures to be encoded in multiple ways, such as numbers being allowed to have multiple bases (decimal, hexadecimal, etc.) or text strings being allowed to have particular characters escaped in several ways. Comparisons must be considered as being made between abstract objects, such as numbers, lists or texts, rather than concrete binary strings.

2.4 Syntactically Lossy Message Translation

What would cause a translation to be syntactically lossy? An encoding could be thought of a set of data structures useful for expressing arbitrary messages. When translating between two encodings, a translator is required to express the data of the original message using the data structures of the target encoding. During translation, information may be omitted or changed, so that the original message can no longer be reconstructed. Consider the example in Figure 4, in which a binary string is converted from XML to JSON and back.



```
<Employee
  name="Alice"
  age="37"
  />
```

```
{
  "name":"Alice",
  "age":"37"
}
```

```
<_>
  <name>Alice</name>
  <age>37</age>
</_>
```

Figure 4: XML [6] string translated to JSON [8] and back to XML, resulting in syntactic information loss.

The XML messages in Figure 4 are syntactically different and, therefore, do not satisfy the lossless property defined in Section 2.3. The first message uses attributes, while the second uses child nodes. Additionally, the second message no longer has the original name of its root element. XML provides no type equivalent to a JSON object, which in this case resulted in lossy syntax transformations. A lossless translation between XML and JSON would have required a rigorous syntax transformation scheme, or *syntax simulation*, described further in Section 4.

3 Representation, Validation and Migration

Having established the notion of a syntactic encoding-to-encoding translation, we now proceed to define syntax trees, syntaxes and intersection syntaxes and discuss how the latter can guarantee lossless translation. In particular, we are interested in the representations of these entities and in knowing when a given representation is valid. Our definitions are presented using common constructs of *first-order logic*.[2]

[2]See [21] for an introduction. Note that we allow for any lowercase letter to denote a variable, use one capital letter to denote a set or another collection, use $P(x)$ to denote a *predicate* P with a single term x, use $P(x, y)$ to denote a predicate with two terms x and y, use $\wedge$ instead of & to signify *conjunction* (AND), use $\oplus$ as *exclusive disjunction* (XOR) connective, use $\exists!$ as the uniqueness quantifier and, finally, consider the implication $\theta \to \psi$ to be equivalent to $\psi \impliedby \theta$.

3.1 Syntax Trees

We consider a *syntax tree* to be a directed acyclic graph constructed from *nodes*. Given a *syntax type name* t and a *syntax value* x, we define a node V as a tuple:

$$V = \langle t : x \rangle \tag{3}$$

The type name t is a reference to a certain *syntax type definition* T_i that describes the set of objects that are allowed to occupy the syntax value $x \in \mathbb{X}$. There are two varieties of syntax values: *sequences* and *elements*. If a syntax value is a sequence of child nodes, its node is considered a BRANCH, while if it is an element, its node is considered a LEAF.

Branch Nodes

A BRANCH node V holds a type name t and a sequence of child nodes $S \in \mathbb{S}$, as follows:

$$V = \langle t : S = [V_0, V_1, ..., V_{|S|}] \rangle \tag{4}$$

$\mathbb{S}$ is the set of all possible child node sequences, while $|S|$ is the number of child nodes in S. Above, t names a syntax type definition T_i that identifies a relevant subset of $\mathbb{S} \subset \mathbb{X}$. Various data structures serving to group values together, such as arrays, tuples, sets, dictionaries, classes, etc., are suitably represented by BRANCHES. We consider all sequences to be fundamentally ordered and view the property of being unordered as superimposed at the level of semantics.

Leaf Nodes

Every LEAF node V contains a type name t and an element $e \in \mathbb{E}$ as follows:

$$V = \langle t : e \rangle \tag{5}$$

$\mathbb{E}$ is the set of all objects not considered to be collections of other objects. Above, t names a syntax type definition T_i that identifies a relevant subset of $\mathbb{E} \subset \mathbb{X}$. Such a subset could include all numbers within a specific range or all strings of bytes conforming to a certain text encoding. What concrete members $\mathbb{E}$ contains is subject to interpretation, but useful definitions could include `null`, `true`, `false`, all other enumerators, all numbers and all binary strings.

Example

The syntactic level of the JSON [8] object in Figure 2 could be written with our tuple notation as

$$\begin{aligned}&\langle \text{Object:} [\\ &\quad \langle \text{Pair:} [\langle \text{String:} ''\textit{name}'' \rangle, \langle \text{String:} ''\textit{Alice}'' \rangle] \rangle,\\ &\quad \langle \text{Pair:} [\langle \text{String:} ''\textit{age}'' \rangle, \langle \text{Number:} 37 \rangle] \rangle \ ] \rangle\end{aligned} \tag{6}$$

The message is also shown in Figure 5 that makes the difference between Branch and Leaf nodes more apparent.

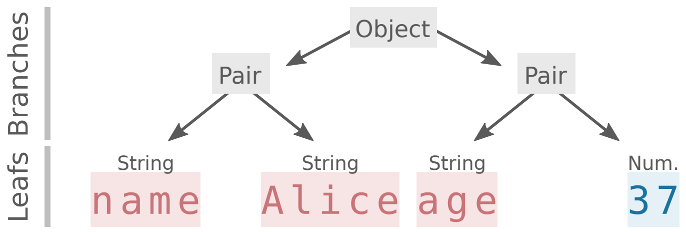


Figure 5: The Branches *and* Leaves *of a JSON [8] syntax tree.* Branches *refer to other nodes while* Leaves *hold elements.*

3.2 Syntaxes

For a syntax tree to be *valid*, it must conform to the structure imposed by an encoding *syntax*. Such validity is typically guaranteed during source string decoding via a set of *parse rules* that effectively limit the set of acceptable lexemes to only those resulting in valid tree nodes. However, syntactic translation entails converting a syntax tree into another such tree, and the new tree may not be guaranteed to be valid. This possibility necessitates the formulation of rules able to verify syntax trees directly. Consequently, we define a *syntax* Σ_j as a collection of such rules, here named *syntax type definitions* $T = \{T_0, T_1, ..., T_{|T|}\}$, and a *root set* R as follows:

$$\Sigma_j = \langle T, R \rangle \tag{7}$$

Each syntax type definition $T_i \in T$ describes one type of syntax tree node permitted by one particular encoding, while the root set R identifies the types of nodes that are allowed to be at the root of a complete syntax tree.

Syntax Type Definitions

A syntax-type definition $T_i \in T$ is a predicate, accepting a syntax tree node V as its only term. T_i *must not* be satisfied unless the type name $t \in V$ equals a type name t_i associated only with T_i. Consequently, every T_i must be defined in general as

$$T_i(V) \impliedby \underbrace{(V = \langle t : x \rangle)}_{1} \wedge \underbrace{(t = t_i)}_{2} \wedge \underbrace{p(x)}_{3} \tag{8}$$

$T_i(V)$ is satisfied only if three conditions are fulfilled: (1) the tested term is a syntax value $V = \langle t : x \rangle$, (2) the syntax-type name $t \in V$ is equal to t_i, which is the name of T_i, and (3) an arbitrary function $p(x)$ yields *true* when provided with the syntax value $x \in V$. To make syntax-type definitions less verbose, the following form is also used:

$$T_i(\langle t : x \rangle) \impliedby (t = t_i) \wedge p(x) \tag{9}$$

For instance, the syntax-type definition for a Boolean LEAF type could be specified as follows:

$$Boolean(\langle t : e \rangle) \impliedby (t = \text{Boolean}) \land (e \in \{1, 0\}) \tag{10}$$

The syntax node $V = \langle \text{Boolean} : 1 \rangle$ would satisfy the $Boolean(V)$ predicate, while the nodes $\langle \text{Boolean} : \mathit{true} \rangle$ and $\langle \text{Binary} : 0 \rangle$ would not.

Syntax Tree Validation

We are now able to represent a syntax as a tuple $\Sigma_j = \langle T, R \rangle$ and are able to determine if any individual syntax tree node is valid. To determine if the entire syntax tree is valid, however, we must examine both the validity of its root node and whether its syntax type is allowed at the root of the tree. For this reason, we define the predicate $Valid_{\Sigma_j}(V, \Sigma_j)$, satisfied as follows:

$$Valid_{\Sigma_j}(V, \Sigma_j) \impliedby \underbrace{(V = \langle t : x \rangle)}_{1} \land \underbrace{(\Sigma_j = \langle T, R \rangle)}_{2} \land \underbrace{(t \in R)}_{3} \land \underbrace{(\exists! T_i \in T)(T_i(V))}_{4} \tag{11}$$

In other words, if (1) V is a syntax tree node, (2) Σ_j is a syntax, (3) the syntax type name $t \in V$ exists in the syntax root set R, and (4) there exists exactly one syntax type definition $T_i \in T \in \Sigma_j$ satisfied by V, then V describes a valid syntax tree according to Σ_j.

Example

We have now provided a sufficient number of definitions to formulate a complete syntax. An example of a naive syntax is given for the JSON [8] encoding in Table 1. A sample JSON syntax tree has already been presented in Equation 6.

3.3 Intersection Syntaxes and Syntax Tree Migration

When comparing any two syntaxes, one may discover that some of their types are similar. Both may define syntax types for numbers, text strings, arrays, maps, etc. Deciding on a list of associations between the types of encoding syntaxes, one might be able to identify syntax trees that would be considered valid by either syntax, i.e., the type name of every node in a tree could be replaced with the name of its corresponding type. We regard such a set of syntaxes with explicit type associations as an *intersection syntax* and refer to the process of changing the type names of a syntax tree as *syntax migration.*

If $C = \{C_0, C_1, ..., C_{|\Sigma|}\}$ is a set of *syntax type intersections* and $\Sigma = \{\Sigma_0, \Sigma_1, ..., \Sigma_{|\Sigma|}\}$ is a set of *concrete syntaxes*, we formally define an intersection syntax $\hat{\Sigma}$ as follows:

$$\hat{\Sigma} = \langle C, \Sigma \rangle \tag{12}$$

Table 1: Naive JSON syntax.

Syntax Type Definitions (T)

Branches

$$
\begin{aligned}
&Object(\langle t : S\rangle) &&\Longleftarrow (t = \text{Object}) \wedge (\forall V_i \in S)(Pair(V_i))\\
&Pair(\langle t : S\rangle) &&\Longleftarrow (t = \text{Pair}) \wedge (S = [V_0, V_1]) \wedge String(V_0) \wedge v(V_1)\\
&Array(\langle t : S\rangle) &&\Longleftarrow (t = \text{Array}) \wedge (\forall V_i \in S)(v(V_i))
\end{aligned}
$$

Leafs

$$
\begin{aligned}
&Number(\langle t : e\rangle) &&\Longleftarrow (t = \text{Number}) \wedge (e \in \mathbb{F} \wedge e \notin \{+\infty, -\infty, \text{NaN}\})\\
&String(\langle t : e\rangle) &&\Longleftarrow (t = \text{String}) \wedge (e \in \text{UTF-8})\\
&True(\langle t : e\rangle) &&\Longleftarrow (t = \text{True}) \wedge (e = \text{true})\\
&False(\langle t : e\rangle) &&\Longleftarrow (t = \text{False}) \wedge (e = \text{false})\\
&Null(\langle t : e\rangle) &&\Longleftarrow (t = \text{Null}) \wedge (e = \text{null})
\end{aligned}
$$

Auxiliary Function

$$
\begin{aligned}
v(V) = \;& Object(V) \oplus Array(V) \oplus Number(V) \oplus\\
& String(V) \oplus True(V) \oplus False(V) \oplus Null(V)
\end{aligned}
$$

Root Set (R)

{Object, Array}

$\mathbb{F}$ is the set of all IEEE 754-2008 [22] binary64 floating-point numbers, while UTF-8 is the set of all UTF-8 compliant byte strings. Note that $u(V)$ does not mention *Pair* since nodes of that type may only occur inside *Objects*.

Syntax Type Intersections

Each $C_i = \{T_0, T_1, ..., T_{|\Sigma|}\}$ is a set of associated syntax type definitions, where exactly one T_k is taken from each associated syntax $\Sigma_j \in \Sigma$.[3] We refer to every such association C_i as a *syntax type intersection* and consider each an effective syntax-type definition, describing the intersection of the sets of syntax values every $T_k \in C_i$ deems valid. Consequently, each C_i can be used to validate a syntax node V as follows:

$$
Valid_{C_i}(V, C_i) \Longleftarrow \underbrace{(V = \langle t : x\rangle)}_{1} \wedge \underbrace{(\exists! T_k \in C_i)(t = t_k)}_{2} \wedge \underbrace{(\forall T_k \in C_i)(T_k(\langle t_k, x\rangle))}_{3} \quad (13)
$$

$Valid_{C_i}(V, C_i)$ is satisfied by three conditions: (1) the tested term is a syntax value $V = \langle t : x\rangle$, (2) the syntax type name $t \in V$ is identical to the name of exactly one associated type definition $T_k \in C_i$, and (3) $x \in V$ together with the type name t_k of each $T_k \in C_i$ satisfies every associated predicate $T_k(V)$. In other words, if a syntax node V names one syntax-type definition in C_i and satisfies all such predicates in C_i, $Valid_{C_i}(V, C_i)$ is also satisfied.

[3]We do not provide any algorithmic means of determining correct or optimal sets of C_i in this paper, even though it could be a relevant topic for future research.

Syntax Tree Validation

Ensuring that the entire syntax tree V is valid according to a certain intersection syntax $\hat{\Sigma}$ requires both that every syntax tree node be valid as asserted by $Valid_{C_i}(V, C_i)$ and that the root node of that tree be valid in every concrete syntax $\Sigma_j \in \Sigma \in \hat{\Sigma}$, as follows:

$$
\begin{aligned}
Valid_{\hat{\Sigma}}(V, \hat{\Sigma}) \iff & \overbrace{(\hat{\Sigma} = \langle C, \Sigma \rangle)}^{1} \land \\
& \underbrace{(\exists! C_i \in C)(ValidRoot(C_i, \Sigma) \land Valid_{C_i}(V, C_i))}_{2} \qquad (14) \\
ValidRoot(C_i, \Sigma) \iff & (\forall T_k \in C_i)((\exists! \langle T, R \rangle \in \Sigma)(t_k \in R))
\end{aligned}
$$

In other words, if (1) $\hat{\Sigma}$ is an intersection syntax $\langle C, \Sigma \rangle$ and (2) there exists exactly one syntax-type intersection $C_i \in C$ where (a) every concrete type definition $T_k \in C_i$ is a valid root type and (b) $Valid_{C_i}(V, C_i)$ is satisfied, then V describes a valid syntax tree according to $\hat{\Sigma}$. If an intersection syntax is able to successfully validate at least one syntax tree V, we refer to its encodings as being *at least partially equivalent*.

Syntax Migration

In Section 2.2, we claimed that if a syntax tree could be converted into another such with another encoding syntax, syntactic translation could be performed. We have just defined $Valid_{\hat{\Sigma}}$ that can be used to determine if a syntax tree would be considered valid by a different syntax if only its type names were changed, or *migrated*, to those of a related syntax. This definition means that if an intersection syntax $\hat{\Sigma}$ can be formulated and a syntax tree V satisfies $Valid_{\hat{\Sigma}}$, V can be translated to any other encoding in $\hat{\Sigma}$.

More formally, given two syntaxes $\{\Sigma_a, \Sigma_b\} \subset \Sigma \in \hat{\Sigma}$ and a syntax tree V_a of Σ_a satisfying $Valid_{\hat{\Sigma}}(V_a, \hat{\Sigma})$, syntax migration is the process of replacing every type name t of every node in V_a with its corresponding type of Σ_b, resulting in V_b. The correspondence between types in Σ_a and Σ_b is established by the syntax-type intersections $C \in \hat{\Sigma}$.

Example

We have again reached the point where we can formulate a concrete example based on the presented theory. As an intersection syntax requires at least two concrete syntaxes, we provide a naive CBOR [9] subset (CBORS). CBOR is used due to being similar to JSON.[4] Disregarding the names of CBOR's and JSON's syntax types, they differ only in JSON requiring the first *Pair* element to be a *String*, in not being able to express the same numbers, and in only CBOR having a dedicated type for arbitrary byte strings. Table 2 outlines the CBORS syntax, while Table 3 shows a JSON/CBORS intersection syntax.

[4]This is no coincidence, as CBOR was designed to be able to encode everything expressible with JSON. For illustrative purposes, however, we do not include enough of the CBOR specification for this to be true here.

Table 2: Naive CBOR Subset (CBORS) syntax.

Syntax Type Definitions (T)

Branches

$Map(\langle t : S \rangle)$	$\Longleftarrow (t = \text{Map}) \wedge (\forall V_i \in S)(Pair(V_i))$
$Pair(\langle t : S \rangle)$	$\Longleftarrow (t = \text{Pair}) \wedge (S = [V_0, V_1]) \wedge d(V_0) \wedge d(V_1)$
$Array(\langle t : S \rangle)$	$\Longleftarrow (t = \text{Array}) \wedge (\forall V_i \in S)(d(V_i))$

Leafs

$Integer(\langle t : e \rangle)$	$\Longleftarrow (t = \text{Integer}) \wedge (e \in \mathbb{Z} \wedge -2^{64} < e < 2^{64})$
$ByteString(\langle t : e \rangle)$	$\Longleftarrow (t = \text{ByteString}) \wedge (e \in \text{STRING})$
$TextString(\langle t : e \rangle)$	$\Longleftarrow (t = \text{TextString}) \wedge (e \in \text{UTF-8})$
$True(\langle t : e \rangle)$	$\Longleftarrow (t = \text{True}) \wedge (e = \text{true})$
$False(\langle t : e \rangle)$	$\Longleftarrow (t = \text{False}) \wedge (e = \text{false})$
$Null(\langle t : e \rangle)$	$\Longleftarrow (t = \text{Null}) \wedge (e = \text{null})$

Auxiliary Function

$$d(V) = Map(V) \oplus Array(V) \oplus Integer(V) \oplus ByteString(V) \oplus TextString(V) \oplus True(V) \oplus False(V) \oplus Null(V)$$

Root Set (R)

{Map, Array, Integer, ByteString, TextString, True, False, Null}

STRING is the set of all possible byte strings. Note that $d(V)$ does not mention *Pair* since nodes of that type may only occur inside *Maps*.

Table 3: JSON/CBORS intersection syntax.

Syntax Type Intersections (C)

Σ_{JSON}		Σ_{CBORS}
Object	$\leftrightarrow$	*Map*
Pair	$\leftrightarrow$	*Pair*
Array	$\leftrightarrow$	*Array*
Number	$\leftrightarrow$	*Integer*
	$\not\leftrightarrow$	*ByteString*
String	$\leftrightarrow$	*TextString*
True	$\leftrightarrow$	*True*
False	$\leftrightarrow$	*False*
Null	$\leftrightarrow$	*Null*

Associated Syntaxes (Σ)

$\{\Sigma_{\text{JSON}}, \Sigma_{\text{CBOR}}\}$

$\leftrightarrow$ denotes syntax type correspondence, while $\not\leftrightarrow$ signifies a syntax type not having a corresponding type.

Consider the JSON syntax tree in Equation 6. Assuming that we desire to convert this syntax tree to CBORS, we first ensure that it is valid according to our intersection syntax $\hat{\Sigma}$ by testing if it satisfies $Valid_{\hat{\Sigma}}(V, \hat{\Sigma})$. After ensuring this, we proceed to migrate it to the desired syntax, resulting in

$$\begin{aligned}
&\langle \text{Map: } [\\
&\quad \langle \text{Pair: } [\langle \text{TextString: } "name" \rangle, \langle \text{TextString: } "Alice" \rangle] \rangle, \\
&\quad \langle \text{Pair: } [\langle \text{TextString: } "age" \rangle, \langle \text{Integer: } 37 \rangle] \rangle \ ] \rangle
\end{aligned} \tag{15}$$

A JSON syntax tree V not satisfying $Valid_{\hat{\Sigma}}(V, \hat{\Sigma})$ would have to refer to a number that is not an integer in the range $(-2^{64}, 2^{64})$. If we, on the other hand, were translating from CBORS to JSON, an unsatisfactory CBORS syntax tree V would have to use a $Pair$ with a node that is not a $TextString$ as the first element, an integer not expressible as a binary 64-bit IEEE float [22], or contain any $ByteString$.

4 Conclusions and Directions for Future Work

The problem of heterogeneous system interoperability has received significant attention in recent years. However, while efforts have made to translate message protocols, the problem of rigorous message payload translation has been largely ignored. In this paper, we presented a theoretical method for message payload translation that facilitates preventing information loss during syntactic encoding-to-encoding translation. The method is described in terms of representations, validation, intersections and migration of syntax trees.

The formalism of representation is a key aspect of the proposed method. If there is any ambiguity, the result may be erroneous. Therefore, the method needs to be used correctly and rigorously. Despite possible disadvantages, we believe that a method of defining various encodings strictly is necessary for being able to correctly translate message payloads between heterogeneous systems.

As the interpretation of a given encoding specification may leave room for ambiguity, multiple incompatible syntaxes could be formulated for that encoding. To prevent such a case, vendors and developers would be responsible for the implementation and provision of syntaxes, avoiding the mismatch between the used encoding and syntax.

Future work includes the extension and refinement of the method, which involves investigating various aspects of syntax simulation and translator implementation.

The syntactic translation solution we presented in this paper only allows for conversions between encodings with intersecting syntaxes. To be able to translate *any* syntax tree to *any other* encoding, there must be a way to simulate syntactic structures that cannot be expressed in the native type system of the target encoding. Such simulation would require one to reason about the types of simulated data structures the receiver of a translated message would be able to interpret correctly, i.e., the activity of simulation could be regarded as constructing new encodings out of existing encodings.

Lastly, we believe that the implementation and evaluation of a multiencoding translator using our translation method would be a significant complement to this work.

Acknowledgements

This work was funded via the *Productive 4.0* project (EU ARTEMIS JU grant agreement no. 737459).

References

[1] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering*, vol. 6, no. 4, pp. 239–242, 2014. [Online]. Available: http://dx.doi.org/10.1007/s12599-014-0334-4

[2] P. Varga, F. Blomstedt, L. L. Ferreira *et al.*, "Making system of systems interoperable — the core components of the arrowhead framework," *Journal of Network and Computer Applications*, vol. 81, pp. 85–95, 2017. [Online]. Available: https://doi.org/10.1016/j.jnca.2016.08.028

[3] M. Weyrich and C. Ebert, "Reference architectures for the internet of things," *IEEE Software*, vol. 33, no. 1, pp. 112–116, 2016. [Online]. Available: https://doi.org/10.1109/MS.2016.20

[4] J. Delsing, *Iot automation: Arrowhead framework.* CRC Press, 2017.

[5] H. Derhamy, J. Eliasson, and J. Delsing, "Iot interoperability—on-demand and low latency transparent multiprotocol translator," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1754–1763, October 2017. [Online]. Available: https://doi.org/10.1109/JIOT.2017.2697718

[6] T. Bray *et al.*, "Extensible markup language (XML) 1.1 (second edition)," W3C, W3C Recommendation, Aug 2006, accessed 2018-06-12. [Online]. Available: http://www.w3.org/TR/2006/REC-xml11-20060816

[7] International Telecommunication Union, "Information Technology – ASN.1 Encoding Rules – Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)," ITU-T Recommendation X.690, July 2002, accessed 2018-06-12. [Online]. Available: http://handle.itu.int/11.1002/1000/12483

[8] T. Bray, "The JavaScript Object Notation (JSON) Data Interchange Format," RFC 7159, Mars 2014. [Online]. Available: https://doi.org/10.17487/RFC7159

[9] C. Bormann and P. E. Hoffman, "Concise Binary Object Representation (CBOR)," RFC 7049, October 2013. [Online]. Available: https://doi.org/10.17487/RFC7049

[10] Google. (2018) Protocol buffers. Accessed 2018-01-30. [Online]. Available: https://developers.google.com/protocol-buffers

[11] C. Lerche, N. Laum, F. Golatowski *et al.*, “Connecting the web with the web of things: lessons learned from implementing a coap-http proxy,” in *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)*, October 2012. [Online]. Available: https://doi.org/10.1109/MASS.2012.6708525

[12] K. L. Calvert and S. S. Lam, “Adaptors for protocol conversion,” in *INFOCOM ’90, Ninth Annual Joint Conference of the IEEE Computer and Communication Societies*, vol. 2. IEEE, June 1990, pp. 552–560. [Online]. Available: https://doi.org/10.1109/INFCOM.1990.91294

[13] D. Tolnay *et al.* (2018) Serde. Accessed 2018-06-13. [Online]. Available: https://serde.rs

[14] T. Saloranta *et al.* (2018) Jackson project home @ GitHub. Accessed 2018-06-13. [Online]. Available: https://github.com/FasterXML/jackson

[15] Newtonsoft. (2018) Json.NET. Accessed 2018-06-13. [Online]. Available: https://www.newtonsoft.com/json

[16] M. Ganzha, M. Paprzycki, W. Pawłowski *et al.*, “Streaming semantic translations,” in *21st International Conference on System Theory, Control and Computing (ICSTCC)*, 2017, pp. 1–8. [Online]. Available: https://doi.org/10.1109/ICSTCC.2017.8107003

[17] D. Dou, D. McDermott, and P. Qi, “Ontology translation on the semantic web,” in *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, R. Meersman, Z. Tari, and D. C. Schmidt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 952–969. [Online]. Available: https://doi.org/10.1007/978-3-540-39964-3_60

[18] H. Chalupsky, “Ontomorph: A translation system for symbolic knowledge,” in *17th International Conference on Knowledge Representation and Reasoning (KR-2000*, April 2000, pp. 471–482, accessed 2018-07-06. [Online]. Available: https://isi.edu/ hans/publications/KR2000.pdf

[19] M. Jung, J. Weidinger, C. Reinisch *et al.*, “A transparent ipv6 multi-protocol gateway to integrate building automation systems in the internet of things,” in *2012 IEEE International Conference on Green Computing and Communications (GreenCom)*. IEEE, 2012, pp. 225–233. [Online]. Available: https://doi.org/10.1109/GreenCom.2012.42

[20] D. M. Yellin and R. E. Strom, “Protocol specifications and component adaptors,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 19, no. 2, pp. 292–333, 1997. [Online]. Available: https://doi.org/10.1145/244795.244801

[21] S. Shapiro and T. Kouri Kissel, “Classical logic,” in *The Stanford Encyclopedia of Philosophy*, spring 2018 ed., E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2018, accessed 2018-08-29. [Online]. Available: https://plato.stanford.edu/archives/spr2018/entries/logic-classical

[22] IEEE Computer Society, “IEEE standard for floating-point arithmetic,” *IEEE Std 754-2008*, August 2008. [Online]. Available: https://doi.org/10.1109/IEEESTD.2008.4610935

Paper C

The Exchange Network: An Architecture for the Negotiation of Non-Repudiable Ownership Exchanges

Authors:
Emanuel Palm, Olov Schelén, Ulf Bodin, Richard Hedman

Reformatted version of paper accepted for publication in:
IEEE International Conference on Industrial Informatics (INDIN), 2019

The Exchange Network: An Architecture for the Negotiation of Non-Repudiable Token Exchanges

Emanuel Palm, Olov Schelén, Ulf Bodin, Richard Hedman*
*Affiliated with *Volvo Group Trucks Operations, Göteborg, Sweden.*

Abstract

Many use cases coming out of initiatives such as *Industry 4.0* and *Ubiquitous Computing* require that systems be able to cooperate by negotiating about and agreeing on the exchange of arbitrary values. While solutions able to facilitate such negotiation exist, they tend to either be domain-specific or lack mechanisms for non-repudiation, which make them unfit for the heterogeneity and scale of many compelling applications. In this paper, we present the *Exchange Network*, a general-purpose and implementation-independent architecture for digital negotiation and non-repudiable exchanges of *tokens*, which are symbolic representations of arbitrary values. We consider the implications of implementing the architecture in three different ways, using a common database, a blockchain, and our own *Signature Chain* data structure, which we also describe. We demonstrate the feasibility of the architecture by outlining our own implementation of it and also describe a supply-chain scenario inspired by one transportation process at Volvo Trucks.

1 Introduction

With the realization of trends such as *Industry 4.0* [1] and *Ubiquitous Computing* [2], more computing devices are becoming interconnected than ever before. While the coming wave of smart machines may be able to facilitate a plethora of compelling use cases, we believe many of them will be economic in nature. Whether in smart manufacturing, value-chain integration, or product life-cycle analysis [1], goods, services, data, money, or other assets may have to change owners for a given use case to become viable. Every change of ownership is always preceded by some form of negotiation, whether it be accepting a delivery or bartering about a price, and the exchange may have to yield a receipt or other proof. While these negotiations could be handled by humans talking or writing to each other, as we assume to be typical nowadays, a digitized solution results in machines being able to monitor, assist or even participate in the negotiation process.

Systems facilitating negotiation and exchange do exist, with applications such as securities trading [3], resource access [4], and e-procurement [5]. However, these solutions make many assumptions about who is in control of the system, who must be trusted, and what can be negotiated about, which make them unfit for use cases outside of their intended application domains. Other recent efforts build on blockchains or other distributed ledgers to guarantee non-repudiation. However, most of these seem to either assume that a domain-specific negotiation protocol is enough [6] or require that code contracts be written for every automatable use case [7] [8] [9].

In this paper, we present the *Exchange Network* (EN), a general-purpose and implementation-independent architecture for the negotiation and exchange of token ownerships, facilitating a form of marketplace where both humans and computers can (1) negotiate, (2) exchange arbitrary assets or other commitments, and (3) prove that past exchanges have taken place. We show how different ways of implementing the architecture, which is defined in terms of abstract components and messages only, have diverging implications on governance, privacy, the credibility of proofs and system scalability. In particular, we briefly present an implementation example based on the *Signature Chain* data structure, a type of distributed ledger that facilitates privacy by not requiring peers to share records of their interactions with others. Furthermore, we compare the implementation to two other possibilities: one based on a common database and the other based on a permissioned blockchain system [10]. We also describe how an EN can facilitate a simple supply-chain use case in which a transport operations unit coordinates transports with a carrier.

A primary objective of our research efforts is to identify an *unobtrusive* architecture for digital cooperation. We assume that this architecture provides constructs with strong and well-understood analogies in real-world practices, which also do not diverge behaviorally from their real-world counterparts in significant ways. Cooperations can be transient or perpetual, remain unchanged for long periods of time or be renegotiated frequently, have strict privacy requirements or be carried out in public. Additionally, cooperation takes place in settings where different means of adjudication are available, making it relevant to ensure that the applied system is compatible with whatever means of litigation, arbitration, or peer judgement is available. To realize an architecture able to represent such characteristics, we chose *ownership* as the major construct and *negotiation* as the means of changing ownerships and then explore how that decision might affect the properties of any would-be implementations. Rather than building on an existing negotiation protocol, such as FIPA00037 [11], in which parties agree about *actions* to perform, we decided to design our protocol own around the concept of owned *tokens*, which are symbolic representations of arbitrary values. As we assume it is generally desirable to prove who owns what tokens, e.g., in courts of law, we also consider how different types of architecture implementations affect that ability.

2 The Exchange Network

An EN is either a monolithic or distributed application that facilitates a digital marketplace where well-known types of assets can be negotiated about, exchanged, and proven to have been part of past exchanges. Concretely, an EN facilitates coordinated changes to the owners of *tokens*, which could be thought of as symbolizing certain rights or obligations, such as the right of ownership, the obligation to render a service, or the obligation to pay. The architecture facilitates this process through four components, shown in Figure 1.

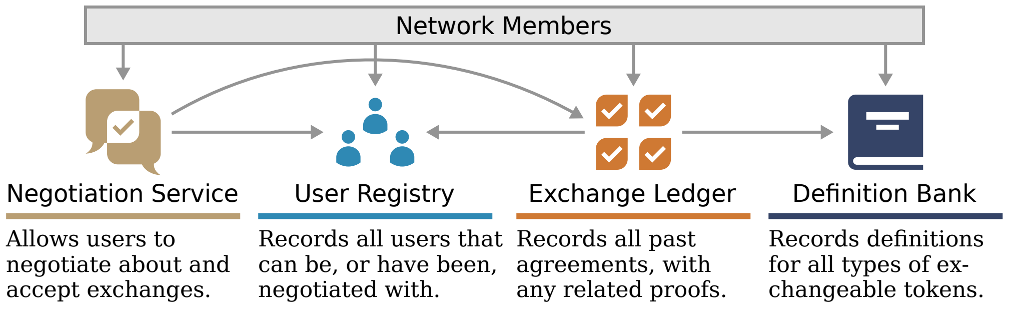

Network Members
Negotiation Service
Allows users to negotiate about and accept exchanges.
User Registry
Records all users that can be, or have been, negotiated with.
Exchange Ledger
Records all past agreements, with any related proofs.
Definition Bank
Records definitions for all types of exchangeable tokens.

Figure 1: Exchange Network components. Arrows denote usage.

Before presenting each of these components in turn, we would like to stress that we make no assumptions about how they store data or coordinate user interactions, as long as data can be accessed and users can interact. The components fulfill abstract functions that can be realized in multiple ways. Later in Section 4, we consider three ways of implementing the components and describe how each way has its own implications on governance and data distribution, as outlined in Figure 2, as well as interaction proofs and performance.

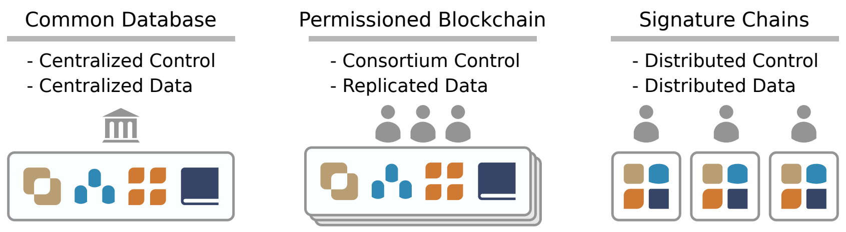


Figure 2: The three considered ways to implement ENs. Note that replication and distribution of data are different in that the former requires each consortium member to own a more or less complete copy of all data, while the latter implies that data are shared only as needed. These implementations are further described in Section 4.

2.1 Negotiation Service

The first component we consider is the *Negotiation Service* (NS), which allows the users of an EN to propose, accept and reject exchanges. It relays *proposals* between pairs of negotiating users, which take turns trying to formulate a proposal that both deem acceptable.[1] If such an acceptable proposal can be identified by those users, the NS submits it to the *Exchange Ledger* (EL) component, which makes sure it can be proven to have taken place to any relevant third party, such as courts of law, insurance agencies, lenders, partners, and so on.

[1]We limit ourselves to negotiations between only two users to avoid making the procedure too complicated. While negotiations with more users may be quite relevant to many scenarios, we leave the topic for future research.

We describe the negotiation procedure in terms of three phases: (1) *qualification*, (2) *acceptance* and (3) *finalization*, which are depicted as a naive state machine in Figure 3.

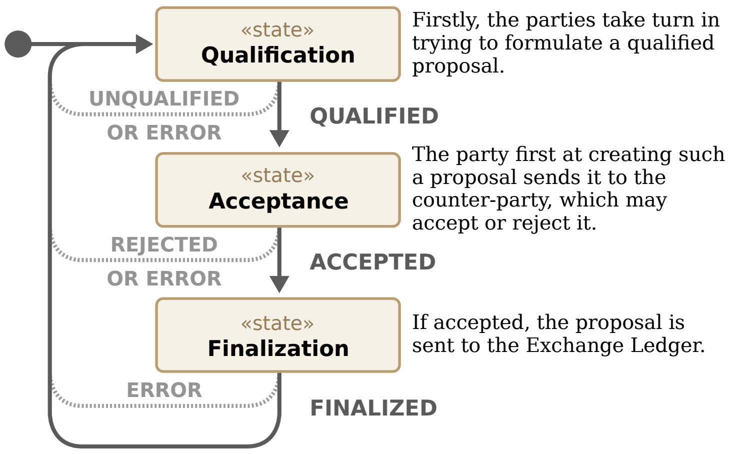

«state»
Qualification
Firstly, the parties take turn in trying to formulate a qualified proposal.
UNQUALIFIED OR ERROR
QUALIFIED
«state»
Acceptance
The party first at creating such a proposal sends it to the counter-party, which may accept or reject it.
REJECTED OR ERROR
ACCEPTED
«state»
Finalization
If accepted, the proposal is sent to the Exchange Ledger.
ERROR
FINALIZED

Figure 3: A state machine showing how two negotiating users could progress from an initial proposal to an accepted and finalized one. A negotiation can be terminated at any time by either participant. Additionally, any number of negotiations can be ongoing at the same time between every pair of users.

Qualification

When a user has found another user that may provide one or more goods, services, or other assets of interest, the first objective is to find a *qualified* proposal believed to be acceptable to both. A qualified proposal is one that leaves no room for ambiguity regarding who would own what, should the proposal be accepted. The proposal is found by having the negotiating users take turn trying to formulate it. If not enough information is available for a candidate proposal to be qualified, an unqualified proposal may be used instead. Unqualified proposals may refer to abstract types of assets, include alternatives, or identify undesired assets. To facilitate the communication required to send these proposals, the NS provides the data types in Figure 4.

The possibility of using the **And**, **Or** and **Not** types, shown in Figure 4, allows users to formulate proposals analogous to those humans make while negotiating. Consider the example in Figure 5.

The example could be thought of as a digital version of the human request *"Can I have a package of 500 small button head screws? Make sure it is not the cap screw kind. I can pay in Euro."* A possible answer to this request is depicted in Figure 6, which could be transliterated as *"I could give you this package of 600 button head machine screws for €4.50."*

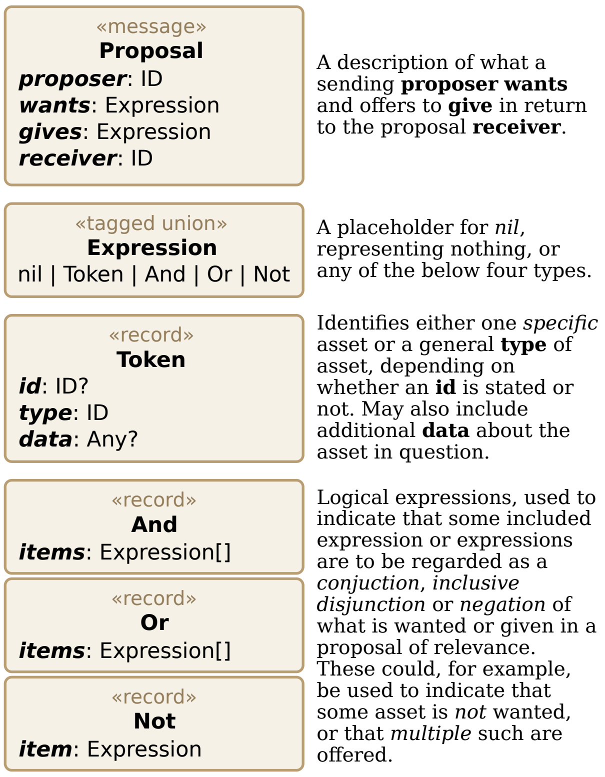

«message»
Proposal
proposer: ID
wants: Expression
gives: Expression
receiver: ID
A description of what a sending proposer wants and offers to give in return to the proposal receiver.
«tagged union»
Expression
nil | Token | And | Or | Not
A placeholder for nil, representing nothing, or any of the below four types.
«record»
Token
id: ID?
type: ID
data: Any?
Identifies either one specific asset or a general type of asset, depending on whether an id is stated or not. May also include additional data about the asset in question.
«record»
And
items: Expression[]
«record»
Or
items: Expression[]
«record»
Not
item: Expression
Logical expressions, used to indicate that some included expression or expressions are to be regarded as a conjuction, inclusive disjunction or negation of what is wanted or given in a proposal of relevance. These could, for example, be used to indicate that some asset is not wanted, or that multiple such are offered.

*Figure 4: The **Proposal** message and associated data types. **ID** represents an arbitrary identifier type, question marks (?) are used for optional values, while brackets ([]) are used to denote array types. Note that the types and fields represent a minimally viable set of proposals, not all useful such.*

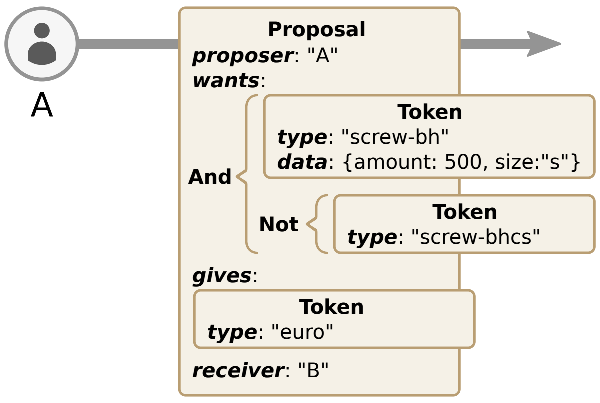


Figure 5: An example of a unqualified proposal.

Let us assume that the first user deems the counter-offer acceptable, carries only a 5 Euro bill, but does not mind giving away the change. That proposal is depicted in Figure 7 and is the first example to be qualified. Because it refers only to tokens with **id** fields and uses no **Or** or **Not** expressions, it is clear who would own what if the proposal would be accepted.

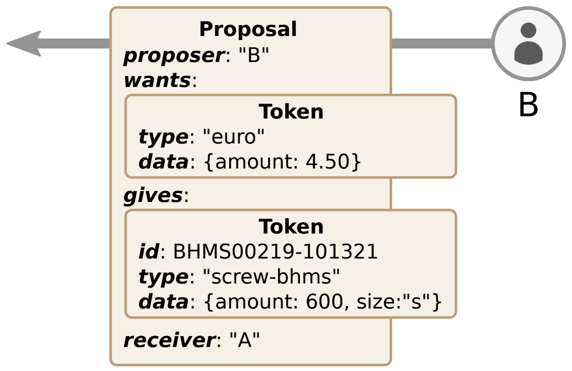


Figure 6: Another unqualified proposal is sent as a reply to that in Figure 5.

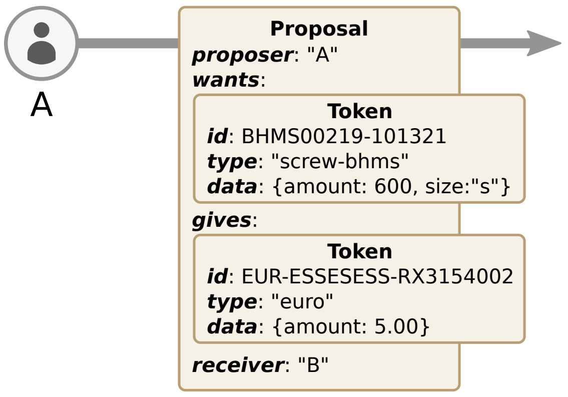


Figure 7: A qualified proposal intended as a reply to the proposal in Figure 6.

Technically, a proposal is qualified if and only if it satisfies the **IsProposalQualified** function outlined in Listing C.1.

```
function IsProposalQualified(proposal):
    return IsExpressionQualified(proposal.wants)
      and IsExpressionQualified(proposal.gives)

function IsExpressionQualified(expression):
    if expression is Token:
        return IsTokenQualified(expression)
    else if expression is And:
        foreach item in expression.items:
            if not IsExpressionQualified(item):
                return false
        return true
    else:
        return false

function IsTokenQualified(token):
    return token.id ≠ nil
```

*Listing C.1: Functions for determining if a **Proposal** is qualified. A qualified proposal must contain only **Token** and **And** instances, and each **Token** must have an **id**. See figure 4 for type definitions.*

Before we continue, we would like to stop and highlight how the proposed system of token expressions makes it possible to formulate logically impossible, or *unsatisfiable*, unqualified proposals, such as *"I want wrench B103, but I do not want wrench B103."* Systems dealing with arbitrary proposals may find it relevant to be able to detect unsatisfiable proposals using an SAT solver [12] or otherwise. Qualified proposals should, however, not be subject to this problem. Formulating unsatisfiable proposals requires one asset to be both wanted/given and not wanted/given at the same time, while qualified proposals do not allow the use of **Not** expressions.

Acceptance

As soon as one user formulates a qualified proposal, the objective becomes to determine if the counter-party also deems that proposal acceptable. While it may seem rather straightforward, it could require extra steps, depending on the NS implementation. Steps include providing signatures or first submitting the proposal to a special verifier service. After having sent a qualified proposal, the counter-party must either reject it by sending a new counter-proposal or accept it using the message in Figure 8.

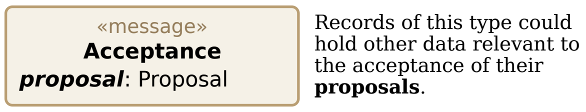

«message»
Acceptance
proposal: Proposal
Records of this type could hold other data relevant to the acceptance of their proposals.

Figure 8: Message used to accept a received qualified proposal. The proposal is rejected by sending a new counter-proposal or terminating the negotiation.

Another way to signal disinterest could be terminating the negotiation. If a counter-proposal is sent or received, the negotiation returns to the **Qualification** phase.

Finalization

When a qualified proposal has been formulated and accepted, it is submitted by the NS to the Exchange Ledger. The users are notified when it is known whether submission succeeded or failed, after which the negotiation returns to the **Qualification** phase. If there is more to negotiate about, negotiation continues. In any other case, the users are free to terminate the negotiation session.

2.2 User Registry

The second logical component, the *User Registry* (UR), is tasked with knowing (1) the *internal* identity and (2) the *external* identities of each EN user. If may provide individual users access to some or all of that information.

Internal Identity

The internal identity allows the EN to refer to a given user, which is what fundamentally enables the network to express that a particular asset belongs to a certain user. What type of internal identifiers are used will depend on the implementation the UR. If both the UR and the Exchange Ledger are hosted by a trusted authority, common integers would likely suffice. In the blockchain and Signature Chains examples in Section 4, public keys [13] would have to be used.

External Identity

External identities, on the other hand, allow users to recognize other users outside the bounds of the EN. To determine where a user is physically located, it may be required to know where to deliver an exchanged good or to whom to render a service. Other details of relevance could be company identifiers, tax numbers, or contact details, which could become relevant in the case of a dispute, to assess user trustworthiness, or to contact a user using a different platform. How external identities are verified or whether multiple such identities are allowed per user depends on the UR implementation.

2.3 Exchange Ledger

The third logical component, the *Exchange Ledger* (EL), allows each user to (1) determine if proposed or already finalized ownership exchanges are *sound* and (2) prove that past ownership exchanges have taken place. While an EL could perhaps fulfill these responsibilities in multiple ways, we conceptualize it as doing so by maintaining and granting access to a history of **Exchanges**, as shown in Figure 9.

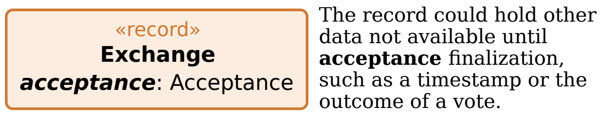

«record»
Exchange
acceptance: Acceptance
The record could hold other data not available until acceptance finalization, such as a timestamp or the outcome of a vote.

Figure 9: Records an accepted and finalized ownership exchange. We also refer to these records as agreements.

Exchange Soundness

In particular, for a given ownership exchange to be sound, it must be known whether

1. the identities of the exchanging parties can be trusted,
2. the proposer owns the *given* tokens, unless created,
3. the acceptor owns the *wanted* tokens, unless created, and
4. the exchanged tokens are well-defined, and their regulations conformed to, as described in Section 2.4.

While we are leaving room for an EL to reject unsound exchanges as part of negotiation finalization, as described in Section 2.1, it may or may not guarantee that all soundness properties are satisfied for each finalized exchange. It might, for example, be difficult to make guarantees about external regulations being adhered to, as explained later in Section 2.4. Users should always validate proposals of concern by themselves to limit the room for mistakes or other errors.

Exchange Proof

Courts of law, insurance agencies, partners, and other parties may be interested in seeing proof that particular ownership exchanges have taken place. How these proofs are facilitated by a particular EL depends on its implementation. The architecture makes no other assumption than that there is some way to present such proofs. We consider how these proofs could be facilitated in Section 4.

2.4 Definition Bank

The fourth and last logical component is the *Definition Bank* (DB). Its main task is to define the implications of owning or creating each type of token, especially in terms of what may be done with the token and the asset it represents. The DB component could be seen as a dictionary, allowing EN users to look up **Definitions**, as outlined in Figure 10, by their names, hashes, or other identifiers.

Internal Regulation

These regulations, which we also refer to as *tests*, ensure that tokens cannot be abused inside an EN. A test could be thought of as a function taking a proposal, an EL and a DB as arguments, returning true only if the proposal is sound. For example, tests could limit the number of times a certain type of token can change owners, restrict creation or ownership of specific tokens to a fixed set of eligible users, or set expiration dates after which some tokens may no longer be exchanged. In other words, they could be used to prevent some unsound ownership exchanges from taking place at all.

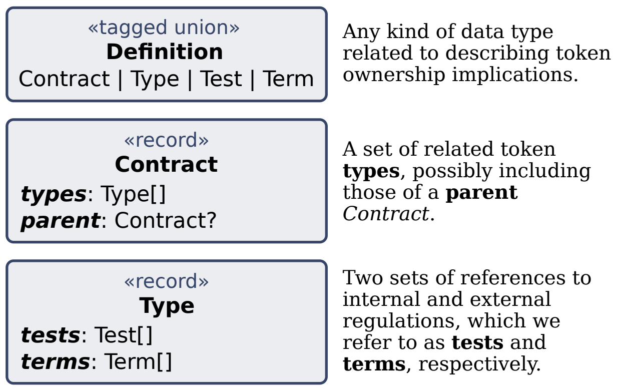

«tagged union»
Definition
Contract | Type | Test | Term
Any kind of data type related to describing token ownership implications.
«record»
Contract
types: Type[]
parent: Contract?
A set of related token types, possibly including those of a parent Contract.
«record»
Type
tests: Test[]
terms: Term[]
Two sets of references to internal and external regulations, which we refer to as tests and terms, respectively.

*Figure 10: The proposed types of DB definitions. Our naive **Contract** contains only **Types**, implying that it can verify contractual events but not facilitate them. This situation is in contrast to systems such as Ethereum [14], where contracts contain executable code.*

External Regulation

These exist to ensure that the assets represented by any EN tokens are not abused outside the bounds of the EN. We refer to these regulations as *contractual terms* or just *terms*. For example, let us assume that two EN users have exchanged one token representing the right to a vehicle repair for another representing a promise of payment. At this point, there is no way for the EN itself to determine if any vehicle is repaired or any payment is made, as these events happen outside the EN's computers. This situation could be mitigated by ensuring that the types referenced by the exchanged tokens contain contractual terms honored by some legal authority, perhaps in the form of legal prose. As long as the exchange itself counts as proof, which we consider in Section 4, that authority could be used to resolve disputes.

3 Signature Chains

To demonstrate the viability of the EN architecture, we now present a implementation designed to maximize the opportunity for exchanges to be kept private. Concretely, the implementation is intended to mimic the way common paper contracts and other forms of signed instruments are used. Such instruments are typically known only to two agreeing parties until the event of a dispute, in which case the instruments are revealed to a legal authority or other arbitrator.

Our implementation operates without mediation, meaning that no set of parties needs to see and ratify each finalized exchange. It relies on a data structure we named the *Signature Chain* (SC), which uses cryptographic signatures and hashing [13] to ensure the (1) authorship, (2) order and (3) definitions of an exchange be denied or altered after its finalization.[2]

[2]The data structure has significant similarities to the *transaction* type employed by R3 Corda [9]. They can both refer to arbitrary definitions and previous interactions by hash, and may also be signed by two parties. Corda transactions, however, carry *state objects*, while SCs carry token exchanges.

3.1 Implementation

Our system consists of a *node* both serving a web client and communicating with other nodes, as depicted in Figure 11.[3]

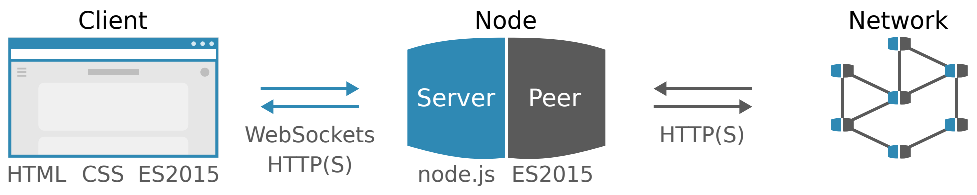


Figure 11: The general design of our EN implementation. Humans operate each Node using a web browser Client. Every Node uses an internal Server to both provide its Client with static HTTP(S) [15] resources and send runtime data via WebSockets [16]. Each Node also contains a Peer module, which is used to communicate with the Peer modules of other Nodes over HTTP(S). The design requires no central data repository or any centralization of control.

Both the node and the client it serves are coded in TypeScript [17], which compiles to ES2015 (JS) [18] before execution. The JS of the node is executed by the node.js runtime [19], while the client HTML [20], CSS [21] and JS are executed by a web browser. We used these technologies and standards mostly because they are familiar to us. There are no inherent reasons why they should be technically superior to any other particular sets of technologies.

The client allows human users to manage negotiations; formulate, modify, accept and reject proposals; list finalized exchanges; list tokens together with the users that own them; and list the users themselves; among other things. It also performs proposal satisfiability tests, mentioned also in Section 2.1.

While the implementation indeed works and can demonstrate the SC concept, some important delimitations were made to reduce implementation effort. For example, all User Registry and Definition Bank data are provided at node startup and cannot change during runtime. Additionally, no communications are encrypted, and client users are not authenticated or authorized.

3.2 Data Structure

A Signature Chain is a chain of records, where each record *may* refer to (1) a previous related record and (2) a definition of relevance. Each record is cryptographically signed [13] by one or more *attestors*, in our case, a proposer and acceptor, and every reference to either a record or definition is the cryptographic hash of that data [13]. By implication, a third party given a chain of records with any associated definitions becomes able to verify that the records

[3]Available at https://github.com/emanuelpalm/en-signature-chains-poc. The paper describes GIT commit `694e3a73a1fbae67b9c106d47bd5`.

1. indeed have been signed by their attestors,
2. were created in a certain order, and
3. always have referred to the provided definitions.

Rather than chains of records being stored in a centralized or replicated repository, each possible pair of EN users maintains and extends its own sets of chains, as depicted in Figure 12. This procedure leaves room for each pair of users to maintain privacy, given that they can agree on not sharing their mutual records with others. Thus, both users of each pair can independently reveal any shared chain to any party of interest, such as a court of law, a partner or another party.

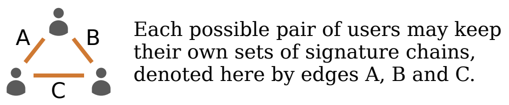

A
B
C
Each possible pair of users may keep their own sets of signature chains, denoted here by edges A, B and C.

Figure 12: The distinct sets of Signature Chains of a three-user EN.

To concretely implement the data structure, a given EN may need to make the messages in Section 2 able to form chains of signatures, which could be realized by amending the **Proposal** and **Acceptance** types as described in Figure 13.

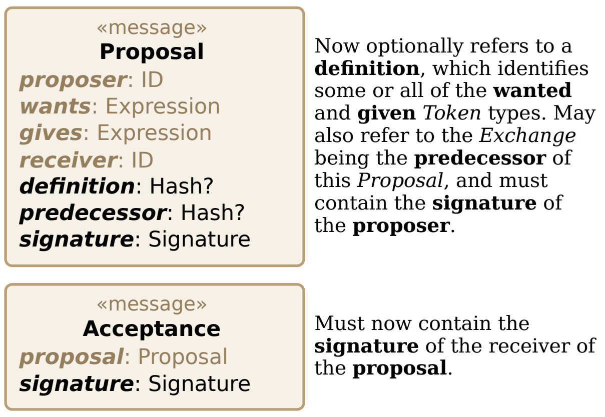

«message»
Proposal
proposer: ID
wants: Expression
gives: Expression
receiver: ID
definition: Hash?
predecessor: Hash?
signature: Signature
Now optionally refers to a definition, which identifies some or all of the wanted and given Token types. May also refer to the Exchange being the predecessor of this Proposal, and must contain the signature of the proposer.
«message»
Acceptance
proposal: Proposal
signature: Signature
Must now contain the signature of the receiver of the proposal.

Figure 13: Amended variants of messages first described in Figures 4 and 8.

Additionally, each relevant EN definition type, such as the ones in Figure 10, ought to refer to its subdefinitions via their hashes. An example of such an SC is illustrated in Figure 14.

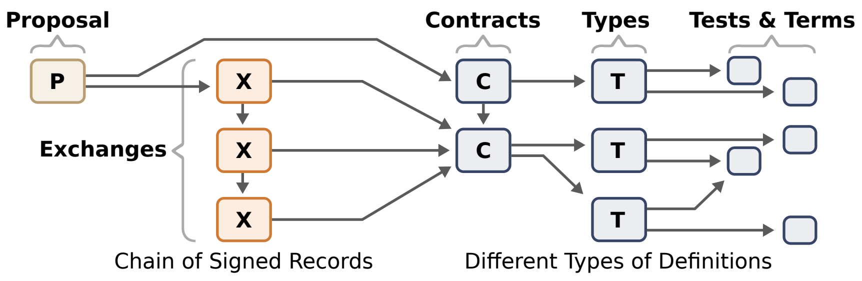


Figure 14: An example EN SC. Arrows denote references by hash.

4 Comparisons of Possible Implementations

The Exchange Network (EN) architecture leaves room for many kinds of concrete implementations. Here, we consider how three possible implementations would affect (1) governance, (2) privacy and data distribution, (3) proofs of interactions and (4) system scalability. The implementation from Section 3 is included in the comparison. We summarize the properties of the implementations in Table 1.

Table 1: Properties of considered possible EN implementations.

	Common Database	*Blockchain*	*Signature Chains*
Governance	Trustee	Consortium	None
Data Distribution	Centralized	Replicated	Distributed
*Interaction Proof**	Trustee Word	Vote	Signatures Only
Scalability	Bound-by-Database	Bound-by-Vote	Unbounded

*Cryptographic signatures can always be used, as in Section 3.2. Note that in blockchain systems, transactions are typically signed only by their issuers by default, as in [22] and [23]. A satisfactory proof requires the signatures of both agreeing parties.

4.1 Common Database

Building an EN around a traditional database, such as MySQL [24], could allow strict control of the members of a given network, as well as rigorous soundness checks of all finalized exchanges. If negotiations are relayed through a centralized Negotiation Service, ongoing negotiations could also be monitored and verified. In either case, a single party must be entrusted with maintaining the network. Due to its appointment and position, the trustee's word could count as proof of interaction, if considered trustworthy. At least the acceptor, proposer and trustee must know of each finalized exchange. System scalability would be limited primarily by the underlying database.

4.2 Permissioned Blockchain

If instead using a system such as Hyperledger Fabric [22] as a foundation, a consortium rather than a single party is entrusted with maintaining the system while enabling the same strict member control and soundness checks as above. However, this process comes at the cost of having to replicate and vote on all data, which significantly limits system scalability. It also means that each consortium member must know of each finalized exchange. While there may be ways to limit interaction visibility through cryptography, those ways would likely also limit the opportunity for exchange soundness to be verified by the consortium. If a fact of significance has been seen and ratified by each maintaining member, a majority testimony could be used as proof that the event has occurred.

4.3 Signature Chains

Without any centralized control, network members are themselves responsible for determining what other parties can be trusted and for ensuring exchange soundness. As no trustee or consortium can testify to the exchange's credibility, only cryptographic signatures can be used as proof. There is no inherent limit to scalability, as there is no global synchronization point. Only the acceptor and proposer of each exchange must know that it has occurred.

5 Supply Chain Use Case

To make the utility of the EN architecture more apparent, we present here an example use case involving the transportation of truck components, akin to how Volvo Trucks currently manages such transports. Some EN implementation is used to propose, accept and register each completed interaction between some *Carrier* (**C**) and a *Transport Operations* (**TO**) unit. The carrier takes a number of components from a manufacturer to some assembly plant, as directed by the **TO**. The purpose of each EN negotiation is to establish a new set of rights and obligations as a result of some contractual term being fulfilled, using the EN messages we describe in Sections 2.1 and 3.2. The four interactions proceed as follows.

1. *Call-Off*: **TO** sends an *EN Propose* to **C**, wanting **C** to accept the obligation to transport a given number of components, guarantee a particular delivery time, and insure the components while in transit. If **C** believes the requested transport capacity will be unavailable, it may reject the proposal or propose another delivery time. We assume **C** replies with *EN Accept*.
2. *Transport Request*: **TO** sends one *EN Propose* to **C** for each individual component, requesting cross-docking and transportation. Each message specifies a pick-up time, a serial number, and sequencing information, which are used to ensure that components are delivered in the order of assembly. **C** would normally commit to each request via an *EN Accept* but could reject them in case of complications. Such early rejections would allow **TO** to immediately search for alternatives to avoid costly delays at the assembly plant.
3. *Pick-Up*: **C** then sends one *EN Propose* to **TO** at the time of loading and departure of each individual component, which would normally be accepted via *EN Accept* messages. Rejections could indicate mismatches in tracking data, perhaps due to human errors. Automatically detecting such errors could save the time and costs that would normally be incurred by manual inspection.
4. *Delivery*: Upon arrival to the assembly plant, **C** sends and *EN Propose* to **TO**, wanting the delivery to be confirmed, which it does by sending an *EN Accept* only if the conditions agreed upon in steps 1 and 2 are met. If, for example, a component would be out of sequence, **TO** could make a counter-proposal for the carrier to agree on a new deal for the deviating item. This setup could lead to faster deviation agreements, easier follow-ups and reduced costs.

While details regarding the obligations of **TO** to **C**, this use case should illustrate how an EN could be used to automatically handle possible deviations online and without human intervention. As an EN is used, all completed negotiations are registered on a shared *Exchange Ledger*, which we hope can be used as evidence in the case of a dispute. Additionally, **C** could use finalized exchanges not yet paid for as a guarantee of future income, which, for example, could be useful when negotiating interest rates with a bank.

6 Discussion

The concepts we present in this paper could be a significant step towards a paradigm in which machines are increasingly able to monitor, assist and autonomously participate in the economy. To make the remaining steps of that journey more apparent, we discuss here (A) shortcomings of our design, (B) the idea of trustless systems, and (C) how our architecture could fit into the context of industry.

6.1 Design Shortcomings

Ambiguity of Ownership

What does it really mean to be the proven owner of a digital token? This problem is fundamental not only to EN tokens but also to paper contracts and other signed instruments. For example, what is a deed of ownership really worth? The answer is that it depends on whether the token or instrument in question is *honored.* This honor is typically established by ensuring that litigation, or some other form of adjudication, is possible in the case of a dispute. Because each party knows that any counter-party can take legal action, a tangible incentive exists to obey the terms of any agreement. For our architecture to be practically useful, effort needs to be spent on formalizing the token tests and terms of Section 2.4 both to avoid the risk of parties interpreting tokens differently and to ensure that courts of law or other adjudicators can be used if desired.

Limits to Negotiation Expressiveness

For an EN to be able to replace human-to-human negotiation, its Negotiation Service (NS) must be able to represent any expression a human could utter in such a context. While we demonstrated how it could represent three expressions in Section 2.1, we know of cases that cannot be easily represented. For example, *"I want at least 1000 small screws and will pay no more than €0."* One way to approach the issue could be to use a corpus of human negotiations and then extend the NS specification until all negotiations in that corpus can be represented.

Involving Secondary Authorities

Having access to one or more trusted authorities can be critical for collaboration to become practically possible. Courts of law, private arbitration firms, insurance agencies, money lenders, or inspection firms could be of relevance to establish trust between parties. In Section 4.2, we showed how more than one authority could be part of maintaining the same EN but believe it will be difficult to gather all useful authorities in the same consortium. A more realistic approach could be to extend the architecture to allow the involvement of secondary authorities, which could veto or approve proposals during negotiation finalization.

Dynamic Definition Creation

In Section 1, we implied that smart contract systems such as [7], [8] and [9] require that code-as-contracts are installed *before* they can be used. In contrast, the abstract negotiation protocol we present in Section 2 does not require definitions to be in place before collaborations can start. As every interaction is a negotiation and the result of every negotiation can be regarded as a new definition, negotiations may result in the creation of new contracts, amendments or exceptions. However, we do not explore how this process can be facilitated in this paper.

Regulating User Identities

We made no assumptions in this paper about how party identification should be regulated while knowing it is an important and delicate issue. Future work should consider how to prevent identity abuse, which could lead to real-world entities being able to avoid being held accountable for their actions.

6.2 Trustless is Not Enough

Readers from the blockchain community may react to our seeming ignorance of blockchain systems being *trustless* [25], which implies they remove the need to rely on trusted middlemen. While blockchain systems indeed do this, they only do it to an extent. Claiming that a system is trustless is the same as saying that it relies on a network of voting computers instead of a traditional authority. Such computer networks are currently unable to perform all useful functions that traditional authorities can. In particular, contemporary blockchain systems are (1) largely limited to acting on signed facts that they cannot verify beyond system boundaries and (2) wield no other fundamental power than deciding what can be recorded in their ledgers. In Bitcoin [23], this arrangement is sufficient to maintain account balances and prevent incorrect transactions, but it is not enough to punish fraudulent users for fooling others into sending them money. In contrast, traditional authorities can make rational decisions regarding the truthfulness of facts, and they could compel misbehaving users into conformance by invoking the power of a police force. Consequently, we do not see how the current state-of-the-art in blockchain technology would be sufficient for typical industrial use cases without also involving trusted authorities.

6.3 Industry Integration

While our solution may help enable entirely new economic use cases, we deem it most relevant to first consider how existing economic processes can be digitized. Digital negotiation and ownership exchange could lead to benefits that are generally in line with process digitization, such as reducing the time needed to complete contractual interactions with new or existing partners or being able to track and analyze those interactions in real-time. Such improvements could lead to (1) increased room for asset accountability, (2) more fine-grained economic forecasting and (3) reduced capital requirements.

There are, however, some roadblocks that need to be cleared before industry adoption can begin. We have already mentioned compatibility with legal authorities and arbitrators, as well as with insurance agencies, money lenders, and inspection firms, which are just a few examples of all potentially relevant authorities. Compatibility with these parties will likely require considerable legal effort both to make the technology lawful and to establish collaboration models and best practices for different industries. Another major roadblock is finding a suitable EN implementation. We have already mentioned that we believe that this implementation needs to be as non-intrusive as possible on existing business models and practices. We had this objective in mind when we designed the Signature Chain implementation in Section 3, but it is far from complete.

7 Conclusions

The EN architecture we proposed in this paper constitutes a generic model for asset transfer, where each asset is represented by a unique digital *Token*. We have shown that a possible implementation of the model based on *Signature Chains* could ensure a high degree of privacy between peers and facilitate horizontal scalability, allowing it to meet the performance requirements and heterogeneity of industrial applications and global markets.

The primary objectives of this work are (1) unobtrusiveness, (2) implementation independence, and (3) reusability. The first objective we address by building on the ideas of negotiation and ownership to formulate the EN domain model. The latter two we approach by separating our architecture into four abstract components, which can be described as follows.

- *Generic Asset Negotiation and Transfer*: The Negotiation Service component provides a clearly defined model for collaboratively refining offers into concrete and tentative asset transfers, which can ultimately be executed atomically and logged immutably. In essence, this conceptual model provides an open high-level protocol specification for *asset-for-asset* transfer negotiations, where an asset represents any form of right or obligation. Like any protocol, it could be part of many kinds of applications and be supported by many different protocol implementations.

- *User Identity Tracking*: The User Registry component keeps track of other EN members in terms of both their internal and external identities. The former allows users to be identified within an EN system, while the latter helps anchor EN members to legal entities or other forms of identities outside the same EN.
- *Exchange Regulation*: The Definition Bank component stores definitions, serving to programmatically and legally define the implications and regulations associated with each kind of exchangeable asset. It fulfills this role by storing regulations, which are both used to verify exchanges and serve as proof of any violations.
- *Exchange Record-Keeping*: Finally, the Exchange Ledger component is tasked with storing an immutable history of ownership exchange records while guaranteeing that each meets the constraints and requirements imposed by the other components. Each record in this ledger serves as proof that a described interaction has taken place and could be useful as evidence if provided to a third party.

It is our belief that the architecture in this paper, or a solution like it, could prove pivotal for digitizing economic interactions between industries and within society at large.

Acknowledgments

We would like to thank Caroline Berg von Linde, Johan Hörmark, Christian Lagerkvist and Jamie Walters at SEB for their indispensable insights regarding banking, money and financial services. We would also like to thank Richard Verbeet at the University of Ulm for making us aware of FIPA00037. This work was funded via the *Productive 4.0* project (EU ARTEMIS JU grant agreement number 737459).

References

[1] L. D. Xu *et al.*, “Industry 4.0: state of the art and future trends,” *International Journal of Production Research*, vol. 56, no. 8, 2018. [Online]. Available: https://doi.org/10.1080/00207543.2018.1444806

[2] M. N. O. Sadiku, Y. Wang, S. Cui, and S. M. Musa, “Ubiquitous computing: A primer,” *International Journal of Advances in Scientific Research and Engineering*, vol. 4, no. 2, 2018. [Online]. Available: https://doi.org/10.7324/IJASRE.2018.32611

[3] V. F. Minton, “Interactive securities trading system,” U.S. Patent 6 014 643, January 11, 2000.

[4] R. Gavriloaie, W. Nejdl, D. Olmedilla, K. E. Seamons, and M. Winslett, “No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web,” in *European Semantic Web Symposium*. Springer, 2004, pp. 342–356. [Online]. Available: https://doi.org/10.1007/978-3-540-25956-5_24

[5] P. H. Ketikidis, A. Kontogeorgis, G. Stalidis, and K. Kaggelides, "Applying e-procurement system in the healthcare: the EPOS paradigm," *International Journal of Systems Science*, vol. 41, no. 3, pp. 281–299, 2010. [Online]. Available: https://doi.org/10.1080/00207720903326878

[6] J. J. Sikorski, J. Haughton, and M. Kraft, "Blockchain technology in the chemical industry: Machine-to-machine electricity market," *Applied Energy*, vol. 195, pp. 234–246, 2017. [Online]. Available: https://doi.org/10.1016/j.apenergy.2017.03.039

[7] A. Norta, "Self-aware smart contracts with legal relevance," in *2018 International Joint Conference on Neural Networks (IJCNN)*, July 2018. [Online]. Available: https://doi.org/10.1109/IJCNN.2018.8489235

[8] Y. Zhang and J. Wen, "The IoT electric business model: Using blockchain technology for the internet of things," *Peer-to-Peer Networking and Applications*, vol. 10, no. 4, pp. 983–994, July 2017. [Online]. Available: https://doi.org/10.1007/s12083-016-0456-1

[9] M. Hearn. (2016) Corda: A distributed ledger. Accessed 2019-02-22. [Online]. Available: https://www.corda.net/content/corda-platform-whitepaper.pdf

[10] M. E. Peck, "Blockchains: How they work and why they'll change the world," *IEEE Spectrum*, vol. 54, no. 10, pp. 26–35, 2017.

[11] "FIPA communicative act library specification," Foundation for Intelligent Physical Agents, FIPA 00037, 2002, accessed 2019-05-09. [Online]. Available: http://www.fipa.org/specs/fipa00037

[12] N. Eén and N. Sörensson. The minisat page. Accessed 2019-01-24. [Online]. Available: http://minisat.se

[13] A. Salomaa, *Public-key cryptography*, 2nd ed., ser. Texts in Theoretical Computer Science. Springer Science & Business Media, 1996.

[14] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014, accessed 2019-02-01. [Online]. Available: http://gavwood.com/paper.pdf

[15] R. Fielding *et al.*, "Hypertext transfer protocol (HTTP/1.1): Message syntax and routing," Internet Request for Comments, RFC Editor, RFC 7230, 2014. [Online]. Available: http://rfc-editor.org/rfc/rfc7230.txt

[16] I. Fette and A. Melnikov, "The WebSocket protocol," Internet Request for Comments, RFC Editor, RFC 6455, December 2011.

[17] G. Bierman, M. Abadi *et al.*, "Understanding TypeScript," in *European Conference on Object-Oriented Programming.* Springer, 2014.

[18] A. Wirfs-Brock, “ECMAScript 2015 language specification,” ECMA International, ECMA-262, 2015. [Online]. Available: http://www.ecma-international.org/ecma-262/6.0

[19] S. Tilkov *et al.*, “Node.js: Using javascript to build high-performance network programs,” *IEEE Internet Computing*, vol. 14, no. 6, 2010.

[20] S. Faulkner *et al.*, “HTML 5.2,” W3C, Working Draft, 2017. [Online]. Available: https://www.w3.org/TR/2018/WD-html53-20180809

[21] T. Atkins and S. Sapin, “CSS syntax module level 3,” W3C, W3C Candidate Recommendation, February 2014. [Online]. Available: http://www.w3.org/TR/2014/CR-css-syntax-3-20140220

[22] E. Androulaki, A. Barger *et al.*, “Hyperledger fabric: A distributed operating system for permissioned blockchains,” in *Proceedings of the Thirteenth EuroSys Conference.* ACM, 2018, pp. 30:1–30:15. [Online]. Available: https://doi.org/10.1145/3190508.3190538

[23] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008, accessed 2019-02-08. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[24] C. Bell, *Introducing the MySQL 8 Document Store.* Apress, 2018. [Online]. Available: https://doi.org/10.1007/978-1-4842-2725-1

[25] K. Christidis and M. Devetsikiotis, “Blockchains and smart contracts for the internet of things,” *IEEE Access*, vol. 4, pp. 2292–2303, 2016. [Online]. Available: https://doi.org/10.1109/ACCESS.2016.2566339

Paper D

Approaching Non-Disruptive Distributed Ledger Technologies

Authors:
Emanuel Palm, Olov Schelén, Ulf Bodin

Reformatted version of paper submitted to:
IEEE Access

Approaching Non-Disruptive Distributed Ledger Technologies

Emanuel Palm, Ulf Bodin and Olov Schelén

Abstract

Distributed ledger technologies have been considered for a plethora of interesting use cases, ranging from supply chain integration to open medical journals. While able to facilitate novel forms of collaboration, the technologies also tend to break with existing business practices by imposing new requirements on cooperation governance, interaction privacy and contract making. In this paper, we identify *distributed consensus algorithms* and *code-as-contracts* as common causes of these paradigmatic divergences, and propose a system design that depends on neither of them. In particular, we present an experimental implementation of our *Exchange Network* architecture that uses a consensus procedure comparable to that of R3 Corda, but that models its interactions as negotiations about ownership exchanges rather than as function invocations and finite state machine transitions. Furthermore, we characterize the current cooperational paradigm and outline six requirements of adherence, as well as considering both how our own solution and how R3 Corda could fulfill them. We conclude that our design approach provides better opportunity for compatibility with conventional legal and business practices than the state-of-the-art.

1 Introduction

Since the introduction of Bitcoin [1] and the subsequent blockchain hype [2], significant effort has been invested into finding new use cases for the technology, not the least during the last few years. Recent works investigate the use of different kinds of distributed ledger technologies to integrate supply chains [3], facilitate tamper-proof land registries [4], create open medical journal systems [5], among a plethora of other examples [6]. A common denominator for many such use cases is the attempt to digitize *contractual cooperation*, a process that until this point has been carried out primarily by humans. While businesses, institutions and individuals rely increasingly on digital systems for managing their assets and collaborations, the step is often yet to be taken for those systems to participate in the negotiation and exchange of value *between* stakeholder systems. Such digital cooperation could enable automated assistance, verification and execution of contractual interactions, which in turn could reduce lead times, cut costs, or enable new types of business models. In the long run, it may even pave the way for economies where most financial interactions are planned, executed and followed-up by computer agents, which would represent the wants and needs of their respective owners.

However, while existing blockchain systems may be able to facilitate new kinds of novel business models and use cases, the concrete technologies they are constructed from tend to make them break with the ways in which businesses and other institutions traditionally have been organizing their collaborations. Through the use of distributed consensus algorithms, as in the case of Bitcoin [1], Ethereum [7] and Hyperledger Fabric [8], they impose new requirements on cooperation governance and interaction privacy by requiring any collaborating parties to review and vote on the validity of each others' interactions. By using executable code rather than legal prose as the primary means of defining contracts, as in Ethereum, Hyperledger Fabric and R3 Corda [9], they effectively sideline existing legal practices and expertise, which are concerned primarily with common contracts, deeds and other signed instruments. Code contracts also inhibit the application of national laws by, for example, making it non-trivial to rigorously map ledger transactions to rights and obligations [10]. Additionally, the distributed consensus algorithms some of these systems rely on have well-known performance and scalability problems in significant contexts [11], which further exacerbates the difficulty of integrating the technology into existing business processes and systems.

In this paper, we consider whether these issues can be mitigated by mimicking the way contractual cooperation is traditionally handled, which typically involves elements such as contracts, receipts, signatures, witnesses, mediators and adjudicators. We note that using voting instead of relying on trusted middle-men is rare and that contracts and receipts tend to be written in well-defined legal language rather than executable computer code. To avoid the disruption caused by these new forms of consensus and contracts, we propose (1) that interactions are made directly between pairs of system participants instead of being mediated by voting networks and (2) that code-as-contracts are replaced with a system of negotiation about digital ownership representations, which we refer to as *tokens*, each of which has its ownership and transfer implications stated in conventional legal language.

Our argumentation for replacing typical code-as-contracts solutions with systems for negotiated ownership exchanges, whenever relevant, is our primary contribution in this paper.

We begin the paper by considering how others avoid this disruption, intentionally or not, after which we present a characterization of conventional contractual cooperation and outline six requirements of adherence. The requirements allow us to reason about the extent the design, which we subsequently present, facilitates collaboration and avoids disruption. To make it more apparent how other solutions relate to our system design, we also consider how R3 Corda [9] can fulfill our requirements and compare it directly to our implementation. Significantly, R3 Corda uses the same fundamental approach to consensus as our solution, even if it provides no direct alternative to code-as-contracts. Finally, we end the paper with a discussion and our conclusions.

2 Related Work

Not all distributed ledger solutions have all the paradigmatic problems we introduce in Section 1. Here, we consider how some of these solutions relate to our contributions.

2.1 No Code-as-Contracts

Bitcoin [12] provides a stack-based language called *Script*,[1] which is limited to specifying programmatic conditions for spending unspent transaction outputs. The Cryptonite system [14], originally created from the Bitcoin source code, omits *Script* support completely to simplify the pruning of older transactions from the blockchain it maintains. Both of these systems are intended to facilitate counter-parts to electronic debit cards, making their purpose familiar to many.

However, their use of distributed consensus algorithms make them considerably slower and less deterministic than debit card transfers [11], causing a considerable paradigmatic divergence. Also, the architecture and implementation we present in this paper provide a *general-purpose* system for negotiating digital ownership exchanges, which should mean that our solutions can facilitate more kinds of use cases.

2.2 Integration of Legal Prose

The Ethereum blockchain system claims to support *Smart Contracts* [7], which were originally described by N. Szabo in [15]. In the case of Ethereum, such a contract is a set of computer instructions that are executed as decided by the majority vote of the Ethereum network. A significant limitation of such a system, however, is that the Ethereum majority wields no other power than deciding what to append to the immutable ledger it maintains. If anything bound to an external domain, such as the physical world, would be out of order, the majority is unable to mitigate it without assistance. While national legal institutions are available for trying and correcting contractual deviations in other human domains, it is currently unclear how and if such institutions will consider smart contracts and their ledgers as evidence [10].

To improve compatibility with existing legal instances, some distributed ledger systems allow for the association of legal prose with their code-as-contracts. That kind of hybrid contracts are sometimes referred to as *Ricardian Contracts*, a term introduced by I. Grigg in [16]. For example, R3 Corda [9] allows its *state objects*, which are consumed and created via transactions, to refer to both legal prose and contract code. Such state objects are exchanged in accordance with predefined patterns, referred to as *flows*, which are defined in a programming language. Another related example is the *Ergo* programming language of the *Accord project* [17]. The language facilitates the creation of logical and legal contracts that can be executed by smart contract systems, such as Hyperledger Fabric [8].

[1]As *Script* is not specified in the Bitcoin paper, interested readers might want to consult [13], which contains a formal description of the language. Note that we consider *Script* too limited to be a code-as-contracts system.

However, referencing or integrating legal prose is not the same as making code-as-contracts optional. Both of these systems rely on domain models in which the state of a distributed computer is updated through the execution of functions. Consequently, the programming of states and functions becomes critical to the formulation of contracts. Further, ensuring a contract is legally compatible should require that each computer state can be mapped to legal rights and obligations in such a way that a given institution can know the standing of each party.

In contrast, the architecture we propose in Section 4.1 relies on a domain model of negotiated ownership exchanges. Consequently, defining the implications of token ownership becomes the primary concern of a contract maker, not the programming of states or functions. As each token ownership symbolizes a set of rights and obligations, determining the legal standing of each relevant party becomes an exercise of determining the ownership history of each relevant token. For certain tokens, only knowledge of the current owner may be enough. While our domain model perhaps could be extended by superimposing state machines or any other code-as-contract capabilities, it is significant that our system *can* be used without such capabilities. We assume that a simpler system stands a better chance of being tried successfully in national courts of law and receive business adoption, for which reason it we deem it relevant to focus primarily on critical functionality, however useful other features may be.

2.3 No Distributed Consensus

While the distributed consensus algorithms often employed by blockchain systems may make them practically resilient to certain kinds of attacks, it also leads to them exposing contracts and transactions to network participants not directly concerned [18]. Additionally, distributed consensus can be a time or resource consuming process, having significant impact on transactions throughput and latency [11].

Instead of trying to mitigate these issues directly, R3 Corda avoids them by not requiring a distributed consensus algorithm to be used by normal nodes [9]. Consensus is reached either between pairs of peer nodes, which does not necessitate the use complex algorithms, or within pools of *notary nodes*, which do have to use distributed consensus algorithms. Notary pools are tasked, as requested by normal nodes, to ensure *state objects* cannot be consumed twice, which ensures that transferable assets cannot be duplicated by sending them to more than one recipient, among other things. As the notaries only see the cryptographic hashes [19] of the state objects they are given, they are unable to inspect the data of the state objects they validate. Note that when not dealing with assets that can be transferred multiple times, notary nodes do not have to be used at all.

The system design we propose in Section 4 relies on the same fundamental approach to consensus as R3 Corda, by which we imply that interactions do not need to be relayed through a network of reviewing voters, and that ledgers are replicated only between pairs of nodes. While perhaps our implementation could have been built on top of R3 Corda as a so-called *CorDapp* [9], or as a modification and extension of the R3 Corda code base, the from-scratch implementation we present in Section 4.2 helped us avoid both the complexities of a layered design and the considerable task of removing and replacing Corda's code-as-contracts cababilities.

3 Contractual Cooperation

To be explicit about what process we attempt to digitize, we here present a sort of specification for verifying our and any other digital cooperation solutions against. We first present an informal *characterization*, intended to capture elements key to contractual cooperation, after which we rephrase our characterization as a list of qualitative *requirements*.

3.1 Characterization

- *What is contractual cooperation?* We think of it as free agents coming together in a *joint undertaking*, where each agent is incentivized to collaborate by it somehow contributing to the fulfillment of the agent's *own* goals or ambitions. In other words, contractual cooperation takes place in a setting where different parties take advantage of each other's capabilities for the sake of promoting the realization of their own distinct ends, which may or may not be conflicting. The undertaking is formalized by having each involved agent accept a contract, which states the terms relevant to the enterprise and the roles of each participant.

- *What problems characterize such cooperation?* More than any other, we believe it to be *uncertainty about the incentives of the counter-parties.* Cooperation takes place in a volatile world where circumstances change, suddenly or gradually. New competitors emerge, laws change, and trends shift, which could make prices drop, new markets become available, or existing businesses unprofitable. Such events might make any counter-parties want to discontinue their involvement or change their terms. Other potential problems may include counter-parties being or becoming fraudulent, incapable of fulfilling their roles, unaware of key limitations, or leaking sensitive facts to competitors.

- *How are those problems mitigated?* Through the use of different kinds of *risk aversion strategies.* A common such is ensuring misbehaving parties can be *compelled* to conform or compensate its counter-parties, which can be accomplished by agreeing on an adjudicator when a contract is accepted. Examples of such adjudicators could include courts of law, private arbitration firms or member councils. A related strategy is to *prevent* or *disincentivize* misbehavior by letting trusted third parties act as mediators, controlling sensitive exchanges or other interactions. The same strategy could also be used in the context of blockchain systems or other distributed ledgers, in which case the voting majority of the system become the mediator. This requires, however, that the majority carriers enough power to prevent all relevant kinds of misbehavior, as we note may be problematic to facilitate in Section 2.2. Other possible strategies could involve the continual assessment of trustworthiness, the use of insurance agreements, or the concealment of significant facts from counter-parties and competitors, which otherwise would be able to use that information to the detriment of the concealing party.

3.2 Requirements

1. *Provable acceptances.* Being able to prove that an agreement has taken place gives each party some power over its counter-parties, in the case they would break the terms of that agreement. Those proofs could, for example, be used in a court of law, or be shown to others to deter them from collaborating with any offenders. Note that with the term *acceptance*, we refer not only to the signing of a contract, but to any interaction where the contractual standings of two or more parties change. Such changes in standing may occur when contractual clauses are fulfilled, such as by delivering a good.

2. *Renegotiable terms.* As circumstances can change after a collaboration has been formalized, there has to be room for its terms to be renegotiated. Ideally, such a renegotiation would not involve having to formally cancel and restart an undertaking, as it could lead to accountability issues or costly delays. Being able to (1) amend contracts and (2) make contractual exceptions could be important tools for facilitating renegotiations.

3. *Effective adjudication.* Having access to adjudication is paramount both because it could allow disputes to be resolved and because it could deter the cooperating parties from deviating from their contract. However, it requires that the adjudicator, whether it be a court of law or anything else, is able to (1) access, interpret and consider relevant evidence, as well as (2) compel the party deemed at fault to make reparations. This may require that a given agreement is considered lawful by the adjudicator, which could, for example, include proof that a voluntary offer and accept has taken place [10]. If the adjudicator is a computer system, that system must be able to access whatever power is required to enforce its judgements.

4. *Consistent interpretation.* There must be some kind of framework in place that guarantees that each contractual partner interprets the terms of their contract the same way, especially considering what right and obligations are associated with each party at every given instance. This is very much related to having access to effective adjudication, as the adjudicator could be asked to judge if interpretations differ. However, it also stresses the need for any contractual partners to agree on a well-defined legal vocabulary when a contract is initially formulated and mutually accepted.

5. *Trustworthy identification.* Contractual partners have to be able to reliably determine if any received request originated with their counter-parties or not, as means of avoiding fraud by malicious third parties. Additionally, adjudicators need to be able to assert that signatures, or other attestations, are authentic and refer to legally relevant entities. In a technical setting, this may involve mapping legal entities to cryptographic primitives, such as public keys.

6. *Interactional privacy.* Knowledge about the activities of one's competitors is a significant means to improve the effectiveness of one's own business strategies. This means that ensuring contracts and cooperations remain concealed from competitors can be key to preventing that information from affecting their competitiveness. It could, for example, involve the obligation of each party to not reveal certain facts about a contract, or the use of cryptographic means to ensure messages remain obscure to potential observers.

As much as we would like to be able to show that this list of requirements is complete, we suspect it serves better as a starting-point than a complete ontological definition. For example, we do not consider the economics of participation. If the cost of taking part in a cooperation system would be too high, the incentives for participating would be defeated. However, including that requirement would demand that we determine whether or not the concepts we consider do fulfill it, which we found no reasonable way of doing. Other requirements of relevance could include message integrity guarantees, or having reasonably synchronized clocks.

4 The Exchange Network

Having presented what contractual cooperation is, as well as key requirements for such a cooperation to be meaningfully executed, we are now ready to describe how we believe the process could be made digital. Concretely, we first outline an implementation-independent architecture, then describe one way to implement it, and then, finally, describe a simplified use case, indented to demonstrate the utility of the architecture and implementation design.[2]

4.1 Architecture

An *Exchange Network*[3] (EN) is a monolithic or distributed application, facilitating a digital marketplace where well-known types of assets can be negotiated about, exchanged, and proven to have been part of past exchanges. Concretely, an EN facilitates coordinated changes of the designated owners of electronic *tokens*, which could be thought of as symbolizing certain rights or obligations, such as the right of ownership, the obligation to render a service, or the right to receive payment for the fulfillment of a task. The architecture consists of four primary components, shown in Figure 1.

[2]We have already presented both the architecture and the implementation in a previous conference paper [20]. That paper is primarily concerned with the abstract negotiation protocol of the architecture, as well as considering different ways in which the architecture can be implemented. In this paper, however, we instead focus on the digitization of contractual cooperation, provide a much more extensive description of how we use the signature chain data structure and our implementation, as well as consider how our implementation and other distributed ledger solutions comply with our understanding of the current contractual paradigm.

[3]Inspired by the term *social network*, we chose the name *Exchange Network* for our architecture. The name is intended to invoke the idea of an ever-changing network of interacting actors, primarily concerned with the negotiation and exchange of goods, services, or other values.

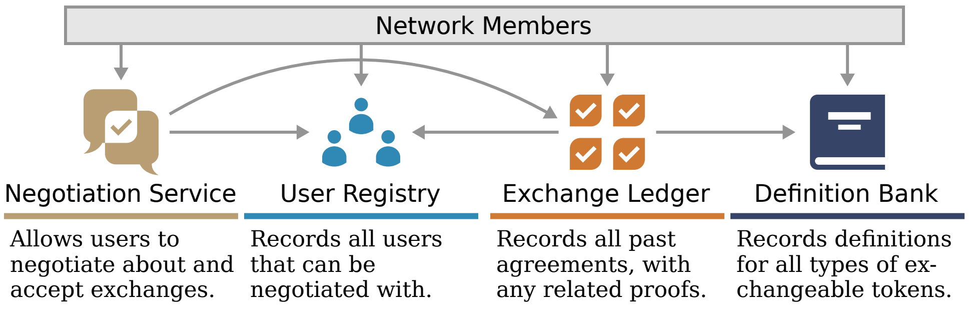

Network Members
Negotiation Service
Allows users to negotiate about and accept exchanges.
User Registry
Records all users that can be negotiated with.
Exchange Ledger
Records all past agreements, with any related proofs.
Definition Bank
Records definitions for all types of exchangeable tokens.

Figure 1: EN components. Arrows point from the using to the used component, which, for example, means that the Negotiation Service makes use of both the User Registry and Exchange Ledger components. Each component also provides some kind of interface for eligible network members.

Before presenting each of those components in turn, we want to stress that we make no assumptions about how they store data or coordinate user interactions, as long as data can be accessed and members interact. The components fulfill abstract functions that can be realized in multiple ways. However, we only describe one way of implementing the architecture in this paper, which is intended demonstrate a non-disruptive approach to designing systems for digital collaboration. In [20], we also consider how a blockchain system like Hyperledger Fabric [8], or a common database system like MySQL [21], could be used as implementation foundations.

Negotiation Service

The *Negotiation Service* (NS) allows the members of an EN to propose, accept and reject exchanges of tokens. It relays *proposals* between pairs of negotiating parties, which take turn in trying to formulate a proposal both deem acceptable. If such an acceptable proposal can be identified by those parties, the NS submits it to the *Exchange Ledger* (EL) component, which makes sure it can be proven to have taken place to any third parties relevance, such as courts of law, insurance agencies, lenders, partners, and so on. A negotiation is a procedure of three phases, (1) *qualification*, (2) *acceptance* and (3) *finalization*, depicted as a state machine in Figure 2.

1. *Qualification.* The first objective of a negotiation is to find a *qualified* proposal believed to be acceptable to each party. A qualified proposal is such that leaves no room for ambiguity regarding who would have what rights and obligations if the proposal would be accepted. The proposal is searched for by having the negotiating members take turn in trying to formulate it. If not enough information is had for a candidate proposal to be qualified, an unqualified such may be used instead. Unqualified proposals may refer to abstract types of tokens, include choices, or identify undesired tokens. To facilitate the communication required to send these proposals, the *Proposal* message in Figure 3 is provided.

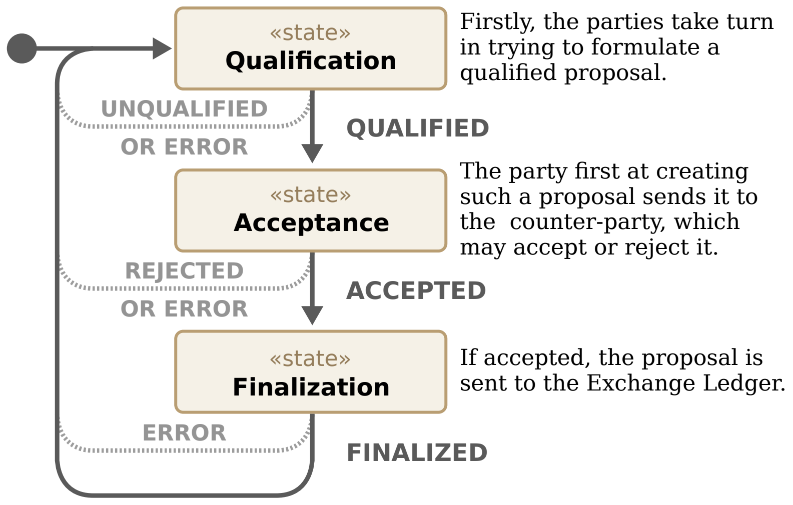

«state»
Qualification
Firstly, the parties take turn in trying to formulate a qualified proposal.
UNQUALIFIED OR ERROR
QUALIFIED
«state»
Acceptance
The party first at creating such a proposal sends it to the counter-party, which may accept or reject it.
REJECTED OR ERROR
ACCEPTED
«state»
Finalization
If accepted, the proposal is sent to the Exchange Ledger.
ERROR
FINALIZED

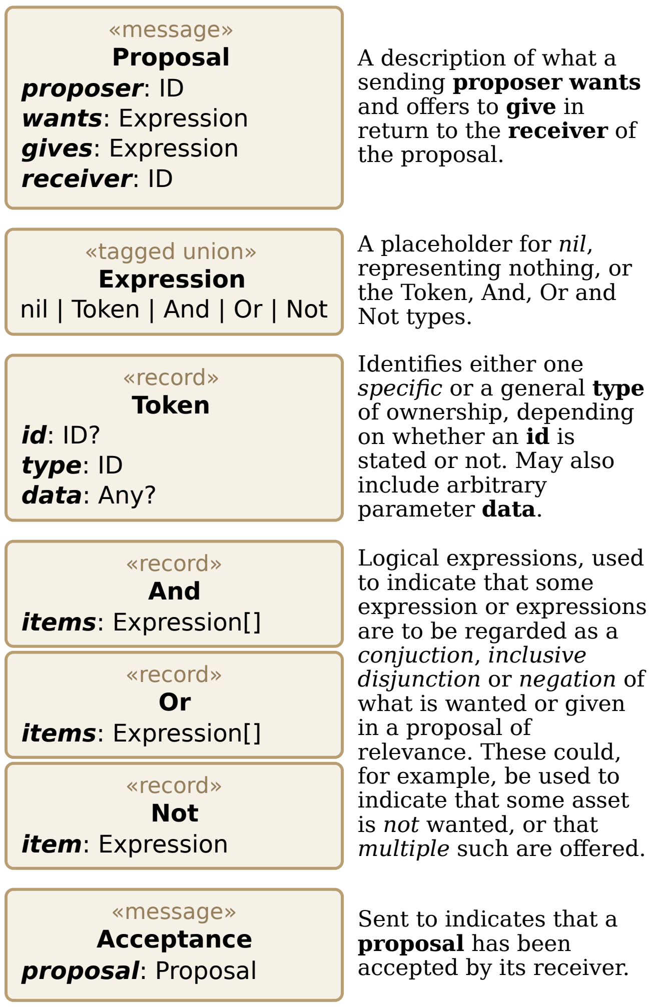
Figure 2: A naive state machine, illustrating how two negotiating parties could progress from an initial proposal to an accepted and finalized such. A negotiation can be terminated at any time by either of its participants.

2. *Acceptance.* As soon as one party formulates a qualified proposal, the objective becomes to determine if the counter-party deems it acceptable. After having sent the qualified proposal, the counter-party either rejects it by sending a new counter-proposal, or accepts it using the *Acceptance* message in Figure 3. If rejected, the negotiation returns to the **Qualification** phase.

3. *Finalization.* When a qualified proposal has been both formulated and accepted, it is submitted by the NS to the EL. The parties are notified when it is known whether that submission succeeded or failed, after which the negotiation returns to the **Qualification** phase. If there is more to negotiate about, negotiation continues. In any other case, the parties are free to terminate the negotiation session.

User Registry

The *User Registry* (UR) is responsible for associating the the *internal* identity of each EN member with its *external* identities. An internal identity is an identifier used to refer to an EN member within the system, such as in proposals, acceptances or exchanges. External identities, on the other hand, is what allows members to recognize other members outside the bounds of the EN. In whatever manner a given UR component is implemented, be it a database of x.509 certificates integrating with some public-key infrastructure [22] or something completely different, it must be able to guarantee that the identities of all members are trustworthy.



A description of what a sending **proposer wants** and offers to **give** in return to the **receiver** of the proposal.

A placeholder for *nil*, representing nothing, or the Token, And, Or and Not types.

Identifies either one *specific* or a general **type** of ownership, depending on whether an **id** is stated or not. May also include arbitrary parameter **data**.

Logical expressions, used to indicate that some expression or expressions are to be regarded as a *conjuction*, *inclusive disjunction* or *negation* of what is wanted or given in a proposal of relevance. These could, for example, be used to indicate that some asset is *not* wanted, or that *multiple* such are offered.

Sent to indicates that a **proposal** has been accepted by its receiver.

Figure 3: The ***Proposal*** *and* ***Acceptance*** *messages, with associated data types.* ***ID*** *represents an arbitrary identifier type, question marks (?) are used for optional values, while brackets ([]) are used to denote array types. The types and fields are a minimally viable set of such, not all that would be useful.*

Exchange Ledger

The EL conceptually maintains an append-only ledger of **Exchange** records, each of which consist of an *Acceptance*, as depicted in Figure 3, and any other data of relevance. As a consequence, the EL can be used by EN members to (1) determine if proposed or already finalized ownership exchanges are *sound*, and (2) prove that past ownership exchanges have taken place. Soundness can be determined by ensuring the tokens of a proposal adhere to their *tests*, which may include taking historic exchanges of relevance into account. Soundness is described further in Section 4.1. The EN architecture makes no assumptions about how past ownership exchanges are proven to have taken place, as long as they can be. However, we consider one concrete way such proofs can be facilitated when we consider our implementation in Section 4.2.

Definition Bank

The last component, the DB, defines the implications of owning or creating every known type of token, especially in terms of what may be done with them and any entities they represent. A DB could be regarded as a dictionary, allowing EN members to look up **Definitions**, as defined in Figure 4, by their names, hashes, or other identifiers.

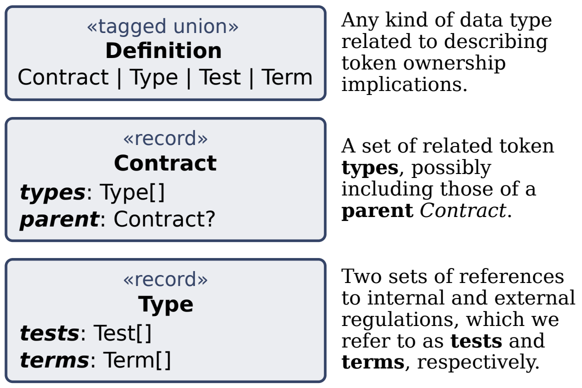

«tagged union»
Definition
Contract | Type | Test | Term
Any kind of data type related to describing token ownership implications.
«record»
Contract
types: Type[]
parent: Contract?
A set of related token types, possibly including those of a parent Contract.
«record»
Type
tests: Test[]
terms: Term[]
Two sets of references to internal and external regulations, which we refer to as tests and terms, respectively.

Figure 4: Some proposed kinds of DB definitions. Our naive ***Contract*** *contains only* ***Types****, implying it could be useful for direct machine-verification of contractual events if any of those* ***Types*** *would refer to* ***Tests****.*

Concretely, the purpose of maintaining definitions is to ensure *soundness* can be verified for proposed and finalized exchanges. Soundness is established by asserting that a given exchange adheres to both internal and external *regulations*.

- *Internal Regulation.* These regulations, which we also refer to as *tests*, ensure tokens cannot be abused inside an EN. A test could be thought of as a function taking a proposal, an EL and a DB as arguments, returning *true* only if the proposal is sound. For example, tests could limit the number of times a certain type of token can change owner, restrict creation or ownership of specific tokens to a fixed set of eligible members, or set expiration dates after which some tokens may no longer be exchanged. In other words, they could be used to prevent some unsound ownership exchanges from taking place at all.
- *External Regulation.* These exist to ensure any entities represented by any EN tokens are not abused outside the bounds of the EN. We refer to these regulations as *contractual terms*, or just *terms*, and they may or may not be machine-readable. For example, let us assume two EN members have exchanged one token representing the right to a vehicle repair for another representing a promise of payment. At this point, there is no way for the EN itself to determine if any vehicle is repaired or any payment is made, as these events happen outside the computers of the EN. This could be mitigated by ensuring the types referenced by the exchanged tokens contain contractual terms, in the form of legal prose, honored by some legal authority. As long as the finalized exchange itself counts as proof, an appeal could be made to that authority to resolve any disputes.

4.2 Implementation

While possible to implement an EN in many different ways, we were particularly interested in doing so such that the existing contractual paradigm could be preserved, as far as reasonably possible. For this reason our implementation does not use any kind of distributed consensus algorithm, and neither does it yet support code-as-contracts, even if we define machine-executable verification functions, or *tests*, as part of the architecture in Section 4.1. Concretely, the implementation is intended to mimic the way common paper contracts and other forms of signed instruments are used, which are known only to two or more parties until the event of a dispute, in which case those instruments are revealed to a court of law or other adjudicator.

We begin the description of our implementation by giving an overview of its design, including its user and application interfaces, after which we describe the data structure it uses to construct nonrepudiable ledgers, and, finally, present a limited scenario that our implementation can run.

Design Overview

Our system consists of a *node* both serving a web client and communicating with other nodes, as depicted in Figure 5.[4]

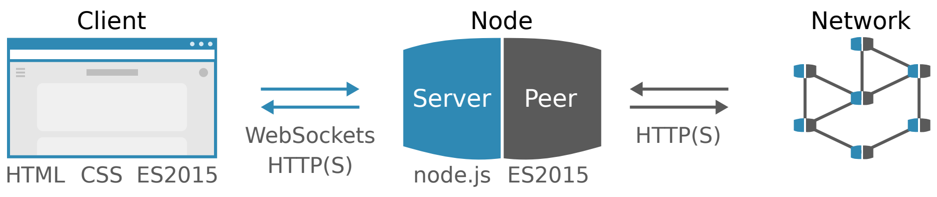


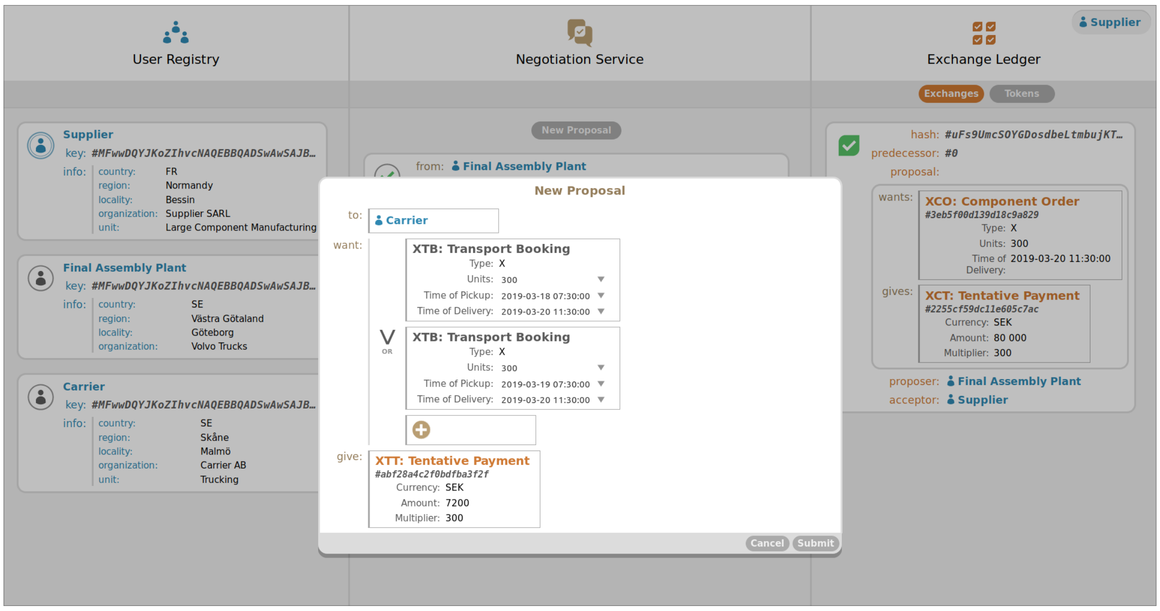
Figure 5: The general design of our EN implementation. Humans operate each Node using a web browser Client. Every Node uses an internal Server to both provide its Client with static HTTP(S) [23] resources and send runtime data via WebSockets [24]. Each Node also contains a Peer module, which is used to communicate with the Peer modules of other Nodes over HTTP(S). The design requires no central data repository or any centralization of control.

The business logic of both the node and the client it serves are programmed in the TypeScript programming language [25], which compiles to ECMAScript 2015 [26], also referred to as JavaScript (JS), before execution. The JS of the node is executed by the node.js runtime [27]. The visual structure, styling of the client application are defined using HTML [28] and CSS [29], respectively, and must be executed via a web browser complying to the referenced standards.[5]

[4] Available at *https://github.com/emanuelpalm/en-signature-chains-poc*. The paper describes commit `694e3a73a1fbae67b9c106d47bd5a1`.

[5] We used these technologies and standards mostly because they are familiar to us. There are no inherent reasons why these should be technically superior to any other particular sets of technologies.

The client, shown in Figure 6, divides it user interface into three columns, (1) User Registry, (2) Negotiation Service and (3) Exchange Ledger, which correspond to three of the EN components. The first column lists trusted user identities, the second column allows sending and accepting proposals to and from other users, while the third column lists all known finalized ownership exchanges. A template token system is provided as a form of naive DB component, which ensures that created tokens follow configurable rules.



Supplier
key: *#MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJB…*
info: country: FR
region: Normandy
locality: Bessin
organization: Supplier SARL
unit: Large Component Manufacturing

Final Assembly Plant
key: *#MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJB…*
info: country: SE
region: Västra Götaland
locality: Göteborg
organization: Volvo Trucks

Carrier
key: *#MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJB…*
info: country: SE
region: Skåne
locality: Malmö
organization: Carrier AB
unit: Trucking

from: **Final Assembly Plant**

New Proposal
to: **Carrier**
want:
XTB: Transport Booking
Type: X
Units: 300
Time of Pickup: 2019-03-18 07:30:00
Time of Delivery: 2019-03-20 11:30:00

V OR

XTB: Transport Booking
Type: X
Units: 300
Time of Pickup: 2019-03-19 07:30:00
Time of Delivery: 2019-03-20 11:30:00

give:
XTT: Tentative Payment
#abf28a4c2f0bdfba3f2f
Currency: SEK
Amount: 7200
Multiplier: 300

hash: *#uFs9UmcSOYGDosdbeLtmbujKT…*
predecessor: *#0*
proposal:
wants: **XCO: Component Order**
#3eb5f00d139d18c9a829
Type: X
Units: 300
Time of Delivery: 2019-03-20 11:30:00
gives: **XCT: Tentative Payment**
#2255cf59dc11e605c7ac
Currency: SEK
Amount: 80 000
Multiplier: 300
proposer: **Final Assembly Plant**
acceptor: **Supplier**

Figure 6: The three columns of the client user interface, behind a dialog in which a new proposal is formulated. The formulated proposal is a request to a carrier to transport 300 components for 7200 SEK per component, while giving the carrier the option of choosing between two pick-up dates. The screenshot is taken from the demo application in the implementation code repository. See Footnote 4 on page 122 for details on how to access the code and instructions for running the demo. The scenario it illustrates is described in Section 4.2.

While human users communicate with their nodes using web clients, the nodes themselves communicate with each other by sending HTTP(S) requests to the endpoints outlined in Table 1, which each node is expected to expose.

Table 1: HTTP(S) endpoints exposed by nodes via their Peer interfaces.

Method	*Path*	*Description*
POST	/**proposals**	Submit a signed exchange **Proposal**.
POST	/**acceptances**	Submit a signed and *qualified* proposal **Acceptance**.
POST	/**exchanges**	Share a known previously finalized **Exchange**.

In particular, the node is designed such that if a sent proposal includes a token containing a reference to a previous exchange, it will automatically send that exchange before the proposal. This means that finalized exchanges can be distributed to third parties as proof of something having been accepted. This is utilized in the example use case in Section 4.2 by a carrier, allowing it to prove to its client that a delivery was accepted by its recipient.

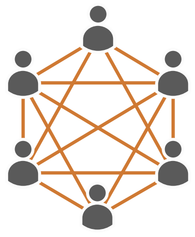
Our implementation exists to fulfill three functions, (1) to force us to confront and reevaluate the EN and SC concepts during its development, (2) to confirm that our concepts are complete enough to be implemented, as well as (3) to give use a tangible artifact to demonstrate to industry experts in order to receive relevant feedback. None of these three functions require having a production ready system, for which reason we made some important delimitations to reduce implementation effort. For example, any UR and DB data must be provided at node startup, and cannot be changed or added to during runtime. Even though messages between peers are signed and verified cryptographically against known users, no communication transports are encrypted. Client users are not authenticated or authorized by their nodes. Finally, while HTTP endpoints are provided for sending proposals, acceptances and exchanges, there are no such for requesting exchanges, definitions, users or other relevant data.

Signature Chains

To ensure that finalized exchanges can be proven to have taken place, our implementation relies on data structures that use signatures and hashes in a way comparable to blockchains. However, as they do not gather records in batches, we instead refer to them by the name *Signature Chains* (SCs). Concretely, an SC consists of a chain of records, each of which *may* refer to (1) a previous record and (2) a definition of relevance. Each record is cryptographically signed by one or more *attestors*, in our case a proposer and acceptor, and any references to records or definitions are the cryptographic hashes of the data referred to [19]. By implication, a third party given a chain of records, with any associated definitions, becomes able to verify that the records

1. indeed have been signed by their attestors,
2. were created in a certain order, and
3. always have referred to the provided definitions.

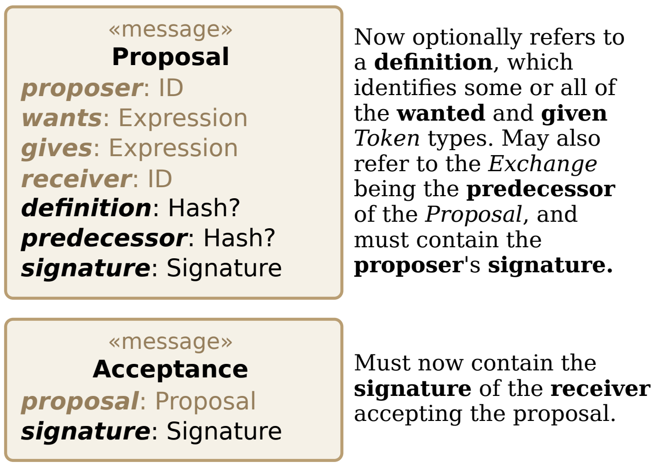
Rather than SCs being stored in a centralized or replicated repository, each possible pair of EN members may maintain their own sets of chains, as depicted in Figure 7. This leaves room for every such pair of members to maintain privacy, given that they can agree on not sharing their mutual records with others. By implication, it also means that both members of each pair can independently reveal any shared chain to any party of interest, such as a court of law, partner or other party.



In Signature Chain ENs, each possible member pair may maintain its own set of SCs. Each such potential set is here denoted by a line connecting two members.

Figure 7: The potential sets of SCs, or ledgers, in a six user EN.

To concretely implement the SC data structure, we amend the **Proposal** and **Acceptance** types as described in Figure 8.



«message»
Proposal
proposer: ID
wants: Expression
gives: Expression
receiver: ID
definition: Hash?
predecessor: Hash?
signature: Signature

Now optionally refers to a **definition**, which identifies some or all of the **wanted** and **given** *Token* types. May also refer to the *Exchange* being the **predecessor** of the *Proposal*, and must contain the **proposer**'s **signature.**

«message»
Acceptance
proposal: Proposal
signature: Signature

Must now contain the **signature** of the **receiver** accepting the proposal.

Figure 8: Amended variants of messages first outlined in Figure 3.

As our particular implementation does not provide a DN component, but a simpler template system assuming tokens are defined elsewhere, the **definition** field of all **Proposals** is always empty. This limitation is not critical for demonstration purposes, as in Section 4.2, as we can assume that the set of used templates, and their associated legal interpretations, are known beforehand by all participants.

In the case of a fully implemented DB component being available, however, then all definitions ought to refer to their subdefinitions via their hashes. This would guarantee that those definitions cannot be modified without it being detectable. An example of a SC with definition references is illustrated in Figure 9.

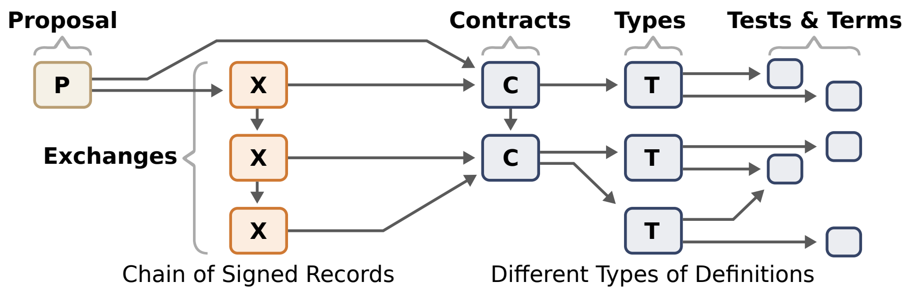


Figure 9: An example SC. Arrows denote references by hash. Assuming that proposals and exchanges are signed, any party trusting the identity of the signatures can verify that any associated definitions are not modified since the proposal or exchange was signed. This requires, however, that the verifying party can access referenced exchanges and definitions.

SCs are created or extended through a procedure of six step, beginning at the point where a **Proposal** is formulated by some party A that will subsequently be accepted by its receiver B. We describe the steps in Figure 10 and below.

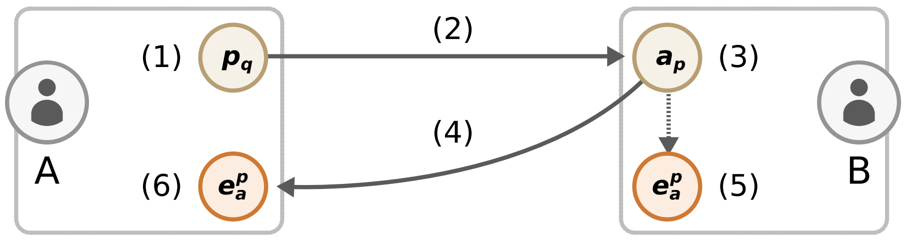


Figure 10: The six steps, taken by A and B, from the creation of a qualified proposal to its acceptance and finalization. Circles denote artifact creation, solid arrows artifact transmission and the dashed arrow an internal transition. Both A and B must cryptographically verify the artifacts they receive.

1. A creates and signs the qualified **Proposal** (p_q).
2. A sends p_q to B.
3. B creates and signs the **Acceptance** (a_p) from p_q.
4. B sends a_p to A.
5. B puts a_p and its hash into an **Exchange** (e_a^p).
6. A puts a_p and its hash into an **Exchange** (e_a^p).

While a benefit to this procedure may be its lack of complexity, it does have the following potential weaknesses.

- *Acceptance asymmetry.* A is unaware that B accepted and signed p_q unless step (4) is completed successfully. If B is malicious, it could perhaps be advantageous for B to create a_p while concealing it from A.
- *Clock divergence.* In certain scenarios, a p_q may have to be accepted within a certain time window. As each party has its own clock, there could be diverging opinions on whether or not an acceptance (a_p) is timely.
- *Exchange malleability.* While p_q and a_p are signed, e^p_a is not. This gives room for both A and B to record whatever private data they want to associate with a_p in e^p_a, but it also means that if e^p_a is distributed, any fields other than a_p are malleable without it being noticeable.
- *Token duplicability.* In cases where tokens needs to be transferable multiple times, it becomes possible for a malicious party to transfer the same token to multiple counter-parties without it being immediately detectable. As information only is shared as strictly needed, it will seem to both as if they now become the legitimate owners of the new token, which, for example, could be tied to the ownership of a physical good.

All of these weaknesses can be countered, however, through different uses of trusted third parties. And if trusting a single third party would be an issue, for reasons such as concerns about trustworthiness, privacy or fault-tolerance, voting networks of third parties could be a viable alternative.[6] For a concrete example of how a single trusted third party could be used, consider the procedure in Figure 11.

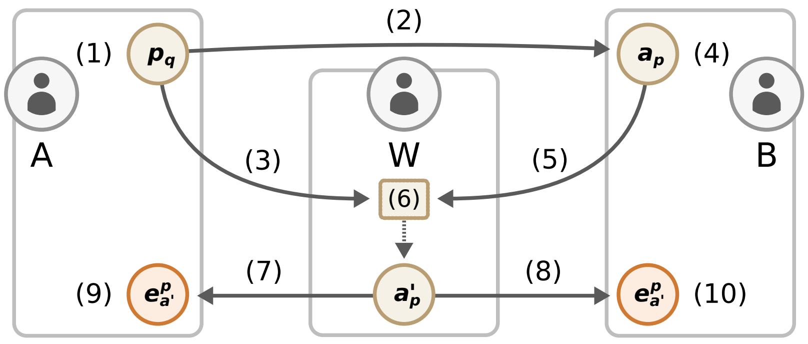


Figure 11: A variant of the steps described in Figure 10. Here, a trusted third party W is provided with the qualified proposal (p_q) in step (3) *and the acceptance (a_p) in step* (5)*. If a_p contains p_q and seems to be valid, W creates and signs a'_p in step* (6) *and then sends it to both A and B.*

A trusted third party (W) could assert the timeliness of the acceptance (a_p), ensure both A and B receive the signed acceptance (a'_p), as well as remember whether or not a token part of a proposal has been transferred before.

[6]This is effectively what R3 Corda achieves via their *notary pools* [9] while preserving a significant level of privacy, as described in Section 2.3.

One or more parties being witnesses of an exchange could also have desirable contractual implications. For example, an insurance agency being a witness, and thereby be given the opportunity to ratify an exchange, could be used as a way of ensuring the insurance agency remains commited to a prior insurance agreement. Another example could be a situation in which some token is owned by multiple parties. All parties except for the one initiating the transfer of the token could be called upon as witnesses to ensure they all ratify it being exchanged.

The simplicity of the basic exchange finalization protocol, which we described in Figure 10, means that it lends itself to many kinds of extensions. One example of such an *extended* exchange finalization protocol is given in Figure 11, where a third party ratifies the exchanges of two other parties. We recognize that many such protocols could be identified for scenarios with different requirements on privacy, robustness, trust, and so on. However, we leave their identification as a subject for future work.

Example Use Case

As our implementation was designed to demonstrate the EN and SC concepts, it comes with a set of files for running an example use case.[7] We here proceed to describe that example scenario, as it gives another perspective on how our implementation is designed to work. The example consists of six interactions between three partners, as described below and in Figure 12, using the tokens in Figure 13.

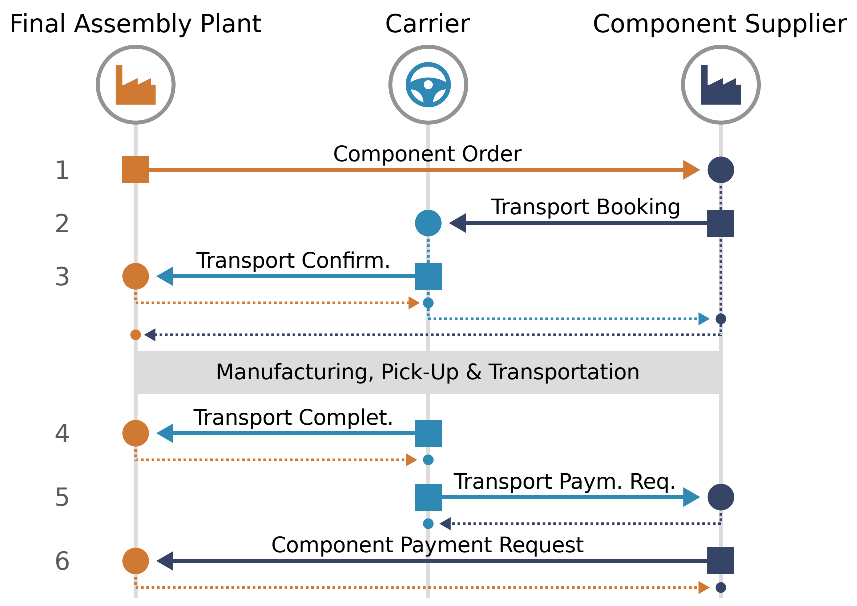


Figure 12: The six steps of the example use case. Solid arrows represent sent proposals, while dotted arrows represent sent proposal acceptances.

[7]See Footnote 4 on page 122 for a link to the source code repository, which also contains detailed instructions for running the demo.

Token ***type***: "xco" ***data***: "**type**": "X" "**units**": *Integer* "**time of delivery**": *Date*	**Component Order** Symbolizes the commitment of its giver to deliver a certain number of units of some component type at a certain date and time. The delivery address is assumed.
Token ***type***: "xct" / "xtt" ***data***: "**currency**": "SEK" "**amount**": *Integer* "**multiplier**": *Integer*	**Tentative Payment** Represents the obligation of its giver to pay a certain amount in return for the completion of the task specified in the exchange the token was created.
Token ***type***: "xtb" ***data***: "**type**": "X" "**units**": *Integer* "**time of pickup**": *Date* "**time of delivery**": *Date*	**Transport Booking** The giver of this token is bound to be able to accept a transport vehicle at a certain pick-up date and time, which will be ready to transport a certain number of units to an implied destination at a specified delivery date and time.
Token ***type***: "xtc" ***data***: "**type**": "X" "**units**": *Integer* "**time of delivery**": *Date*	**Transport Confirmation** Represents the commitment of the giver of the token to receive a delivery of a certain number of units at a specified time of delivery.
Token ***type***: "xto" ***data***: "**transport**": *#xtc*	**Transport Completion** The giver of the token accepts that the delivery associated with the referenced transport confirmation has been completed.
Token ***type***: "xcp" / "xtp" ***data***: "**payment**": *#xct / #xtt* "**transport**": *#xto*	**Payment Request** The giver of the token accepts to pay, as specified in a referenced tentative payment, as compensation for a referenced transport completion.

Figure 13: Informal definitions of the token types used to facilitate the example use case, with technical descriptions on the left and legal implications on the right. Two of the tokens exist in two type variants each, useful only to allow our implementation determine how to automatically populate certain data fields. Compare with the Token type definition in Figure 3.

The goal of the following interactions is to have certain components manufactured and delivered from a *Supplier* (S), via a *Carrier* (C), to a *Final Assembly Plant* (A).

1. *Component order.* A sends a proposal to S, wanting a component order of 200 units, which are to be delivered at a certain date. In return, A offers a tentative payment of 100 000 SEK per component.

2. *Transport booking.* S sends a proposal to C, wanting a transport booking for the components and the delivery time requested in (1). In return, S offers a tentative payment of 7 800 SEK per transported component.

3. *Transport confirmation.* C sends a proposal to A, in which C requests that A confirms the transportation in (2). When accepted by A, C proceeds to also accept the proposal in (2), and S then accepts the proposal in (1).

4. *Transport Completion.* C sends another proposal to A, wanting A to confirm that the transportation accepted in (3) has been completed, which is then accepted by A.

5. *Transport Payment Request.* C then sends the exchange finalized in (4) together with a proposal of payment to C, in which C refers to the transport completion in (4) and the tentative payment in (2). S accepts.

6. *Component Payment Request.* S, which now knows that the transport has been completed, sends a proposal of payment to A, which refers to the transport completion in (4) and the tentative payment in (1). A accepts.

While the scenario illustrates how ordering, transport and payments could be handled in an industrial scenario, there are a few things we want to note.

- No actual payments were issued or executed by the EN used by the three parties, even if several interactions related to money. The purpose of the EN architecture is to facilitate digital changes to the rights and obligations between partners. It does not move any concrete assets in and of itself, even if events in an EN could trigger other systems to perform such functions. On the other hand, the signed exchanges resulting from the example interactions should be useful as evidence in a court of law, in the case of any party not meeting its obligations, such as by refusing to pay.

- The EN architecture only relays data that is directly related to the rights and obligations of contractual partners. If other information would be of relevance, such as tracker coordinates or digital twins, that would have to be sent via some other system.

- The example most likely contains too few steps to be practical in a real-world setting. Pick-up, quality checks or other significant interactions could likely be of benefit to also negotiate about. The purpose, however, of the example use case is demonstrate how the technology works and how it *could* be used, not necessarily how it *should* be used.

5 Requirements Conformance

In Section 3.2, we listed requirements we believe to be key to meaningful contractual interaction. Here, we qualitatively evaluate how an EN implementation based on SCs, as described in Section 4, could facilitate those requirements. We also consider how R3 Corda [9] fulfill the same requirements,[8] and end the section by identifying differences between the Corda and SC EN systems. A summary of our analyses is outlined in Table 2.

[8]While making the same kind of analysis could be relevant also for other systems, such as Hyperledger Fabric [8], we settle with considering only R3 Corda for two reasons, (1) as Corda uses the same general approach to consensus as the SC implementation, it becomes relevant to clarify their differences, as well as (2) for the sake of limiting the scope of the paper.

Table 2: An overview of how our EN implementation based on SCs and R3 Corda fulfill the requirements we list in Section 3.2. Parentheses are used signify functionality that can only be supported if provided via an external means. We consider the differences between the systems in Section 5.3.

	SC EN	*R3 Corda*
Provable acceptances	Signatures & witnesses	Signatures & notaries
Renegotiable terms	Negotiation Service	(External negotiation)
Effective adjudication	(External adjudicator)	(External adjudicator)
Consistent interpretation	Definition Bank	Code & legal prose
Trustworthy identification	User Registry	x.509 certificates
Interactional privacy	Witnesses optional	Notaries optional

5.1 Signature Chain Exchange Network (SC EN)

Here follows a list corresponding to that in Section 3.2, describing how each of those requirements is fulfilled. As the EN architecture fulfill some of our requirements by itself, we make note of which of the EN or SC concepts is realizing each fulfillment.

1. *Provable acceptances.* The SC data structure facilitates provable acceptances by requiring each such acceptance to be cryptographically signed [19] by its proposer and acceptor. Additional parties can also add their signatures or act as witnesses, given a suitable finalization protocol, as described in Section 4.2. If an acceptance includes hashes of a related earlier acceptance or contractual definitions, no party will be able to deny the history of the acceptance or the commitments associated with it, respectively.
2. *Renegotiable terms.* The NS component of each EN facilitates meaningful negotiation only when there are mutually known token definitions that can be used to formulate proposals. As definitions dictate what can be negotiated about, there could, theoretically, be room for negotiating about further definitions, the creation of new contracts, the amendments of existing contracts, as well as contractual exceptions. Apart from the existence of useful definitions, however, the extent to which agreements can be renegotiated will also depend on the flexibility of the agents driving the negotiations, which could be anything from humans to simpler software.
3. *Effective adjudication.* The EN architecture leaves room for using machine-executable tests, which could be used to *prevent* some unsound exchanges from taking place. Neither ENs or SCs are, however, able to *correct* any contractual misbehavior after it has occurred, which we discuss further in Section 6.3. That being said, the SC data structure is designed to prove that the agreements it records have taken place. That proof, which is based on the cryptographic signatures of well-known entities and immutable references to contractual definitions, should be useful as evidence in traditional courts of law or to private arbitration firms, which must be able to provide such correction. It remains to be seen, however, if things like SCs will be considered by such adjudicators.

4. *Consistent interpretation.* Every EN must provide a DB component, which is intended to maintain definitions of sufficiently rigorous interpretation for negotiation to be practically possible. No particular shape or format is required for any such definitions, even though it may be relevant to support machine-executable validation code and conventional legal prose,[9] as it would enable automatic proposal verification and provide opportunity for adjudication, respectively.
5. *Trustworthy identification.* This problem is meant to be solved by the UR component of all ENs. However, the EN architecture does not explicitly specify how it is to be facilitated. In our SC concept implementation, we used data taken from x.509 certificates [22] to identify parties, which we then manually provided to each node. An implementation for production use would likely need to support distribution of certificates at runtime, handle parties transitioning to new certificates, as well as being able to assess the likelihood of any certificates being compromised. Whether public key infrastructure [22] or some other technology is most suitable for distributing certificates, of any relevant kind, remains to be determined.
6. *Interactional privacy.* We understand complete privacy to be the situation in which only those parties that must know a fact do have direct or indirect knowledge of it. As the EN architecture in itself does not regulate how proposals are relayed, it cannot be decided whether or not it facilitates privacy. Technology such as transport layer security [31] could be used by SC implementations, however, to conceal data in-transit between parties. Also, since using SCs does not require a distributed consensus algorithm, there is not necessarily any voting procedure during which any details about proposals or exchanges could be seen by third parties. As long as information is not leaked intentionally, higher degrees of privacy should be possible.

5.2 R3 Corda

Here follows another list corresponding to that in Section 3.2, this time describing how the requirements outlined there are fulfilled by R3 Corda [9].

1. *Provable acceptances.* In R3 Corda, each contractual interaction results in the creation of a *transaction*, which contains a list of input state objects, commands to apply to those objects, any resulting output state objects, apart from other details such as designated notaries. Each such transaction proves its validity by including at least the signature of its issuer. Depending on the code-as-contracts and flows regulating the transaction, it may also include additional signatures, which could facilitate explicit acceptance of whatever intent is encoded in a given transaction. Pools of notary nodes, which we describe briefly in Section 2.3, can also be used as a form of witnesses.

[9]If common legal prose in digital cooperation systems becomes accepted by most relevant legal institutions and other adjudication instances, it may become relevant to also structure that legal prose in a machine-readable way. It could, for example, enable increased levels of contractual automation. *OASIS LegalRuleML* [30] is one example of such a document structure.

2. *Renegotiable terms.* Corda is not a negotiation system, it is a system for maintaining and updating replicated state machines. In other words, it does not provide any primitives explicitly designed for making or accepting proposals. This implies that the details of any collaboration must to be negotiated outside Corda itself, including how and what can be negotiated about within the confines of a given collaboration. That being said, there could likely be room for implementing a general-purpose negotiation system on top of the primitives Corda does provide.[10]

3. *Effective adjudication.* The fact that Corda maintains a replicated state machine means that it leaves plenty of opportunity for programmatically validating states and state changes, which is useful for *preventing* unsound state transitions from taking place. However, Corda does not in and of itself provide any direct means of *correcting* contractual misbehavior that cannot be prevented. Rather it depends on its code-as-contracts, transactions, immutably referenced legal prose, and any other artifacts, to be accepted as evidence by a court of law or other adjudicator.

4. *Consistent interpretation.* In order to guarantee that code-as-contracts and other machine-readable artifacts are interpreted consistently, Corda comes with its own machine language interpreter. It also leaves room for its transactions to refer to legal prose and other forms of attachments. If such legal prose is formulated in accordance with the norms of some well-established legal tradition, it should be able to facilitate meaningful collaboration.

5. *Trustworthy identification.* To guarantee non-repudiation and unambiguous association of transactions with legal entities, Corda employs x.509 certificates [22]. Further, it *"assumes [the existence of] an identity infrastructure between the participants in the network but makes no assumption as to its sophistication or mode of operation"* [9]. However, if participating in the global Corda network, a network maintained by the R3 organization, the use of a custom infrastructure created by R3 is mandatory.

6. *Interactional privacy.* R3 Corda guarantees privacy by encrypting messages in-transit between parties. Notary pools, when used, are only provided with hashes of state objects, as described briefly in Section 2.3, which means that the contents of any considered objects are not accessible to the notaries in any given pool.

[10] In [20], we consider the possibility of building an Exchange Network on top of a blockchain system such as Hyperledger Fabric [8]. It is not inconceivable that also R3 Corda could be built upon to facilitate general-purpose negotiation, perhaps only using the code-as-contracts capabilities provided by the system. However, providing a programming language and execution runtime is not the same as providing a concrete feature, for which reason we do not consider Corda to directly facilitate renegotiable terms.

5.3 Comparison of SC EN and R3 Corda

Having presented how an SC EN implementation and R3 Corda could fulfill our requirements, it should be apparent that these two systems have significant similarities. For instance,

1. EN SC *witnesses* serve a role similar to Corda *notaries*;
2. the EN DB component could be designed to hold machine-executable verification code and legal prose, which is also provided by Corda; and
3. the EN UR component could be implemented using x.509 certificates and Public Key Infrastructure [22], a variant of which is employed by Corda.

What we consider to be the primary distinction between the two systems, however, is the *domain models* they employ. An EN SC provides a general-purpose negotiation protocol as its most significant coordination tool. Each interaction via that protocol is, conceptually, an attempt to change the state of rights and obligations in a group of collaborating parties. In R3 Corda, on the other hand, the foundational concept is that of cooperating parties jointly maintaining replicated state machines. Consequently, each Corda interaction is an attempt to change the state of such a machine via the invocation of a valid state transition. While there may be ways to model rights and obligations in terms of state machines, as we discuss in Section 6.7, we observe that the choice of the EN domain model has the following consequences.

1. *States can be implicit.* In a Replicated State Machine (RSM) system, such as Corda, any maintained state machines must always be in well-defined states. An EN, on the other hand, is not strictly required to know in and of itself what the current rights and obligations of each cooperating party is, given that the parties can determine it some other way. This means that an EN can operate without any code-as-contracts at all, given that legal prose, or whatever means of keeping track of rights and obligations is used, can be referenced during negotiation.
2. *The possible state space can be unbounded.* As the current state of rights and obligations can be tied to any external artifacts, such as traditional paper contracts, no collaborating EN members are forced to assume that they know the full extent of the possible state space. This leaves room for making contractual amendments and exceptions as needed, rather than requiring that every possible exceptional scenario be predicted before collaboration starts or state machines to be replaced or extended after deployment.
3. *Negotiations can be about negotiations.* In Corda, the details about a collaboration, such as what flows, state objects and legal prose to use, must be negotiated outside the Corda system. In an EN, however, every interaction is negotiational, which means that, given the right set of definitions, an EN system could be used to negotiate about the details of new collaborations.

4. *Rights and obligations must be explicitly accepted.* In RSM systems, like Corda, transactions are only strictly required to be approved by their issuers, even though additional approvals can be demanded. This can be used to design collaborations where individual parties can change the rights or obligations of other parties without their explicit consent. An example of such a transaction could be a transfer of money, which would not have to be accepted by its receiver. While money may be commonly considered a universal good, even owning such may entail obligations not desired by its receiver. In an EN system, every change to a party's rights or obligations must be explicitly accepted by that party.

6 Discussion

We have now considered what contractual cooperation is, key requirements for it to be meaningful, the EN architecture and SC implementation, as well as how our implementation and R3 Corda fulfill our requirements. Here, we discuss the wider implications and weaknesses of our contributions.

6.1 A Minimum-Viable Integration Strategy

While we have described an EN SC implementation and how it ought to deviate less than the state-of-the art from the traditional contractual paradigm, we have not presented any strategy for using it to digitize any existing business collaborations. We do believe, however, that the following five steps could be a suitable starting-point for finding a minimum-viable integration approach.

1. Each party of the collaboration in question installs and configures an EN SC software on a server it manages.

2. Cryptographic certificates are generated by each party, and uploaded to the UR of every server.

3. Legal documents are scanned, hashed and uploaded to the DB of each server.

4. Key interactions defined by the legal documents are identified, and tokens referring to them are specified.

5. A new legal document is signed by each participant, containing their cryptographic certificates.

Essentially, finalized EN exchanges are used as digital receipts referring to the terms of existing paper contracts. If assuming that the EN SC software provides a graphical user interface, no further technical integration would be strictly required. Each party would be able to independently automate any of the interactions in which it takes part.

6.2 The Centrality of Negotiation

In Section 3, we assumed *negotiation* to be the primary activity of interacting collaborators. As this assumption has driven the formulation of the Exchange Network architecture, the utility of the architecture is contingent on the validity of that assumption. All other comparable systems we know of, however, rather assume that collaborations can be modeled as parties updating replicated state machines using complex programming languages. This could be regarded as requiring weaker assumptions, as the programming languages of these systems can be used for building other kinds of protocols. If our assumption is correct about the centrality of negotiation, the additional capabilities and complexity of the state machine systems may be of little relevance in many cases.

6.3 The Prospect of Machine Adjudication

In this paper, we identify distributed consensus algorithms and code-as-contracts as primary obstacles to compatibility with existing adjudicators, such as national courts of law. Those two technologies are, however, what many envision will allow trusted middle-men, among which courts of law are a primary example, to be fully circumvented. However, apart form disrupting current practices, as we already noted, the concrete systems currently using these technologies do have some additional shortcomings. In particular, they are limited in

1. what they can consider as evidence,

2. their power over parties deemed at fault, as well as

3. their ability to take the wider context into account.

In systems like Bitcoin [1] or Ethereum [7], the only unbiased evidence available is signed transactions, the only available punishment is refusing to append transactions to the maintained ledger, and no given data will be considered in terms of its original domain or context. Taken together, this means that these systems are unable to identify or mitigate transactions sent to fraudulent users, for example, even if they are able to detect double-spending, overspending, and so on. Strides of advancement in AI technologies, in areas such as trust assessment, faction mapping and computational ethics, would be required in order to make computer systems approach the capabilities of present day human institutions. For this reason, we assume that traditional adjudicators will remain indispensable for many kinds of use cases in the foreseeable future, given that they adapt to the increasingly digital nature of the disputes they will have to mediate. That being said, however, there may be a plethora of relevant use cases that could become viable with only incremental improvements to existing blockhain systems, which would allow additional trusted middle-men to be circumvented.

6.4 The Economics of System Participation

It seems to us that participating in a digital collaboration system always will require (1) installing, configuring and connecting a node to a network; (2) reliably identifying the user identities of any relevant counter-parties; as well as (3) formulating and distributing contractual definitions. While we noted in Section 3.2 that qualitatively reasoning about the costs of such participation is difficult, we do believe that work to reduce both its costs and its complexity is paramount to ensure the real-world utility of these systems.

6.5 Consensus Speed and Business Impact

In Section 1, we claimed that the performance of distributed consensus algorithms could impact business processes in significant ways. While we showed how this the case for one kind of application in Section 2.1, we expect performance sensitivity to vary greatly from use case to use case. In other words, the value of fast consensus depends on the scenario.

6.6 Risks of Comparing Concepts to Systems

In Section 5.3, we compare our SC EN design with R3 Corda. The former is a concept, proven only by a limited implementation, while the latter is a relatively complete system. When comparing concepts to reality, there is always a risk of idealizing the concept, as its concrete properties cannot be evaluated directly. In order to avoid this, we regard the R3 Corda system as an attempt to realize a concept, and compare the SC EN with that concept instead. Concretely, we base our conclusions primarily on the Corda white-paper [9], rather than technical documentation or source code. That comes with another risk, however, which is relying on a document that, at the time of writing, is about three years old and, therefore, could be outdated. It is not inconceivable that the project has made significant advances to their concept since then, but without formalizing them via a new paper. While the comparison represents our best effort at being objective, we cannot completely rule out any bias introduced by our misunderstanding the Corda white-paper or basing our conclusions on out-dated claims.

6.7 Rights and Obligations as State Machines

Even though we present the Exchange Network concept as facilitating *negotiation about token ownership*, via the use of an abstract message protocol, there is nothing preventing that a computational model be superimposed on that protocol. Concretely, tokens refer to *types*, which in turn refer to *tests* and *terms*, as described in Section 4. One could regard a test as an invariant of a type system, a term as a function body, a type as a full function declaration, and a token as a function invocation. As tokens have room for arbitrary data items, they could even be said to include function arguments. If taking this perspective, which we do briefly in Section 5.3, finalized negotiations result in functions updating a state machine of rights and obligations. Formally defining such a state machine and comparing it to the state-of-the-art is left as a topic for future research.

7 Conclusions

In this paper, we make the case that distributed consensus algorithms and code-as-contracts disrupt common business practices. We also claim that those disruptors can be replaced by non-mediated message passing and a system of negotiated token exchanges. By using cryptographic signatures and hash pointers, akin to how they are used in R3 Corda [9], we ensure messages sent directly between two peers can be proved to authentic later to third parties. By substituting the prevailing finite state machine code-as-contracts model with one of negotiations about ownership exchanges, we change the primary concern of the model from function invocations and state transitions to exchanges of rights and obligations.

We believe that already existing professionals, such as procurement engineers, legal experts and adjudicators, will be able to fruitfully apply the kind of technology we propose with little to no training, given the right kind of supporting software exists. The use of voting to ratify interactions is rare in conventional kinds of collaboration, and writing software is far removed from what most relevant kinds of professionals are accustomed to. However, digital signatures, ownership statements, as well as the other primitives our system design provides, should be perceived as more familiar and, therefore, require less training, as well as requiring fewer adaptations to existing business norms and practices. If the assumptions we make are correct, our approach lowers the barriers to adoption of distributed ledger technologies for businesses, legal institutions and others in comparison to state-of-the-art solutions, such as Hyperledger Fabric [8] or R3 Corda [9].

That being said, there might be compelling use cases that cannot be facilitated by our approach. For example, solutions such as Ethereum [7] are able to facilitate code-controlled agents via a public and global process reminiscent of voting, which can be used to circumvent traditional third parties in certain situations. Our current understanding is that nothing similar could be achieved with the system design we propose, at least without extending it to also support defining the behavior of and facilitating such agents. However, the practical utility of such agents has proven very limited, as we discuss in Section 6.3, which means that they cannot replace most interesting third parties to an adequate degree, such as inspection firms, courts of law and so on.

In Section 3.1, we characterize cooperation as *being* the process of collaborating parties continuously renegotiating their current sets of rights and obligations, while never being absolutely sure about the aims and incentives of their counter-parties. If nothing else, we believe this paper should establish that digital cooperation systems must be able to represent the contentious nature of this process to remain useful over time.

Acknowledgments

We would like to thank Richard Hedman at Volvo Group for his insights regarding industrial supply chains. We would also like to thank Caroline Berg von Linde, Johan Hörmark, Christian Lagerkvist and Jamie Walters at SEB for helping us better understand the utility of trusted middle-men.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, accessed 2019-02-08. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[2] M. E. Peck, "Blockchains: How they work and why they'll change the world," *IEEE Spectrum*, vol. 54, no. 10, pp. 26–35, 2017.

[3] S. S. Arumugam, V. Umashankar, N. C. Narendra, R. Badrinath, A. P. Mujumdar, J. Holler, and A. Hernandez, "IoT enabled smart logistics using smart contracts," in *2018 8th International Conference on Logistics, Informatics and Service Sciences (LISS)*. IEEE, August 2018, pp. 1–6. [Online]. Available: https://doi.org/10.1109/LISS.2018.8593220

[4] U. M. Ramya, P. Sindhuja, R. A. Atsaya *et al.*, "Reducing forgery in land registry system using blockchain technology," in *Advanced Informatics for Computing Research*. Springer Singapore, 2018, pp. 725–734. [Online]. Available: https://doi.org/10.1007/978-981-13-3140-4_65

[5] L. Chen, W.-K. Lee, C.-C. Chang, K.-K. R. Choo, and N. Zhang, "Blockchain based searchable encryption for electronic health record sharing," *Future Generation Computer Systems*, vol. 95, pp. 420–429, 2019. [Online]. Available: https://doi.org/10.1016/j.future.2019.01.018

[6] F. Casino, T. K. Dasaklis, and C. Patsakis, "A systematic literature review of blockchain-based applications: Current status, classification and open issues," *Telematics and Informatics*, vol. 36, pp. 55–81, 2019. [Online]. Available: https://doi.org/10.1016/j.tele.2018.11.006

[7] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014, accessed 2019-02-01. [Online]. Available: http://gavwood.com/paper.pdf

[8] E. Androulaki, A. Barger *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, pp. 30:1–30:15. [Online]. Available: https://doi.org/10.1145/3190508.3190538

[9] M. Hearn. (2016) Corda: A distributed ledger. Accessed 2019-02-22. [Online]. Available: https://www.corda.net/content/corda-platform-whitepaper.pdf

[10] M. Giancaspro, "Is a 'smart contract' really a smart idea? insights from a legal perspective," *Computer Law & Security Review*, vol. 33, no. 6, 2017. [Online]. Available: https://doi.org/10.1016/j.clsr.2017.05.007

[11] A. Chauhan, O. P. Malviya, M. Verma, and T. S. Mor, "Blockchain and scalability," in *2018 IEEE International Conference on Software Quality, Reliability*

and Security Companion (QRS-C), July 2018, pp. 122–128. [Online]. Available: https://doi.org/10.1109/QRS-C.2018.00034

[12] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, accessed 2019-02-08. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[13] R. Klomp and A. Bracciali, "On symbolic verification of bitcoin's script language," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology.* Springer International, 2018, pp. 38–56. [Online]. Available: https://doi.org/10.1007/978-3-030-00305-0_3

[14] J. D. Bruce. (2014) The mini-blockchain scheme. Accessed 2019-03-20. [Online]. Available: http://cryptonite.info/files/mbc-scheme-rev3.pdf

[15] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997. [Online]. Available: https://doi.org/10.5210/fm.v2i9.548

[16] I. Grigg, "The ricardian contract," in *IEEE International Workshop on Electronic Contracting*, July 2004, pp. 25–31. [Online]. Available: https://doi.org/10.1109/WEC.2004.1319505

[17] Accord Project LLC. The smart legal contract stack. Accessed 2019-02-25. [Online]. Available: https://www.accordproject.org

[18] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016. [Online]. Available: https://doi.org/10.1109/ACCESS.2016.2566339

[19] A. Salomaa, *Public-key cryptography*, 2nd ed., ser. Texts in Theoretical Computer Science. Springer Science & Business Media, 1996. [Online]. Available: https://doi.org/10.1007/978-3-662-03269-5

[20] E. Palm, O. Schelén, U. Bodin, and R. Hedman, "The exchange network: A general-purpose architecture for digital negotiation and exchange," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN), accepted for publication.*

[21] C. Bell, *Introducing the MySQL 8 Document Store.* Apress, 2018. [Online]. Available: https://doi.org/10.1007/978-1-4842-2725-1

[22] D. Cooper *et al.*, "Internet x.509 public key infrastructure certificate and certificate revocation list (CRL) profile," Internet Request for Comments, RFC Editor, RFC 5280, May 2008. [Online]. Available: https://doi.org/10.17487/RFC5280

[23] R. Fielding *et al.*, "Hypertext transfer protocol (HTTP/1.1): Message syntax and routing," Internet Request for Comments, RFC Editor, RFC 7230, 2014. [Online]. Available: https://doi.org/10.17487/RFC7230

[24] I. Fette and A. Melnikov, "The WebSocket protocol," Internet Request for Comments, RFC Editor, RFC 6455, December 2011. [Online]. Available: https://doi.org/10.17487/RFC6455

[25] G. Bierman, M. Abadi *et al.*, "Understanding TypeScript," in *European Conference on Object-Oriented Programming.* Springer, 2014, pp. 257–281. [Online]. Available: https://doi.org/10.1007/978-3-662-44202-9_11

[26] A. Wirfs-Brock, "ECMAScript 2015 language specification," ECMA International, ECMA-262, 2015. [Online]. Available: http://www.ecma-international.org/ecma-262/6.0

[27] S. Tilkov *et al.*, "Node.js: Using javascript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, 2010. [Online]. Available: https://doi.org/10.1109/MIC.2010.145

[28] S. Faulkner *et al.*, "HTML 5.2," W3C, Working Draft, 2017. [Online]. Available: https://www.w3.org/TR/2018/WD-html53-20180809

[29] T. Atkins and S. Sapin, "CSS syntax module level 3," W3C, W3C Candidate Recommendation, February 2014. [Online]. Available: http://www.w3.org/TR/2014/CR-css-syntax-3-20140220

[30] T. Athan, G. Governatori *et al.*, *LegalRuleML: Design Principles and Foundations.* Springer International Publishing, 2015, pp. 151–188. [Online]. Available: https://doi.org/10.1007/978-3-319-21768-0_6

[31] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," Internet Request for Comments, RFC Editor, RFC 8446, December 2018. [Online]. Available: https://doi.org/10.17487/RFC8446