Railz

A High Throughput Negotiated Consensus Protocol

John Corr Phil Millo

March 27, 2018

Abstract

Although algorithms exist for establishing simple consensus, none allows machines to self-configure and negotiate parameters with each other at scale independently of human intervention. This is primarily due to two factors: (1) the current lack of support for negotiated consensus, and (2) unacceptably low transaction throughput. We present here a novel consensus algorithm that delivers negotiated Nash equilibria with minimal game turns, along with a protocol for securely encoding bundled transactions to achieve effective throughputs many orders of magnitude greater than those currently being achieved. The result is a high throughput negotiated consensus algorithm that is suitable for industrial machine-to-machine price and parameter negotiating applications. This solution is presented in the context of the Railz Protocol.

Contents

List of Figures

1 Introduction

The last few years has seen an explosion of interest in distributed ledger systems which operate on consensus algorithms and are not controlled by a central authority [1]. Although such systems have been in use in various guises for many thousands of years, their recent popularity stems from the Bitcoin blockchain algorithm [2] and its many successors which use network nodes and cryptographic techniques to verify transactions in a public ledger [3]. Much work has been done in considering the general economic advantages and disadvantages of such systems, as well as addressing general technical problems which include but are not limited to simple consensus and authority. However, there remain a number of specific problems which relate to applications for machine-to-machine interactions. Chief amongst these are the challenges of reaching a machine-negotiated consensus, delivering higher ledger throughput, and achieving an appropriate level of utility.

By 'machine-negotiated consensus' we mean a distributed system where two or more machines agree a parameter without the need for human intervention. In traditional human systems such negotiation may be thought of as analogous to a game between players taking turns, each of whom may be rational, and each of whom may play with a public or private strategy. It has been proved that, although such games necessarily have a theoretical consensus point, [4] gameplay may not converge on such a consensus, and accordingly the game may require modification to have utility. We propose, that the general case be modified so that all (machine) players be assumed rational and adopt the same strategy of play which is published and designed to converge on consensus with minimal game turns.

By 'ledger throughput' we mean the number of authenticated machine-negotiated consensuses completed within a unit of time. Since the computing power to reach negotiated consensus may be considered minimal, the effective bottleneck to throughput in distributed systems is the throughput of the system itself. Traditionally, work has focused on increasing such throughput by reducing network latency or otherwise increasing the speed of transactions. Since the application we are considering requires a throughput increase of approximately three orders of magnitude, a different solution is needed. We therefore present a methodology for encoding multiple individual consensuses into bundles each of which is transmitted at the normal network speed. By analogy and applying Goldratt's 'Theory of Constraints' [5], this may be thought of as similar to increasing water throughput by widening the pipe instead of pumping the water faster. Bundles are securely encoded but may be interrogated without disaggregation.

By 'utility' we mean the extent to which a system is fit-for-purpose. We consider utility within the context of a large, distributed system of machines negotiating and agreeing parameters in pairs. As discussed, this machine-to-machine world – referred to by some commentators as 'the fourth industrial revolution', '4IE', or 'Industrie 4.0' [6] – requires not only appropriate levels of robustness, but also a way of achieving an appropriate level of machine negotiated consensus at acceptable throughput rates. For example, a system which could elect to alter the state of a switch but could not agree a switch parameter based on exogenous conditions might not be fit-for-purpose. Similarly, a system which took a day to authenticate that a consensus had been reached would be unlikely to be useful in many real world applications.

The problem of non-cooperating players reaching optimal agreement was hypothesized as early as 1838 [7] but more formally addressed and conceptually solved in the 1950s [8]. Simply stated, 'optimal agreement' means that each player is making the best decision possible, taking into account the decisions of the others in the game as long as the other players' decisions remain unchanged. This state is commonly known as a 'Nash equilibrium' [8] (after Nash who made significant contributions to the field), and is a foundational concept in game theory. However, although it can be proved that a Nash equilibrium always exist for certain types of strategies [4], a number of problems remain: first, Nash equilibria may not always give rise to optimal outcomes (Braess's paradox) [9]; second, the length of time it takes to discover a Nash equilibrium is not addressed. For these reasons, a modified game is proposed for a machine-to-machine negotiation system with cooperating (rather than non-cooperating) players, and public or private game strategies designed to deliver Nash equilibria within a set number of game turns.

In this paper we consider a specific instantiation of a machine-to-machine negotiation system:

the Railz Protocol. We analyse the methods and algorithms which are used to deliver the design goals of machine-negotiated consensus, throughput, and utility. Finally, we present the results of a large-scale simulation which demonstrates these goals are achievable to industrial standards. The methodology and code used in the experiment is provided for review.

Acknowledgements

The authors thank the following people for their contributions: Moira Chadwick for her review of the sections relating to game theory and hypothesis testing; Vignesh Iyer for his review of the sections relating to cryptographic encoding and hash functions, and his assistance with the simulation experiment. The authors also thank the following people for their help in reviewing drafts of this paper: Vichi Chandra; Vicky Coulthard; Simon Hookway; Caelim Parkes; Alain Richardt;

2 Machine-negotiated consensus

2.1 Introduction and design goals

The human world sees complex negotiations occurring between multiple people as a routine part of everyday life. These might be as trivial as a shopper agreeing to purchase a good at the advertised price, or as complex as an agreement between multiple business counterparty to deliver a range of goods and services with multiple interdependencies. Negotiating the trivial case of an agreement to buy at an advertised price is itself trivial, and can straightforwardly be synthesised by an algorithm that accepts or declines execution of the contract according to one or more simple parameters. These systems are sometimes described as "self-executing contract", although any contract where one party signs leaving the other the option of either signing and thus completing the contract or declining to sign and thus declining the contract is similarly 'self-executing'.

A more sophisticated approach needs to be taken if the goal is to negotiate a parameter. The wealth of literature on human negotiation strategies and, in particular, the work done since 1979 by the , indicates that negotiation between parties is unpredictable, hard to control, may not reach an optimal conclusion, and in fact may not necessarily reach any conclusion at all. To analyse why this is the case we can model such negotiations as a game between two players, each taking sequential turns in offering outcomes. The work done by Nash and others since the 1950s on game theory shows that such games always have one or more theoretical equilibria points such that a deviation results in a worse outcome for both players. However:

1. None of the equilibria is necessarily acceptable to any of the players.
2. None of the equilibria is necessarily optimal.
3. None of the equilibria is necessarily discoverable in a fixed number of game turns.

Our goal is to design a system capable of being populated entirely by machines governed by algorithms such that they can easily and quickly negotiate between themselves to reach optimal consensus. To achieve this, we consider a special case of the game described above whereby the players agree to a 'gameplay agreement' that compels them to implement the optimal Nash equilibrium based on their joint payoff functions. In this case:

1. The equilibrium is by definition acceptable.
2. The chosen equilibrium is optimal (since that is a condition).
3. The equilibrium may be discovered in a fixed number of game turns.

This allows machine-negotiated consensus to be effected with an appropriate level of utility.

2.2 Consideration of equilibria in mixed strategy games

To examine this in more detail, we formally consider what we mean by an 'optimal equilibrium' with reference to a theoretical game between two players.

Consider therefore two players: Satoshi and Vitalik, each of whom has n possible moves to chose from. An $n \times n$ matrix M specifies the payoffs as follows: if Satoshi chooses move i and Vitalik chooses move j, then the payoff to Satoshi is $M[i, j]$ and the payoff to Vitalik is $-M[i, j]$. In this way, the game is "zero sum" since Satoshi and Vitalik's payoffs sum to zero.

A *mixed strategy* is one in which the player chooses from some distribution of moves, that is they choose move i with probability p_i. So $\forall\ i,\ p_i \geq 0$ and $\sum_i p_i = 1$.

A pair of mixed strategies p and q, for Satoshi and Vitalik gives rise to an expected payoff to Satoshi of $\sum_{i,j} p_i q_j M[i, j]$ and an expected payoff to Vitalik of $-\sum_{i,j} p_i q_j M[i, j]$.

Such a pair is said to be a *Nash equilibrium* if no player has any incentive unilaterally to deviate from their current mixed strategy. So

$$\forall\ p^* \neq p, \quad \sum_{i,j} p_i^* q_j M[i, j] \ \leq\ \sum_{i,j} p_i q_j M[i, j] \tag{1}$$

and

$$\forall\ q^* \neq q, \quad -\sum_{i,j} p_i q_j^* M[i, j] \ \leq\ -\sum_{i,j} p_i q_j M[i, j] \tag{2}$$

2.3 Negotiated consensus algorithm

2.3.1 Modified gameplay and gameplay agreement

Players can negotiate a consensus with minimal game turns using the following modified gameplay:

1. The players exchange a *gameplay agreement* which defines the protocol and commits them to implementing the Nash equilibrium.[1]
2. The players exchange payoff matrices in encrypted form.
3. The players exchange decryption keys for their payoff matrices.
4. The Nash equilbrium is discovered and implemented by both players.

The gameplay agreement governs and enforces not only the protocol of sequential play, but also the final outcome. It should include at a minimum:

- A statement of which players goes first.
- The turn-by-turn protocol so that the players effect first the exchange of their encrypted payoff matrices, confirm receipt (if necessary), and then (and only then) exchange decryption keys.
- An agreement that the optimal equilibrium is enforced as the 'move' for both players.

2.3.2 Complex payoff functions

On a theoretical basis, payoff matrices can be discrete or continuous, linear or non-linear, and contain unusual contingent elements, provided that they can be solved using an algorithm for Nash equilibria without resorting to iterative or recursive techniques.

The working prototype of Railz models payoff functions as up to five linear equations joined to comprise a continuous function.

[1]Note that, depending on the complexity of payoff matrix, there may be more than one equilibrium.

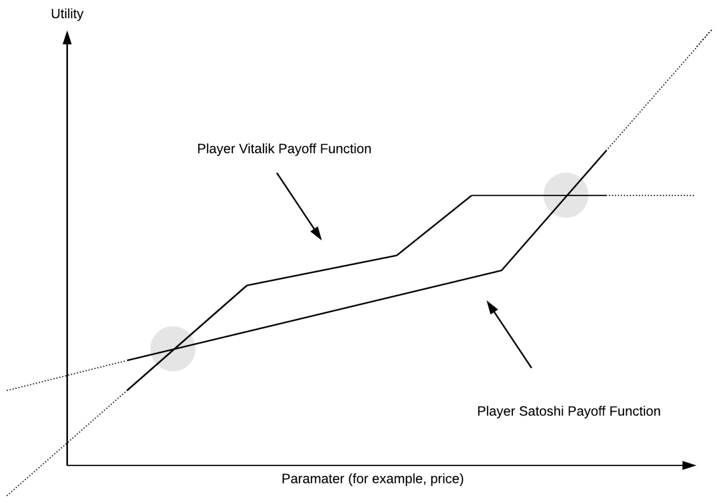

Figure 1: Example of found equilibria for complex payoffs implemented in Railz

2.3.3 Ensuring honest play

By first exchanging their payoff matrices in encrypted form and only then exchanging decryption keys means that the players can be sure that the game is honest. Were they to exchange unencrypted payoff matrices then there is a risk that the player who moves second in the exchange could alter her payoff matrix before sending based on information from the first player.

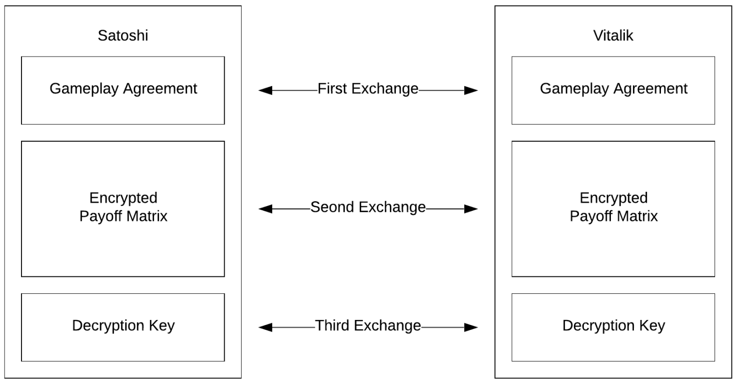

Figure 2: Schematic showing modified gameplay for Railz

2.3.4 Consideration of Braess's paradox

Such an equilibrium may not equate with the best overall flow through a network. This paradox was noted by Dietrich Braess as early as 1968 1968 [9] who stated:

> For each point of a road network, let there be given the number of cars starting from it and the destination of the cars. Under these conditions, one wishes to estimate the distribution of traffic flow. Whether one street is preferable to another depends not only on the quality of the road, but also on the density of the flow. If every driver takes the path that looks most favourable to them, the resultant running times need

> not be minimal. Furthermore, it is indicated by an example that an extension of the road network may cause a redistribution of the traffic that results in longer individual running times.

This is easily seen by considering a shortcut on a journey – if too many people choose the shortcut then it may become congested and soon cease to reduce overall journey time, and has analogies to the prisoner's dilemma. [10]

This paradox is highly relevant to the considered application where a large network of machines are continuously negotiating consensus in pairs. Care needs to be taken to ensure that in such a dynamic system equilibria remain optimal.

2.4 ERC20 implementation

The Railz Protocol is entirely implemented and executed under the ERC20 Ethereum token standard and not at the base code level.

1. Game turns are managed and effected by a Railz Agent.
2. Interim data is stored within the agent of, if necessary, written to a side blockchain.
3. The final consensus output is optionally written to the main Ethereum blockchain.

2.4.1 Use of Railz validators to effect game turns

The multiple game turns mean that there are multiple transactional steps, and accordingly interim parameters that need to be stored, albeit temporarily. Rather than these all being written to the main Ethereum blockchain, Railz uses validators or 'miners' in a sidechain that is separate from the main Ethereum blockchain.

2.4.2 Use of side blockchain and other temporary storage for interim data

In this way, nothing is written to the main Ethereum blockchain until the final consensus is reached, and there are, accordingly, no transaction fees.

2.4.3 Final optional writing to main Ethereum blockchain

Once the final consensus is reached then, depending on a parameter stored in the Railz metadata, either discarded if there is no requirement to retain it, or written to the main Ethereum blockchain which triggers the standard mining cost.

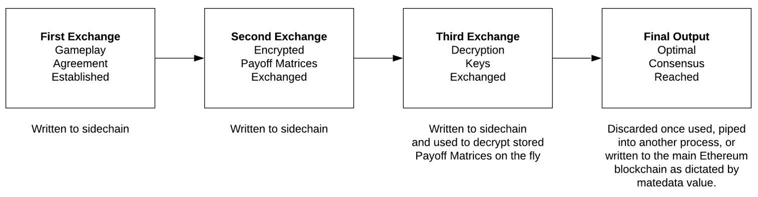
First Exchange
Gameplay Agreement Established
Second Exchange
Encrypted Payoff Matrices Exchanged
Third Exchange
Decryption Keys Exchanged
Final Output
Optimal Consensus Reached
Written to sidechain
Written to sidechain
Written to sidechain and used to decrypt stored Payoff Matrices on the fly
Discarded once used, piped into another process, or written to the main Ethereum blockchain as dictated by matedata value.

Figure 3: Schematic showing stage by stage data storage in Railz

3 High throughput

3.1 Introduction and design goals

Many systems desire to increase their performance in terms of the number of outcomes they are able to deliver in a fixed unit of time. Although this is straightforwardly a consideration of the system's theoretical 'throughput', this is often confused with a desire to improve the system's raw 'speed' – a related, though different, concept.

Consider the system of a water pipe whose goal is to deliver the maximum amount of water in a fixed unit of time. This may be achieved:

1. By moving the water faster through the pipe, thereby increasing the speed of the system.
2. By making the pipe wider, thereby increasing the bandwidth or throughput of the system, but not increasing the speed of the water.

Where systems need greater throughput it is therefore important to ask: Do we need to (analogously) move the water faster, or can we simply make the pipe wider?

There is currently a limitation in the speed with which ledger entries can be written to and retrieved from distributed ledgers, and specifically (for our purposes) the Ethereum network. This limitation arises in part from the same attribute that delivers the advantage of immutability – the distributed nature of the system combined with the proof of work mechanism – since these place a necessary time overhead on propagation.

However, many applications (including the specific use case we are considering) require only greater throughput, and would not be materially improved by greater speed – in the language of the earlier analogy, a wider pipe rather than faster water. Such a 'wider pipe' may be effected by bundling ledger entries together and transmitting this bundle at the *same speed* as that of a single ledger entry.

Such modification can be made to most existing blockchain networks including the Ethereum network, and have the following high level design goals:

1. Multiple ledger entries should be bundled into a single payload for transmission.
2. Bundled payload should support the ability to be either private (encrypted) or public (readable).

3.2 Current transport mechanism

Current systems allow single ledger items to be transmitted for hashing onto a blockchain as shown in Figure 4.

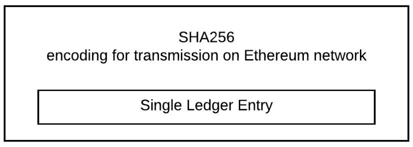

Figure 4: Schematic showing conventional hashing of single ledger item

3.3 Improved transport mechanism

For our purposes, we need to make a number of modifications:

1. Multiple ledger are first composited using asymmetric encryption.
2. Metadata is added to the composited 'bundle', encrypted using a one-way hash function, and transmitted in the usual way.

3.3.1 Metadata structure

The metadata is designed to be straightforward interrogatable from the blockchain and, depending on its content, can be used to make further interrogations on the fly. The metadata should include, at a minimum:

- Either the decryption key (if the data is to be publically interrogatable) or a null value.
- The structure of the ledger data.
- The number of ledger items encoded.

This is shown in Figure 5.

3.3.2 Multi-layer encoding design

The SHA-256 secure hash algorithm is analogous to a 'digital signature' in that, by comparing hashes, observers can verify *post hoc* that the package and its contents were not tampered with. This anti-tempering technology is needed at the top level, but redundant if included again within since the package authentication verifies by inference the package contents. Therefore, the package contents – in our example, the metadata, the bundled items, and so on – can be straightforwardly encoded using symmetric or asymmetric encryption to provide however many layers of security are needed, as well as the functionality of combining the data into a single payload for transmission. We propose the following three stage encoding protocol:

1. Multiple ledger entries are encoded into a bundle using asymmetric encryption to produce a *raw bundle.*
2. This raw bundle is then combined with metadata using an archiving algorithm to produce a single *payload.*
3. This payload is hashed using a secure hash algorithm for transmission on, in our use case, the Ethereum blockchain.

The choice of algorithms is not critical to the theory, but the Railz proof of concept is implemented using the following:

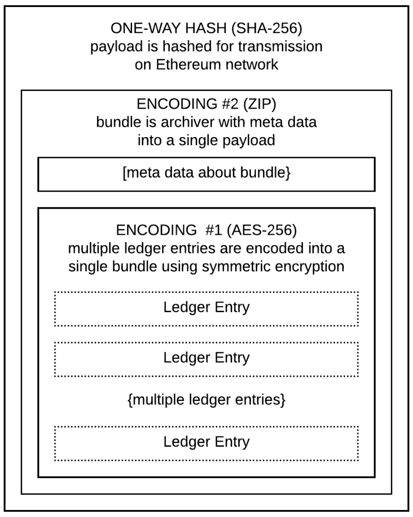
ONE-WAY HASH (SHA-256)
payload is hashed for transmission
on Ethereum network
ENCODING #2 (ZIP)
bundle is archiver with meta data
into a single payload
[meta data about bundle}
ENCODING #1 (AES-256)
multiple ledger entries are encoded into a
single bundle using symmetric encryption
Ledger Entry
Ledger Entry
{multiple ledger entries}
Ledger Entry

Figure 5: Schematic showing multi-encoded and hashed final payload for Railz

1. The multiple ledger items are symmetrically encrypted into a single bundle using the Advanced Encryption Standard with 256 bits (AES-256) [11].
2. The single bundle is combined with metadata and compressed into a single payload using the ZIP archive and compression standard [12].
3. The payload is securely hashed using the Secure Hash Algorithm 2 with a hash value of 256 [13].

3.4 ERC20 implementation

3.4.1 Writing to the Ethereum blockchain

It is important to note that the digest from the hashed encoded payload is written as a single item to the Ethereum blockchain, and that the payload is not unbundled. In this way, although the transaction cost arising from the machine-negotiated consensus algorithm is increased, the cost per unit entry is decreased (assuming that sufficient ledger entries are bundled).

3.4.2 Interrogation on the Ethereum blockchain

Once the digest payload has been written to the Ethereum blockchain, it may be interrogated in situ according to the following protocol:

1. The digest is identified as a Railz compliant digest.
2. The metadata and bundle are decoded and exposed by reversing the ZIP archiving and compression function.
3. The metadata is interrogated and the decryption key for the bundle is retrieved, if available.
4. The decryption key is used to decrypt and disassemble the bundle on the fly.

4 Utility

4.1 Introduction and context

The question of whether of not a system has 'utility' has to be considered within the context of the application for which it was designed. This paper contemplates two significant modifications to current standard practice for ERC20 tokens on the Ethereum network:

1. The usual 'smart contract' algorithm be extended to encompass the ability to negotiate consensus outcomes (as described in Section 2).
2. The effective throughput be increased by programatically bundling ledger items (as described in Section. 3)

4.1.1 Definitions of utility

Utility tests for each of the above are as follows:

1. The new negotiated consensus algorithm is deemed to have sufficient utility if it does not fail to produce the desired outcome when tested.
2. The increased throughput protocol is deemed to have sufficient utility if it can be shown to deliver the desired increase in throughput.

Since there is currently a working proof of concept for the consensus algorithm, this test is considered to have been passed for the purpose of this paper, and the rest of this section deals only with assessing the test for utility of the increased throughput protocol.

4.2 Statistical test of throughput utility using Student's *t*-test

To test for the utility of the throughput, we use a statistical hypothesis test using the test statistic following Student's[2] t-distribution under the null hypothesis.

To do this, we assume that our application requires a delivered effective throughput of 10,000 times current rates. To assess the fitness-for-purpose of the bundling algorithm – in other words the statistical confidence that the bundling algorithm does not materially lower the effective delivered throughputs – we established samples for the bundled delivery times and a control.

4.2.1 Bundle and control samples

For the bundle sample:

1. 10,000 single ledger items plus metadata were first combined using the bundling algorithm into a single payload.
2. The bundled payload was transmitted via the Ethereum network.
3. Time to settle was recorded.
4. Steps 2 and 3 were repeated (with the same bundled payload) to obtain 100 observations ($N_b = 100$).

For the control sample:

1. a single ledger item was transmitted via the Ethereum network.

[2]"Student" was the pen name of William Sealy Gosset who introduced the t-statistic in 1908.

2. Time to settle was recorded.

3. The experiment was repeated to obtain 100 observations ($N_c = 100$).

Since both samples sizes were 100, and assuming normality, we then used Student's t-test with 99 degrees of freedom[3] noting that the t-distribution[4] is the distribution of the sample mean relative to the true mean, divided by the sample standard deviation, after multiplying by the standardizing term $\sqrt{n}$.

If v denotes the degrees of freedom, and Γ denotes the gamma function, then the probability density function for Student's t-statistic for comparing two sample means is:

$$f(t) = \frac{\Gamma\left(\frac{v+1}{2}\right)}{\sqrt{v\pi}\,\Gamma\left(\frac{v}{2}\right)}\left(1+\frac{t^2}{v}\right)^{-\frac{v+1}{2}} \tag{3}$$

4.2.2 t-statistic

If $\bar{x}$ is the sample mean, S is the standard error, N is the sample size, and the bundle and control populations are indicated by the suffixes b and c, then the t statistic is:

$$t = \frac{\bar{x}_b - \bar{x}_c}{\sqrt{\dfrac{S_b^2}{N_b} + \dfrac{S_c^2}{N_c}}} \tag{4}$$

4.2.3 p value and Type II error estimation

The p value produced by the experiment represents the statistical confidence that the time to deliver the bundled payload time is not greater than the time to deliver the control. The statistical chance of the incorrect rejection of the null hypothesis H_0 (known as a Type II error) is $1 - p$.

4.2.4 Equivalent hypothesis test: definitions of H_0 and H_a

This is equivalent to conducting a one-tailed hypothesis test where the null hypothesis H_0 and the alternate hypothesis H_a are as follows:

$$H_0 = \bar{x}_b \leq \bar{x}_c \tag{5}$$

$$H_a = \bar{x}_b > \bar{x}_c \tag{6}$$

4.2.5 Rejection of H_0

The experiment produced a p value which led us to reject the null hypothesis H_0 in favour of the alternate hypothesis H_a and conclude with a high level of statistical confidence that the bundling algorithm is capable of delivering effective throughputs 10,000 greater than the current level.

4.3 Results data

A copy of the results data is available for peer review.

[3]For n observations there are $n - 1$ degrees of freedom

[4]being the distribution of the t-statistic

5 Railz: high throughput negotiated consensus

The theories outlined in this paper were designed to be implemented with a specific application in mind: the Railz Protocol.

The Rail Protocol, or simply 'Railz', delivers high-throughput negotiated consensus between nodes on the Ethereum network, and is designed to be straightforwardly implemented under the ERC20 Ethereum Token Standard as opposed to requiring implementation at the base network level.

5.1 Proof of ability to implement under ERC20

The ERC20 Ethereum token standard (ERC standing for "Ethereum request for comments") was developed in 2015 and is used for Ethereum smart contracts. The standard defines a common list of rules that an Ethereum token has to implement to function properly on the Ethereum ecosystem. The standard is highly specific to contracts and includes support for functions like:

- The ability to confirm total token supply.
- The ability to confirm account balance of another account with a specified address.
- The ability to confirm amount of Ether to be sent.
- The ability to confirm the address the Ether is to be sent from.

The main programming language is Solidity which is a specialist contract-oriented language designed for writing smart contracts to various blockchains including (but not limited to) the Ethereum blockchain. Although Solidity has a number of specialist features which contemplate certain applications, the language is Turing-complete which means it can provably be used to simulate any Turing machine, which in turn means it can provably be used to simulate any algorithm capable of being expressed in logic. It is therefore provably true that Solidity can be used to program the various algorithms outlined in this paper, and consequently it is provably true that these methods can be implemented under the ERC20 Ethereum token standard.

5.2 Design goals met

These joint design goals of high throughput and negotiated consensus are designed for large machine-to-machine 'internet of things' applications, and the following specific use cases were considered when assessing utility:

As described in this paper, the design goals have sufficient utility to be considered fit for purpose. Both the negotiated-consensus algorithm and high throughput protocols are novel steps as implemented on the Ethereum or other blockchain based networks. The result is that Railz is provably superior in these respects to current implementations.

- Bi-directional price negotiation such as might be found in an automated bidding system.
- Single or multiple simultaneous parameter negotiation such as might be found in an automated machine configuration system.
- Binary or multi-state switch negotiation such as might be found in a distributed home automation system.

5.3 Conclusion

We believe that Railz represents a materially novel improvement on the current standard 'smart contract' implementation found on the Ethereum network. Its ability not only to facilitate sophisticated machine-to-machine negotiations but also to do this at significantly higher throughputs than the data flows currently available make it highly suitable for large scale distributed internet of things applications.

References

[1] Vitalik Buterin. *A next-generation smart contract and decentralized application platform.* 2014. URL: http://www.the-blockchain.com/docs/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf.

[2] Satoshi Nakomoto. *Bitcoin: a Peer-to-Peer Electronic Cash System.* May 2009. URL: https://bitcoin.org/bitcoin.pdf.

[3] Gavin Wood. *Ethereum: A secure decentralised generalised transaction ledger.* 2014. URL: http://www.cryptopapers.net/papers/ethereum-yellowpaper.pdf.

[4] Prajit K. Dutta. *Strategies and games: theory and practice.* MIT Press, 1999. ISBN: 978-0-262-04169-0.

[5] Eliyahu M. Goldratt and Jeff Cox. *The Goal.* North River Press, 1984. ISBN: 978-0-88427-178-9.

[6] BMBF-Internetredaktion. 2016. URL: https://www.bmbf.de/de/zukunftsprojekt-industrie-4-0-848.html.

[7] Antoine A. Cournot. *Recherches sur les Principes Mathématiques dela Théorie des Richesses.* L. Hachette, 1838. ISBN: ISBN-10: 2100058967.

[8] John F. Nash. *Equilibrium points in n-person games.* 1950. URL: http://www.pnas.org/content/36/1/48.full.

[9] Dietrich Braess. "Über ein Paradoxon aus der Verkehrsplanung". In: *Unternehmensforschung 12, 258–268* (1969).

[10] Steven Kuhn. *Prisoner's Dilemma.* 1997. URL: https://plato.stanford.edu/archives/spr2017/entries/prisoner-dilemma/.

[11] Federal Information Processing Standards Publication 197. *Announcing the Advanced Encryption Standard (AES).* 2001. URL: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[12] Paul Lindner. *Registration of a new MIME Content-Type/Subtype.* 1993. URL: https://www.iana.org/assignments/media-types/application/zip.

[13] Andrew W. Appel. *Verification of a Cryptographic Primitive: SHA-256.* 2015. URL: https://www.cs.princeton.edu/~appel/papers/verif-sha.pdf.

Index