TODO TITLE

Kelong Cong

TODO TITLE

Master's Thesis in Computer Science

Parallel and Distributed Systems group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Kelong Cong

25th June 2017

Author
Kelong Cong

Title
TODO TITLE

MSc presentation
31st August 2017

Graduation Committee

Prof. D. H. J. Epema	Delft University of Technology
Dr. J. Pouwelse	Delft University of Technology
Dr. Z. Erkin	Delft University of Technology

Abstract

TODO ABSTRACT

Preface

TODO MOTIVATION FOR RESEARCH TOPIC

TODO ACKNOWLEDGEMENTS

TODO AUTHOR

Delft, The Netherlands
25th June 2017

Contents

Chapter 1

Introduction

We live in a world where technologies have become vital for our welfare and success. The internet, for instance, gave us the ability of efficiently exchange information on a global scale. However, in contrast to its original design, the internet and in particular the world wide web is becoming increasingly centralised. The domain name system, certificate authorities, to name a few, carry enormous responsibilities and are a central point of failure. The same can be said for many other services such are online marketplaces, cloud services, hospitality services and even our banking system. The 2008 financial crisis is an example of the banking system making poor choices which result in a decline in consumer wealth in the order of trillions [2] and led to the European debt crisis.

Ironically, also in 2008, Satoshi Nakamoto published the Bitcoin whitepaper [16]. Which, for the first time, gave us a simple banking system in the form of an distributed ledger. It needs no central control but still incorruptable even if there are malicious parties that aim to undermine the system. We call such a ledger a blockchain.

The primary innovations are (1) its consensus model, which prevents double spending, and (2) its incentive mechanism that encourages anyone (with adaquate hardware) to participate in the network and keep it running. The double spending problem can be seen as an inconsistency issue. For example, C has 5 units of currency in her account. At some instant in time, A sees that C transferred 5 units to A and B sees that C transferred 5 units to B. Hence, the transaction is inconsistent with respect to the observer. Bitcoin and many other blockchain systems solve the inconsistency problem with a consensus algorithm. The goal of the algorithm is reach agreement on a set of values, e.g. transactions, which eliminates inconsistencies. In Bitcoin's case, the consensus algorithm is called *proof-of-work*. Where miners (parties that runs the Bitcoin network) collect transactions and compete in solving (using brute force) a puzzle. The first miner to solve it generates a block containing all the collected transactions. The miner is also rewarded with

new coins and transaction fees. It is important to note that every block, that is the solution of the puzzle, depends on the previous block. Hence the name blockchain. Further, honest miners always work on the largest chain. Due to the difficulty of the puzzle, it is unlikely for more than one blockchain to exist in the network for a long period of time. Thus every party sees a consistent blockchain which solves the double spending problem. Further, due to the reward mechanism, people are incentivised to act as miners and keep the system running.

Bitcoin has had its ups and downs, but over it has grown into an enormous system. Its power consumption is as high as Republic of Ireland [17]. Its market cap, at the time of writing, is over 40 Trillion USD [7]. Many online marketplaces are using Bitcoin, for example Steam[1] and even Amazon[2]. Due to its success, people from many different disciplines began investigating in way to use blockchain technology. This includes finance [**todo**], heath care [**todo**], logistics [**todo**], energy [**todo**] and so on.

Sadly, as traditional blockchain systems began to gain popularity, their limitations also became apparent. Bitcoin has the infamous 7 transactions per second upper bound. This is due to the fact that blocks are fixed to 1 MB are only generated on average every 10 minutes. Since every Bitcoin transaction is at least 250 bytes, it computes to about 7 transactions per second. Due to this limitation, it is not uncommon to see a long backlog of about 20000 transactions on `https://blockchain.info`[3]. A few months ago the backlog even reached 100000, which meant new transactions would take at least 11 hours to hit the blockchain [21]. The issue has plagued the Bitcoin community for some time and is the root cause for the block size debate, which some calls it a civil war [23]. Parties that are for the increase in block size argue that a larger block would improve the transaction rate. Parties against it argue that it would make mining more centralised because blocks take longer to propogate through the network and it does not solve the fundamental problem. A recent empirical study Croman et el. [8] has shown that simply increasing the block size may help. But given the bandwidth and latency constraints, it not possible to have more than 758 transactions per second and it may give some miner an unfair lead over others. Thus fundamental changes are necessary run at the scale of centralised payment processors such as Visa [26], which is in the order of tens of thousands transactions per second.

In this work, we take the advice of Croman et el. [8] and Marko [27] and rethink the blockchain architecture. Our primary insight came from observing the differences between how transactional systems work in the real world and how they work in blockchain systems like Bitcoin. Take a

[1] `https://store.steampowered.com/`
[2] Not directly, but via `https://purse.io/`.
[3] `https://blockchain.info/unconfirmed-transactions/`

restaurant owner for example, most of the time the customer is honest and pays the bill. There is no need for the customer or the restaurant owner to report the transaction to any central authority because both parties are happy with the transaction. On the other hand, if the customer leaves without paying the bill, then the restaurant owner would report the incident to some central authority, e.g. the police. On the contrary, in blockchain systems, every transaction is effectively sent to the miners, which can be seen as a collective authority. This consequently lead to limited scalability because every transaction must be validated by the authority even when most of the transaction are legitimate.

We describe and analyse the aforementioned idea in the remainder of this work. We begin with the problem description in Chapter 2. This is followed by a detailed formulation of the model in Chapter 3. Next, we analyse the correctness and performance of our design in Chapter 4, this is where we present our key theorems. Implementation and experimental results are discussed in Chapter 5. Finally, we compare our system with other state-of-the-art blockchain systems and conclude in Chapters 6 and 7 respectively.

Chapter 2

Problem Description

- long history of BFT, since Lamport - PBFT - PBFT triggered a renaissance in BFT replication research, with protocols like Q/U

- classical blockchains are nice, but is it possible to also use 30 years of BFT research? - non-consensus chains scales, but no consensus... - alternative consensus model

Chapter 3

System Architecture

The goals guiding our system design are as follows.

- Performance and scalability,
- application neutrality and
- security.

As mentioned in the Introduction, the primary goal is performance and scalability. Having a scalable blockchain system while still keeping global consensus allows the system to be ubiquitous and realise the full potential of blockchain.

The secondary goal is to design an application neutral system. In particular, it should act as a backbone that provides the building blocks of blockchain based applications. It should be possible to buld any kind of application on top our system. Further, we do not impose on a consensus algorithm, as long as it satisfies the properties of atomic broadcast which we describe in Section 3.1.2.

Due to the nature of our system, we do not explicitly address the Sybil attack [10]. Sybil defence mechanism always require some form of reputation score from the application. For example, social network based Sybil defence mechanims use graph structure of real-world relationships [29]. Online marketplaces such as Amazon use the rating of buyer and sellers. Thus it is not possible to design a Sybil defence mechanism with a application neutral framework. On the other hand, our system also has no restrictions on the Sybil defence technique and applications built on top of our system can use the best mechanism for their purpose.

The third and final goal is security. Our system should be unaffected in the presence of powerful adversaries. In particular, adversaries are Byzantine meaning that they can have arbitrary behaviour. Thus anything is possible from simply ommitting messages to colluding with eachother in order to undermine the whole system.

We begin the chapter with an intuitive overview of the architecture in Section 3.1. Next, we give the formal description, starting with the model and assumptions in Section 3.2. Then, the three protocols which make up the complete system, namely consensus protocol (Section 3.4), transaction protocol (Section 3.5) and validation protocol (Section 3.6). Finally, the possible extensions are described in **??**.

3.1 System Overview

The system consist of one data structure—Extended TrustChain, and three protocols—consensus protocol, transaction protocol and validation protocol. We first describe each component individually and then explain how they fit together in Section 3.1.4.

3.1.1 Extended TrustChain

Extended TrustChain naturally builds on top of the standard TrustChain. Thus we first describe the standard TrustChain. Our description has minor differences compared to the description in [**trustchain**]. This is to help with the description of the extended TrustChain. We remark the difference when it occurs. However, the two descriptions are functionally the same.

Standard TrustChain

In TrustChain, every node has a "personal" chain. Initially, the chain only contains a genesis block generated by the nodes themselves. When a node A wishes to add a new transaction (TX) with B, a new TX block is generated and then appended to A's chain. The TX block must have a valid hash pointer pointing to the previous block and a reference[1] to its *pair* on B's chain. As a result, a single transaction generates two TX blocks, one on each party's chain. An example of is shown in Figure 3.1.

If every node follows the rules of TrustChain and we only consider hash pointers, then every chain effectively forms a singly linked list. However, if a node violates the rules, then a *fork* may happen. That is, there may be more than one TX block with a hash pointer pointing back to the same block. In Figure 3.1, node b (in the middle chain) created two TX blocks that both point to $t_{b,5}$. If this is a ledger system it can be seen as a double spend, where the currency accumulated up until $t_{b,5}$ are spent twice.

[1]This is different from the original TrustChain definition found in [**trustchain**]. In there, a TX block has two outgoing edges which are hash pointers to the two parties involved in the transaction. This work uses one outgoing edge and a reference.

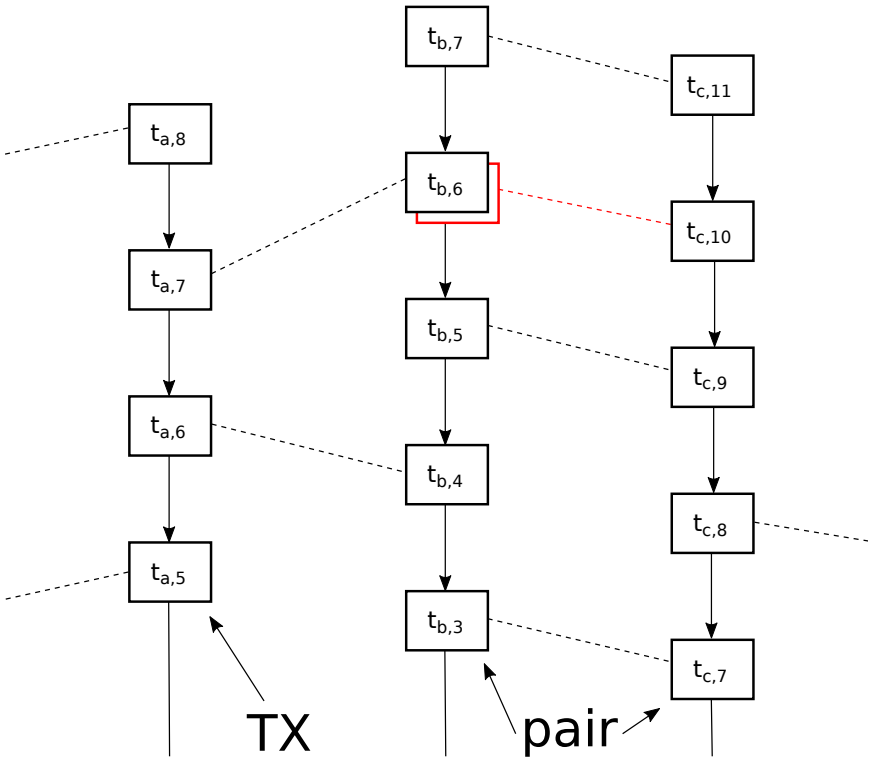


Figure 3.1: Every block is denoted by $t_{i,j}$, where i is the node ID and j is the sequence number of the block. Thus we have three nodes and three corresponding chains in this example. The arrows represent hash pointers and the dotted lines represent references. The blocks at the ends of one dotted line are pairs of each other. The red block after $t_{b,5}$ indicate a fork.

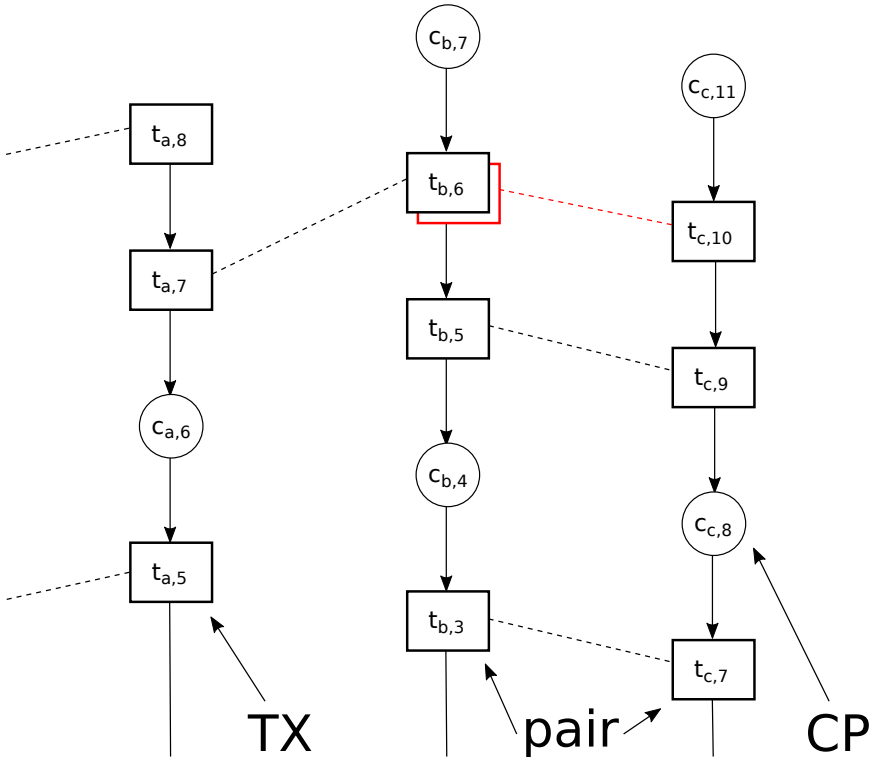


Figure 3.2: The circles represent CP blocks, they also have hash pointers (arrow) but do not have references (dotted line). Note that the sequence number counter do not change, it is shared with TX blocks.

Extended TrustChain

We are now ready to explain the Extended TrustChain. The primary difference is the introduction of a new type of block—checkpoint (CP) block. In contract to TX blocks, CP blocks do not store transactions or contain references. Their purpose is to capture the state of the chain and the state of the whole system. In particular, the state of the chain is captured with a hash pointer. The state of the whole system is captured in the content of the CP block, namely as a digest of the latest *consensus result* which we explain in Section 3.1.2. A visual representation is shown in Figure 3.2.

From this point onwards, we use TrustChain to mean the Extended TrustChain unless explicitly clarified.

3.1.2 Consensus Protocol

The consensus protocol can be seen as a technique of running infinitly many rounds of some Byzantine consensus algorithm[2], starting a new execution immediately after the previous one is completed. This is necessary because blockchain systems always need to reach consensus on new values proposed by nodes in the system, or CP blocks in our case.

The high communication complexity of Byzantine consensus algorithms prohibits us from running it on a large network. Thus, for every round, we randomly select some node—called facilitators—to collect CP blocks from every other node and use them as the proposal. The facilitators are elected using a *luck value*, which is computed using $H(\mathcal{C}_r||pk_u)$, where $\mathcal{C}_r$ is the consensus result (which can be seen as the set union of all the CP blocks collected by the facilitators) in round r and pk_u is the public key of node u. Intuitively, the election is guaranteed to be random because the output of a cryptographically secure hash function is unpredictable and $\mathcal{C}_r$ cannot be determined in advance.

A visual explaination can be found in Appendix A, it walks through the steps needed for a node to be selected as a facilitator using an example.

3.1.3 Transaction and Validation

The TX protocol is a simple request and response protocol. The nodes exchange one round of messages and create new TX blocks on their respective chains. Thus, as we mentioned before, one transaction should result in two TX blocks.

The consensus and transaction protocol by themselves do not provide a mechanism to detect forks or other forms of tamperaing. Thus we need a validation protocol to counteract malicious behaviour. When a node wish to validate one of its TX, it asks the counterparty for the *fragment* of the TX. A fragment of a TX is a section of the chain beginning and ending with CP blocks that contains the TX and are in consensus. Upon the counterparty's response, the node checks that the CP blocks are indeed in consensus, the hash pointers are valid and his TX is actually in the fragment. The TX is valid if these conditions are satisfied. Intuitively, this works because it is hard (because a cryptographically secure hash function is second-preimage resistant) to create a different chain that begins and ends with the same two CP blocks but with a different middle section.

3.1.4 Combined Protocol

The final protocol is essentially the concurrent composition of the three aforementioned protocols, all making use the Extended TrustChain data

[2]More accuratly it is ACS or asynchronous subset consensus, we describe ACS in Section 3.4.1.

structure.

Our subprotocol design gives us the highly desireable non-blocking property. In particular, we do not need to "freeze" the state of the chain for some communication to complete in order to create a block. For instance, a node may start the consensus protocol, and while it is running, create new transactions and validate old transactions. By the time the consensus protocol is done, the new CP block is added to whatever the state that the chain is in. It is not necessary to keep the chain immutable while the consensus protocol is running.

3.2 Model and Assumptions

This section and the ones following it give a technical treatment of what the content in System Overview. For notational clarity, we use the following convention (adapted from [15]) throughout this work.

- Lower case (e.g. x) denotes a scalar object or a tuple.
- Upper case (e.g. X) denotes a set or a constant.
- Sans serif (e.g. $\mathsf{fn}(\cdot)$) denotes a function.
- Monospace (e.g. `ack`) denotes message type.

Further, we use $a||b$ to denote concatination of the binary representations of a and b.

We assume purely asynchronous channels with eventual delivery. Thus in no circumstance do we make timing assumptions. The adversary has fully control of the delivery schedule and the message ordering. But they are not allowed to drop messages[3].

We assume there exist a Public Key Infrastructure (PKI), and nodes are identified by their unique and permanent public key. Finally, we use the random oracle model, i.e. calls to the random oracle are denoted by $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$, where $\{0,1\}^*$ denotes the space of finite binary strings and λ is the security parameter [3].

In our model we consider N nodes, which is the population size. n of them are facilitators, t out of n are malicious and the inequalities $n \geq 3t+1$ and $N \geq n+t$ must hold. The first is from the FLP impossibility result TODO. The second is necessary due to our system design, which becomes clear in TODO.

Our threat model is as follows. We use a restricted version of the adaptive corruption model. The first restriction is that corrupted node can only change across rounds. That is, if a round has started, the corrupted nodes

[3] Reliability can be achieved in unreliable networks by resending messages or using some error correction code.

cannot changed until the next round. The second restriction is that the adversary, presumably controlling all the corrupted nodes, is forgetful. Namely the adversary may learn the internal state such as the private key of a corrupted node, but if the node recovers, then the adversary must forget the private key. Otherwise the adversary can eventually learn all the private keys and sabotage the system. Finally, we assume computational security. That is, the adversary can run polynomial-time algorithms but not exponential-time algorithms.

3.3 Extended TrustChain

The primary data structure used in our system is the Extended TrustChain. Each node u has a public and private key pair—pk_u and sk_u, and a chain B_u. The chain consist of blocks $B_u = \{b_{u,i} : k \in \{0, \ldots, h-1\}\}$, where $b_{u,i}$ is the ith block of u, and $h = |B_u|$. We often use $b_{u,h}$ to denote the latest block. There are two types of blocks, TX blocks and CP blocks. If T_u is the set of TX blocks of u and C_u is the set of CP blocks of u, then it must be the case that $T_u \cup C_u = B_u$ and $T_u \cap C_u = \varnothing$. The notation $b_{u,i}$ is generic over the block type. We assume there exist a function $\mathsf{typeof} : B_u \to \{\tau, \gamma\}$ that returns the type of the block, where τ represents the TX type and γ represents the CP type.

3.3.1 Transaction Block

The TX block is a six-tuple, i.e $t_{u,i} = \langle \mathsf{H}(b_{u,i-1}), seq_u, txid, pk_v, m, sig_u \rangle$. We describe each item in turn.

1. $\mathsf{H}(b_{u,i-1})$ is the hash pointer to the previous block.
2. seq_u is the sequence number which should equal i.
3. $txid$ is the transaction identifier, it should be generated using a cryptographically secure random number generator by the initiator of the transaction.
4. pk_v is the public key of the counterparty v.
5. m is the transaction message.
6. sig_u is the signature created using sk_u on the concatination of the binary representation of the five items above.

The fact that we have no constraint on the content of m is in alignment with our design goal—application neutrality.

TX blocks come in pairs. In particular, for every block

$$t_{u,i} = \langle \mathsf{H}(b_{u,i-1}), seq_u, txid, pk_v, m, sig_u \rangle$$

there exist one and only one *pair*

$$t_{v,j} = \langle \mathsf{H}(b_{v,j-1}), seq_v, txid, pk_u, m, sig_v \rangle.$$

Note that the $txid$ and m are the same, and the public keys refer to each other. Thus, given a TX block, these properties allow us to identify its pair.

3.3.2 Checkpoint Block

The CP block is a five-tuple, i.e. $c_{u,i} = \langle \mathsf{H}(b_{u,i-1}), seq_u, \mathsf{H}(\mathcal{C}_r), r, sig_u \rangle$, where $\mathcal{C}_r$ is the consensus result (which we describe in Section 3.3.3) in round r, the other items are the same as the TX block definition. Note that unlike in our prior work [22], CP blocks and TX blocks do not have independent sequence numbers.

The genesis block in the chain must be a CP block in the form of $c_{u,0} = \langle \mathsf{H}(\perp), 0, \mathsf{H}(\perp), 0, sig_u \rangle$ where $\mathsf{H}(\perp)$ can be interpreted as applying the hash function on an empty string. The genesis block is unique because every node has a unique public and private key pair.

3.3.3 Consensus Result

Our consensus protocol runs in rounds as discussed in Section 3.1. Every round is identified by a round number r, which is incremented on every new round. The consensus result is a tuple, i.e. $\mathcal{C}_r = \langle r, C \rangle$, where C is a set of CP blocks agreed by the facilitators of round r.

3.3.4 Chain Properties

Here we define a few important properties which results from the interleaving nature of CP and TX blocks.

If there exist a tuple $\langle c_{u,a}, c_{u,b} \rangle$ for a TX block $t_{u,i}$, where

$$a = \underset{k, k<i, \mathtt{typeof}(b_{u,k})=\gamma}{\arg\min} (i-k)$$

$$b = \underset{k, k>i, \mathtt{typeof}(b_{u,k})=\gamma}{\arg\min} (k-i),$$

then $\langle c_{u,a}, c_{u,b} \rangle$ is the *enclosure* of $t_{u,i}$. Note that $c_{u,a}$ is the more recent CP block. Also, some TX blocks may not have any subsequent CP blocks, then its enclosure is $\perp$.

If the enclosure of some TX block is $\langle c_{u,a}, c_{u,b} \rangle$, then its *fragment* is defined as $\{b_{u,i} : a \leq i \leq b\}$. For convenience, the function $\mathsf{fragment}(\cdot)$ represents the fragment of some TX block if it exists, otherwise $\perp$.

Agreed enclosure is the same as enclosure with an extra constraint where the CP blocks must be in some consensus result $\mathcal{C}_r$ and also must be the

smallest enclosure. That is, suppose a chain is in the form $\{c_i, c_{i+1}, t_{i+2}, c_{i+3}\}$[4] and c_i, c_{i+1}, c_{i+3} are in $\mathcal{C}_r, \mathcal{C}_{r+1}, \mathcal{C}_{r+3}$ respectively, then the agreed enclosure of t_{i+2} is $\langle c_{i+1}, c_{i+3} \rangle$ and cannot be $\langle c_i, c_{i+3} \rangle$. Similarly, *agreed fragment* has the same definition as fragment but using agreed enclosure. We define its function to be $\mathsf{agreed\_fragment}(\cdot)$ which we use later in the validation protocol (Section 3.6).

3.4 Consensus Protocol

Our consensus protocol runs on top of Extended TrustChain. It is directly related to the creation of CP blocks. The objectives of the protocol are to allow honest nodes always make progress (in the form of creating new CP blocks), compute correct consensus result in every round and have unbiased election of facilitators. Concretely, we define the necessary properties as follows.

Definition 1. Properties of the consensus protocol
$\forall r \in \mathbb{N}$, *the following properties must hold.*

1. Agreement*: If one correct node outputs a list of facilitators $\mathcal{F}_r$, then every node outputs $\mathcal{F}_r$*
2. Validity*: If any correct node outputs $\mathcal{F}_r$, then*
   (a) *$|\mathcal{C}_r| \geq N - t$ must hold for the $\mathcal{C}_r$ which was used to create $\mathcal{F}_r$,*
   (b) *$\mathcal{F}_r$ must contain at least $n - t$ honest nodes and*
   (c) *$|\mathcal{F}_r| = n$.*
3. Fairness*: Every node with a CP block in $\mathcal{C}_r$ should have an equal probability of becoming a member of $\mathcal{F}_r$.*
4. Termination*: Every correct node eventually outputs some $\mathcal{F}_r$.*

These properties may look like Byzantine consensus properties (which we describe next in Section 3.4.1) but they have some subtle differences. Firstly, they are properties for every node in the network and not just the facilitators. Secondly, they must be satisfied for all rounds because the whole system falls apart if one of the property cannot be satisfied in one of the rounds.

Before describing the protocol in detail, we take a brief detour to give background on the asynchronous subset consensus. This is the primary building block of our protocol.

[4] Usually the notation is of the form $c_{u,i}$, but the node identity is not important here so we simplify it to c_i

3.4.1 Background on Asynchronous Subset Consensus

The best way to explain asynchronous subset consensus (ACS) is to contrast it with the typical Byzantine consensus. We adapt the description from [28, Chapter 17].

Definition 2. Byzantine Consensus
There are n nodes, of which at most t might experience Byzantine fault. Node i starts with an input value v_i. The nodes must decide for one of those values, satisfying the the following.

1. Agreement: *If a correct node outputs v, then every node outputs v.*
2. Validity: *The decision value must be the input value of a node.*
3. Termination: *All correct nodes terminate in finite time.*

A node under Byzantine fault means that it can have arbitrary behaviour. For example not sending message or colluding with other Byzantine nodes to undermine the entire system. Note that the decision is on a single value. This is in contrast to ACS which we describe next.

ACS shares many similarities with Byzantine consensus. But it is an especially useful primitive for blockchain systems. It allows any party to propose a value and the result is the set union of all the proposed values by the majority. This is the primary difference with Byzantine consensus. Concretely, ACS needs to satisfy the following properties (adapted from [15]).

Definition 3. Asynchronous Subset Consensus
There are n nodes, of which at most t might experience Byzantine fault. Node i starts with an non-empty set of input values C_i. The nodes must decide an output C, satisfying the following.

1. Agreement: *If a correct node outputs C, then every node outputs C.*
2. Validity: *If any correct node outputs a set C, then $|C| \geq n - t$ and C contains the input of at least $n - 2t$ nodes.*
3. Totality: *If $n - t$ nodes receive an input, then all correct nodes produce an output.*

ACS has the nice property of censorship resiliance when compared to other consensus algorithms. For instance, Hyperledger and Tendermint uses Practical Byzantine Fault Tolerance (PBFT) [6] as their consensus algorithm. In PBFT, a leader is elected, if the leader is malicious but follows the protocol, then it can selectivly filter transactions. In contract, every party in ACS are involved in the proposal phase, and it guarantees that if $n - 2t$ parties propose the same transaction, then it must be in the agreed output. Thus, if some value is submitted to at least $n - 2t$ nodes, it is guaranteed to be

in the consensus result. For a detailed description of ACS we refer to the HoneyBadgerBFT work [15].

The main drawback with ACS and also Byzantine consensus algorithms is the high message complexity. Typically, such protocols have a message complexity of $O(n^2)$, where n is the number of parties. Hence, it may work with a small number of nodes, but it is infeasible for blockchain systems where thousands of nodes are involved.

3.4.2 Bootstrap Phase

Now we have all the necessary information to describe our consensus protocol. We begin with the bootstrap phase and then move onto the actual consensus phase.

Recall that facilitators are computed from the consensus result, but the consensus result is agreed by the facilitators. Thus we have a dependency cycle. The goal of the bootstrap phase is to give us a starting point in the cycle.

To bootstrap, imagine that there is some bootstrap oracle, that initiates the code on every node. The code satisfied all the properties in Definition 1. Namely every node has the same set of valid facilitators $\mathcal{F}_1$ that are randomly choosen. This concludes the bootstrap phase.

In practice, the bootstrap oracle is most likely the software developer and some of the desired properties cannot be achieved. In particular, it is not possible to have the fairness property because it is unlikely that the developer knows the identity of every node in advance.

3.4.3 Consensus Phase

The consensus phase begins when $\mathcal{F}_r$ is available to all the nodes. Note that $\mathcal{F}_r$ indicates the facilitators that were elected using result of round r and are responsible for driving the ACS protocol in round $r+1$. The goal is to reach agreement on a set of new facilitators $\mathcal{F}_{r+1}$ that satisfies the four properties in Definition 1.

There are two scenarios in the consensus phase. First, if node u is not the facilitator, it sends $\langle \texttt{cp\_msg}, c_{u,h} \rangle$ to all the facilitators. Second, if the node is a facilitator, it waits until it has received $N - t$ messages of type `cp_msg`. Invalid messages are removed, namely blocks with invalid signatures and duplicate blocks signed by the same key. With the sufficient number of `cp_msg` messages, it begins the ACS algorithm and some $\mathcal{C}'_{r+1}$ should be agreed upon by the end of it. Duplicates and blocks with invalid signatures are again removed from $\mathcal{C}'_{r+1}$ and we call the final result $\mathcal{C}_{r+1}$ We have the remove invalid blocks a second time (after ACS) because the adversary may send different CP blocks to different facilitators, which results in invalid blocks in the ACS output.

At the core of the consensus phase is the ACS protocol. While any ACS protocol that satisfies the standard definition will work, we use a simplification of HoneyBadgerBFT [15] as our ACS protocol because it is the only (to the best of our knowledge) consensus algorithms designed for blockchain systems. We do not use the full HoneyBadgerBFT due to the following. First, the transactions in HoneyBadgerBFT are first queued in a buffer and the main consensus algorithm starts only when the buffer reaches an optimal size. We do not have an infinite stream of CP blocks, thus buffering is unsuitable. Second, HoneyBadgerBFT uses threshold encryption to hide the content of the transactions. But we do not reach consensus on transactions, only CP blocks, so hiding CP blocks is meaningless for us as it contains no transactional information.

Continuing, when $\mathcal{F}_r$ reaches agreement on $\mathcal{C}_{r+1}$, they immediately broadcast two messages to all the nodes— first the consensus message $\langle \texttt{cons\_msg}, \mathcal{C}_{r+1} \rangle$, and second the signature message $\langle \texttt{cons\_sig}, r, sig \rangle$. The reason for sending `cons_sig` is the following. Recall that channels are not authenticated, and there are no signatures in $\mathcal{C}_{r+1}$. If a non-facilitator sees some $\mathcal{C}_{r+1}$, it cannot immediately trust it because it may have been forged. Thus, To guarantee authenticity, every facilitator sends an additional message that is the signature of $\mathcal{C}_{r+1}$.

Upon receiving $\mathcal{C}_{r+1}$ and at least $n-f$ valid signatures by some node u, u performs two asks. First, it creates a new CP block using $\mathsf{new\_cp}(\mathcal{C}_{r+1}, r+1)$ (Algorithm 2). Second, it computes the new facilitators using $\mathsf{get\_facilitator}(C, n)$ (Algorithm 1) and updates its facilitator list to the result. This concludes the consensus phase and brings us back to the beginning of the consensus phase.

Algorithm 1 Function $\mathsf{get\_facilitator}(C, n)$ takes a list of CP blocks C and an integer n, sort evey element in C by its luck value (the λ-expression), and outputs the smallest n elements.

$\mathsf{take}(n, \mathsf{sort\_by}(\lambda x.\mathsf{H}(x||pk \text{ of } x), C))$

Algorithm 2 Function $\mathsf{new\_cp}(\mathcal{C}_r, r)$ runs in the context of the caller u. It creates a new CP block and appends it to u's chain.

$h \leftarrow |B_u|$
$c_{u,h} \leftarrow \langle \mathsf{H}(b_{u,h-1}), h, \mathsf{H}(\mathcal{C}_r), r, sig_u \rangle$
$B_u \leftarrow B_u \cup c_{u,h}$

3.4.4 Relationship with α-Synchroniser

3.5 Transaction Protocol

The TX protocol, shown in Algorithm 4, is run by all nodes. It is also known as True Halves, first described by Veldhuisen [25, Chapter 3.2]. Node that wish to initiate a transaction calls $\mathsf{new\_tx}(pk_v, m, txid)$ (Algorithm 3) with the intended counterparty v identified by pk_v and message m. $txid$ should be a uniformly distributed random value, i.e. $txid \in_R \{0,1\}^{256}$. Then the initiator sends $\langle \texttt{tx\_req}, t_{u,h} \rangle$ to v.

Algorithm 3 Function $\mathsf{new\_tx}(pk_v, m, txid)$ generates a new TX block and appends it to the caller u's chain. It is executed in the private context of u, i.e. it has access to the sk_u and B_u. The necessary arguments are the public key of the counterparty pk_v, the transaction message m and the transaction identifier $txid$.

$h \leftarrow |B_u|$
$t_{u,h} \leftarrow \langle \mathsf{H}(b_{u,h-1}), h, txid, pk_v, m, sig_u \rangle$
$B_u \leftarrow B_u \cup \{t_{u,h}\}$

Algorithm 4 The TX protocol which runs in the context of node u.

Upon $\langle \texttt{tx\_req}, t_{v,j} \rangle$ from v
  $txid, pk_v, m \leftarrow t_{v,j}$ ▷ unpack the TX
  $\mathsf{new\_tx}(pk_u, m, txid)$
  store $t_{v,j}$ as the pair of $t_{u,h}$
  send $\langle \texttt{tx\_resp}, t_{u,h} \rangle$ to v
Upon $\langle \texttt{tx\_resp}, t_{v,j} \rangle$ from v
  $txid, pk_v, m \leftarrow t_{v,j}$ ▷ unpack the TX
  store $t_{v,j}$ as the pair of the TX with identifier $txid$

A key feature of the TX protocol is that it is non-blocking. At no time in Algorithm 3 or Algorithm 4 do we need to hold the chain state and wait for some message to be delivered before committing a new block to the chain. This allows for high concurrency where we can call $\mathsf{new\_tx}(\cdot)$ multiple times without waiting for the corresponding `tx_resp` messages.

3.6 Validation Protocol

Up to this point, we do not provide a mechanism to detect forks or other forms of tampering or forging. The validation protocol aims to solve this issue. The protocol is also a request-response protocol, just like the transaction protocol. But before explaining the protocol itself, we first define what it means for a transaction to be valid.

3.6.1 Validity Definition

A transaction can be in one of three states in terms of validity—*valid*, *invalid* and *unknown*. Given a fragment $F_{v,j}$, the validity of the transaction $t_{u,i}$ is captured by the function $\mathsf{get\_validity}(t, F)$ (Algorithm 5). The first four conditions (up to Line 21) essentially check whether the fragment is the one that the verifier needs. If it is not, then the verifier cannot make any decision and return *unknown*. This is likely to be the case for new transactions because $\mathsf{agreed\_fragment}(\cdot)$ would be $\perp$. The next two conditions checks for tampering or missing blocks, if any of these misconducts are detected, then the TX is invalid.

Note that the validity is on a transaction (two TX blocks with the same *txid*), rather than on one TX block owned by a single party. It is defined this way because the malicious sender may create new TX blocks in their own chain but never send `tx_req` messages. In that case, it may seem that the counterparty, who is honest, purposefully ommitted TX blocks. But in reality, it was the malicious sender who did not follow the protocol. Thus, in such cases the whole transaction, identified by its *txid* is marked as invalid.

Further, the caller of $\mathsf{get\_validity}(t_u u, i, F_{v,i})$ is not necessarily u[5] Any node w may call $\mathsf{get\_validity}(t_u u, i, F_{v,i})$ as long as the caller w has an agreed fragment of $t_{u,i}$—$F_{u,i}$. $F_{u,i}$ may be readily available if $w = u$ or it may be from some other `vd_resp` message, which we describe next in the validation protocol.

3.6.2 Validation Protocol

With the validity definition, we are ready to construct a protocol for determining the validity of transactions. The protocol is a simple response and request protocol (Algorithm 6). If u wishes to validate some TX with ID *txid* and counterparty v, it sends $\langle \texttt{vd\_req}, txid \rangle$ to v. The desired properties of the validation protocol are as follows.

Definition 4. Consensus on transactions

1. Correctness*: The validation protocol outputs the correct result according to the aforementioned validity definition.*

2. Agreement*: If any correct node decides on the validity (except when it is* unknown*) of a transaction, then all other correct nodes are able to reach the same conclusion or* unknown*.*

3. Liveness*: Any valid transactions can be validated eventually.*

[5] In practice it often is because after completing the TX protocol the parties are incentivised to check that the counterparty "did the right thing".

Algorithm 5 Function get_validity($t_{u,i}, F_{v,j}$) validates the transaction $t_{u,i}$. $F_{v,j}$ is the corresponding fragment received from v.

We assume there exist a valid $F_{u,i}$, namely the agreed fragment of $t_{u,i}$. The caller is w, it may be u but this is not necessary.

$c_{v,a} \leftarrow \mathsf{first}(F_{v,j})$
$c_{v,b} \leftarrow \mathsf{last}(F_{v,j})$
if $c_{v,a}$ or $c_{v,b}$ are not in consensus **then**
    return *unknown*
end if ▷ v has agreed fragment

if $|F_{v,j}| > L$ **then**
    return *unknown*
end if ▷ fragment not too long

if sequence number in $F_{v,j}$ is correct (sequential) **then**
    if hash pointers in $F_{v,j}$ is wrong **then**
        return *unknown*
    end if
end if ▷ correct TrustChain structure

$c_{u,b} \leftarrow \mathsf{last}(F_{u,i})$
if $c_{u,b}$ is not created using the same $\mathcal{C}_r$ as $c_{v,b}$ **then**
    return *unknown*
end if ▷ correct consensus round

$txid, pk_v, m \leftarrow t_{u,i}$
if number of blocks of $txid$ in $F_{v,j} \neq 1$ **then**
    return *invalid*
end if ▷ TX exists

$txid', pk'_u, m' \leftarrow t_{v,j}$
if $m \neq m' \vee pk_u \neq pk'_u$ **then**
    return *invalid*
end if ▷ no tampering

return *valid*

Algorithm 6 Validation protocol

Upon $\langle \texttt{vd\_req}, txid \rangle$ from v
  $t_{u,i} \leftarrow$ the transaction identified by $txid$
  $F_{u,i} \leftarrow \mathsf{agreed\_fragment}(t_{u,i})$
  send $\langle \texttt{vd\_resp}, txid, F_{u,i} \rangle$ to v
Upon $\langle \texttt{vd\_resp}, txid, F_{v,j} \rangle$ from v
  $t_{u,i} \leftarrow$ the transaction identified by $txid$
  set the validity of $t_{u,i}$ to $\mathsf{get\_validity}(t_{u,i}, F_{v,j})$

We make two remarks. First, just like the TX protocol, we do not block at any part of the protocol. Second, suppose some $F_{v,j}$ validates $t_{u,i}$, then that does not imply that $t_{u,i}$ only has one pair $t_{v,j}$. Our validity requirement only requires that there is only one $t_{v,j}$ in the correct consensus round. The counterparty may create any number of fake pairs in a later consensus rounds. But these fake pairs only pollutes the chain of v and can never be validated because the round is incorrect.

3.7 Optimisations

(Move to future work?)

Up to this point, we have discussed our protocol in the context of the model and assumptions defined in Section 3.2. In this section, we remove a few assumptions and discuss how our architecture is adapted.

Gossip?

Churn?

Sybil?

Chapter 4

Analysis of Correctness and Performance

Up to this point we described our system specification in detail. Of course, specification along does not establish any truths. In this chapter, we analyse two aspects of our system. First we show it has the desired properties. That is, the properties in Definition 1 and Definition 4 should hold. Then we analyse the performance, especially the throughput, and show that it out performs classical blockchain systems.

4.1 Correctness in the Presense of Faults

Our first objective is to show that Definition 1 holds for our consensus protocol. Then, building on top of it, we show Definition 4 holds for the validation protocol. The resulting theorem shows that only using CP blocks in the consensus algorithm implies consensus on TX blocks, in other words, implicit consensus.

4.1.1 Analysis of the Consensus Protocol

We begin our analysis by establishing truths on the four properties in Definition 1, namely agreement, validity, fairness and termination. Using these results, we use mathematical induction to show that they hold for all rounds.

Lemma 1. *For an arbitrary round r if $\mathcal{F}_r$ is known by all correct nodes and one correct node outputs a list of facilitators $\mathcal{F}_{r+1}$, then all correct nodes output $\mathcal{F}_{r+1}$.*

Proof. The argument follows from the protocol description. Given that $\mathcal{F}_r$ is known, correct nodes will send CP blocks to all members in $\mathcal{F}_r$. The ACS algorithm starts independently whenever the facilitator has $N - t$ valid CP blocks (recall from Section 3.4.3 that invalid blocks are ones with an invalid

signature or has a duplicate signature). It cannot make progress until $n-t$ honest facilitators start algorithm, but this eventually happens because there are $N-t$ correct nodes and all correct facilitator eventually receives $N-t$ valid CP blocks. At the end of ACS, some $\mathcal{C}_{r+1}$ is created, and is broadcasted along with the signature of the facilitators. Due to the agreement property of ACS (Definition 3), every correct node should receive at least $n-t$ valid signatures on the agreed $\mathcal{C}_{r+1}$. Thus they use $\mathcal{C}_{r+1}$ to generate a new CP block and compute new facilitators. Since $\mathsf{get\_facilitators}(\cdot)$ is a deterministic algorithm and the input $\mathcal{C}_{r+1}$ is in agreement, the output $\mathcal{F}_{r+1}$ is also in agreement. □

Lemma 2. *For an arbitrary round r, if $\mathcal{F}_r$ is known by all correct nodes and any correct node outputs $\mathcal{F}_{r+1}$, then (a) $|\mathcal{C}_{r+1}| \geq N-t$ must hold for the $\mathcal{C}_{r+1}$ which was used to create $\mathcal{F}_{r+1}$, (b) $\mathcal{F}_{r+1}$ must contain at least $n-t$ honest nodes and (c) $|\mathcal{F}_{r+1}| = n$.*

Proof. The validity follows from the validity property of ACS and the definition of our model, namely $N \geq n+t$ and $n \geq 3t+1$. Given $\mathcal{F}_r$, since $N \geq n+t$, there is at least n nodes that would send their CP block to $\mathcal{F}_r$. From the validity property of ACS, we know the output must contain the input of at least $n-2t$ nodes. But $n-t$ facilitators must have received $N-t$ valid CP blocks, so $|\mathcal{C}_{r+1}| \geq N-t$, this proves (a). There are $n-t$ honest nodes in $\mathcal{F}_{r+1}$ follows from the model, this proves (b). Finally, since $N-t \geq (n+t)-t = n$ and $\mathsf{get\_facilitators}(\cdot)$ outputs n items, $|\mathcal{F}_{r+1}| = n$ and this proves (c). □

Lemma 3. *For an arbitrary round r, if $\mathcal{F}_r$ is known by all correct nodes then every node with a CP block in $\mathcal{C}_{r+1}$, should have an equal probability to be elected as a facilitator in $\mathcal{F}_{r+1}$.*

Proof. We have already established that $|\mathcal{C}_{r+1}| \geq N-t \geq n$ from Lemma 2. Then the proof directly follows from the random oracle model. Recall that the luck value is computed using $\mathsf{H}(\mathcal{C}_{r+1}, ||pk_u)$. Since pk_u is unique for every node that has a CP block in $\mathcal{C}_{r+1}$, the output of $\mathsf{H}(\cdot)$ is uniformly random. This effectively generates a random permutation so every node has the same probability of being in the top n for the ordered sequence, namely the output of $\mathsf{get\_facilitators}(\cdot)$. □

Lemma 4. *For an arbitrary round r, if $\mathcal{F}_r$ is known by all correct nodes then every correct node eventually outputs some $\mathcal{F}_{r+1}$.*

Proof. This follows directly from the properties of the channel (eventual delivery) and the termination property of ACS. That is, $\mathcal{F}_r$ eventually receives all the CP blocks required to begin ACS. ACS eventually terminates. Finally the results are eventually dissemminated to all the nodes. □

From Lemmas 1, 2, 3 and 4, we have shown that the 4 properties of Definition 1 holds when assuming the existance of some $\mathcal{F}_r$. Thus, to proof the whole of Definition 1, we need to proof these 4 properties under the universal quantifier. We do this using mathematical induction.

Theorem 1. *For all rounds, the consensus protocol satisfies agreement, validity, fairness and termination (Definition 1).*

Proof. We proof using mathematical induction.

In the base case, agreement, validity fairness and termination follows directly from the bootstrap protocol, due to the bootstrap oracle. Note that the result is $\mathcal{F}_1$, which indicates the facilitators that are agreed in round 1, who are responsible for driving the ACS protocol in round 2.

For the inductive step, we assume that the 4 properties hold in round r and prove that they also hold in round $r+1$. Using Lemmas 1, 2, 3 and 4, it directly follows from Modus Ponens that these properties hold for $r+1$. Due to the principals of mathematical induction, these properties hold for all r. □

4.1.2 Correctness of Validation

The consensus protocol (on CP blocks and facilitators) is the backbone for consensus on transactions. In this section we build on top of Theorem 1 to show that most (except liveness) properties in Definition 4 can be satisfied.

Lemma 5. *The validation protocol outputs the correct result according to the validity definition.*

Proof. The algorithm (Algorithm 5) is the validity definition. □

Theorem 2. *If any correct node decides on the validity (except when it is* unknown*) of a transaction, then all other correct nodes are able to reach the same conclusion or* unknown*.*

Proof. We proof by contradiction. Without loss of generality, for some transaction t with an agreed fragment F, node u decides *valid* but node v decides *invalid*. Then there exist a fragment $F' = \{\dots, t', c'\}$ which u received that contains a valid pair of t—t'. There also exist a fragment $F'' = \{\dots, t'', c''\}$ which v received that does not contain or contains an invalid pair—t''. In both cases, the $\mathsf{get\_validity}(\cdot)$ function must have reached Line 21. Due to Theorem 1, we have $c' = c''$, otherwise the result would be *unknown*. Since $c'(= c'') = \langle \mathsf{H}(t'), \dots \rangle$ we must have $\mathsf{H}(t') = \mathsf{H}(t'')$ and $t' \neq t''$ (because t'' is invalid). In other words, whoever sent F'' must be able to create some t'' that has the same digest as t'. But we assumed that the adversary can only perform polynomial-time algorithm, so in order to find t'' it needs to query the random oracle exponentially many times. Thus we have a contradiction and this completes the proof. □

Theorem 2 is our first major result. It shows that consensus on CP blocks would lead to consensus on TX blocks when the nodes are running the validation protocol. Just like in our prior work [22], we call this behaviour implicit consensus. One of the main advantages over running a consensus algorithm on all the transactions is that the rate of transaction is no longer dependent on the consensus algorithm—ACS. This enables horizontal scalability where adding new nodes would lead to higher global transaction rate. In addition, a convenient consequence Theorem 2 is unforgeability. That is, no polynomial time adversary is able to create two chains $F = \{\ldots, t, c\}$ $F' = \{\ldots, t', c\}$ with correct hash pointers and the same end of chain c.

4.1.3 Impossibility of Liveness

While Theorem 2 is a major result that allows significantly improved performance over traditional blockchain systems, it is not perfect. Now we show a negative result, where the liveness property of Definition 4 cannot be attained. Meaning that trasanctions with adversaries cannot always be validated.

Lemma 6. *There exist a valid transactions that cannot be validated eventually.*

Proof. We proof by providing a counterexample. Suppose nodes u and v correctly perfomed the TX protocol which resulted a transaction t. Then when u wants to validate t, it does so by sending `vd_req` message to v. v can act maliciously and ignore all `vd_req` message, thus t can never be validated. □

Although this is a negative result, it does not put the adversary in an advantageous position. If the adversary is observed to ignore validation requests, then the honest nodes may prefer not to transact with her in the future. Thus, to stay relevant in the system, the adversary need to comply to the protocol.

4.2 Performance

This section aims to (analytically) answer our research question. That is, does the global throughput increase linearly with respect to the population size? We begin by looking at the communication and time complexity of ACS, and then the bandwidth requirement for a single transaction. These are prerequisites of the global throughput analysis.

4.2.1 Communication Complexity of ACS

The communication complexity of ACS is $O(n^2|v| + \lambda n^3 \log n)$ [15], where $|v|$ is the size of largest message and λ is the security parameter. Note

that the security parameter is the same as the one for our random oracle described in Section 3.2. In particular, it is from the use of $\mathsf{H}(\cdot)$ in the reliable broadcast phase in ACS. In our system, we wish to understand the scalability properties. Thus we consider the complexity as a function of N rather than n or λ. Since $|v|$ is at most all the CP blocks from every node, we have $|v| = cN$, where c is a constant representing the size of one CP block. Therefore the communication complexity of ACS in our system is $O(N)$. Since we use a constant n, $O(N)$ communication complexity also holds for a single facilitator.

4.2.2 Time Complexity of ACS

In order to make arguments on bandwidth or throughput, which are concepts that depend on time, we must make some assumptions on the time complexity of ACS. The time complexity we use here is not the same as what is typically used in distributed systems. In analysis of distributed systems, time complexity is often in number of rounds. For example, ACS runs in a constant number of rounds because its subprotocols—reliable broadcast and binary Byzantine consensus—also run in a constant number of rounds. However, in practice, making a unit of communication always has some overhead associated with it, for example serialising and writing it to some network socket. Hence, for the remainder of our performance analysis, add the following to our computational model. For every unit of communication, we assume they take some non-negligible but constant time to perform. From our result in Section 4.2.1 and the fact that ACS runs in a constant number of rounds, it follows that ACS has a time complexity of $O(N)$.

4.2.3 Bandwidth Requirement for Transactions

With our assumption on the time complexity of ACS, we are ready to analyse the bandwidth. We know that to create and then validate a transaction, the bandwidth required is of $O(l)$, where l is the length of the agreed fragment. This can be seen from the fact that the largest message by far is the `vd_resp` message, which contains the agreed fragment, the other messages (`tx_req`, `tx_resp` and `vd_req`) are constant factors. If we assume that every node performs transactions at a constant rate of r_{tx} per second. Then

$$l = r_{\text{tx}} D_{\text{acs}},$$

where D_{acs} is the duration an instance of ACS. But from Section 4.2.2, we know that D_{acs} is of $O(N)$, thus the bandwidth per transaction is $O(N)$. This is intuitive because consensus duration would be longer if there are more CP blocks, which means that the agreed fragments are longer (assuming nodes transact at a constant rate). This behaviour is also verified experimentally in Chapter 5.

4.2.4 Global Throughput

Using our results so far, we are able to analyse the global throughput. Suppose every node has some fixed bandwidth capacity C (unit of communication per second), they make transactions at r_{tx} per second. Then we have the inequality $C \geq r_{\text{tx}} l$, where l is the length of the the agree fragment as before. Rearranging, we get

$$\frac{C}{l} \geq r_{\text{tx}}.$$

With this, we consider two cases, first is when all the nodes are running at maximum capacity, i.e. $\frac{C}{l} = r_{\text{tx}}$. Recall that l is of $O(N)$, so as the the population increases, the transaction rate must decrease in order to maintain the inverse relationship, thus r_{tx} is of $O(N^{-1})$. Therefore, the global throughput would be $O(N^{-1})N = O(1)$. In the second case, nodes are not running at maximum capacity and r_{tx} is maintained. Therefore, if every node runs at r_{tx}, the global throughput becomes $O(N)$ until N is too large and we go back to the first case. A visualisation of the result is shown in Figure 4.1.

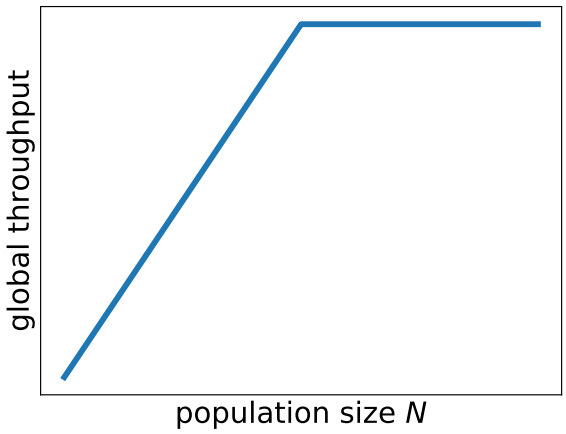


Figure 4.1: Visualisation of the worst case global throughput.

The upshot of this analysis is that our system scales nicely at a global throughput of $O(N)$ until the population gets too large. Then we maintain a constant global throughput. This result falls a bit short of what we envisioned in the introduction. However, the population is not the number of nodes that use the system, but the number of nodes that are online during a single round. Furthermore, this result is for the worst case where every transaction needs an agreed fragment to be transmitted. In practice, nodes are able to cache agreed fragments. For instance, if u and v make x transactions in a single round, then only one agreed fragment need to be exchanged

as it contains all the transactions rather than x agreed fragments.

4.3 Effect of A Highly Adverserial Environment

Our last study consideres the effect when the number of adversaries is more than t. This is useful because in practice it is difficult to guarantee that t satisfied $n \geq 3t + 1$, especially when N is large. Hence we are interested in the probability for this to happen under our facilitator election process.

The problem can be formulated as follows. Suppose an urn contains N balls, t are black and $N - t$ are white. If n balls are drawn uniformly at random without replacement, what is the probability that $\lfloor \frac{n-1}{3} \rfloor$ are black? The random variable X in this case is the number of black balls, or the number of successful events. It follows the hypergeometric distribution since we pick balls *without* replacement [24]. Hence, we are interested in the following probability.

$$1 - \sum_{k=0}^{\lfloor \frac{n-1}{3} \rfloor} \Pr[X = k] = 1 - \sum_{k=0}^{\lfloor \frac{n-1}{3} \rfloor} \frac{\binom{t}{k}\binom{N-t}{n-k}}{\binom{N}{n}}$$

This is not in closed form, but we can visualise the effect in Figure 4.2. We set the population size N to 2000 and plot the probability of more than $\lfloor \frac{n-1}{3} \rfloor$ successful events for different numbers of draws. Evidently, if the number of black balls (traitors) is a third of the population (666 out of 2000) we have about 0.5 probability of electing more than $\lfloor \frac{n-1}{3} \rfloor$ black balls for sufficiently large n. Thus, we cannot expect the system to function correctly when the expected value is close to the number of black balls that we can tolerate.

On the other hand, due to the fact that hypergeometric distributions have light tails, that is "faster-than-exponential fall-off" [24], the probability for picking more than $\lfloor \frac{n-1}{3} \rfloor$ black balls when the expected value is much smaller than $\lfloor \frac{n-1}{3} \rfloor$ is small. For example, we can use tail inequality to bound the probability of picking more than $\lfloor \frac{n-1}{3} \rfloor$ black balls when only 5% are black. The tail inequality is

$$\Pr[X \geq E[X] + \tau n] \leq e^{-2\tau^2 n},$$

where $E[X] = n/20$. We are interested in $\Pr[X \geq \lfloor \frac{n-1}{3} \rfloor + 1]$, so

$$\begin{aligned} \tau &= \frac{\lfloor \frac{n-1}{3} \rfloor + 1}{n} - \frac{1}{20} \\ &> \frac{\lfloor \frac{n-1}{3} \rfloor}{n} - \frac{1}{20} \\ &\geq \frac{1}{4} - \frac{1}{20} = \frac{1}{5} \quad \text{for } n \geq 4. \end{aligned}$$

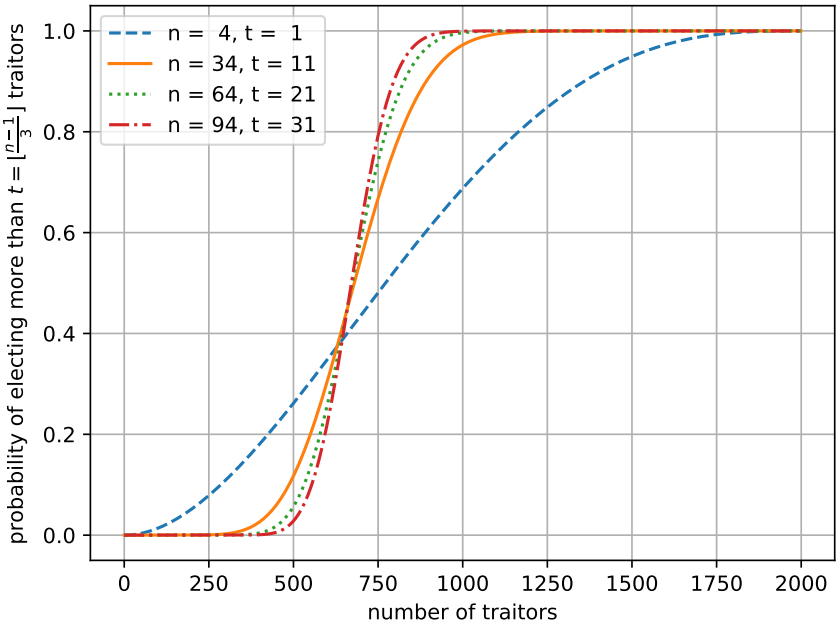


Figure 4.2: Plot of the probability of selecting more than $\lfloor \frac{n-1}{3} \rfloor$ black balls for different numbers of

Putting τ back into the tail inequality we get

$$\Pr[X \geq \lfloor \frac{n-1}{3} \rfloor + 1] \leq e^{-\frac{2}{25}n}.$$

The bound is not tight, but it is clear that for sufficiently large n, the probability becomes vanishingly small. Hence, it is possible to use our system in practice when 5% of the population are adversarial but $n \geq 3t+1$ does not hold, as long as n is large.

Chapter 5

Implementation and Experimental Results

Thus far, we have only discussed our system from a theoretical perspective. Henceforth, we evaluate out system experimentally and compare the results with the theoretical analysis. We begin this chapter by a description of the implementation in Section 5.1. Then, we move on to describing our experimental setup in Section 5.2. Finally, Section 5.3 presents our experimental results and our evaluation. Our experiment primarily focuses on the consensus duration and the throughput.

5.1 Implementation

The prototype implementation is done in the event driven paradigm, using the Python programming language. We use Twisted as our networking library. The code can be found on GitHub[1].

The structure of the implementation is primarily made up of three modules—`acs`, `trustchain` and `node`. `acs`, as its name suggests, implements ACS. We use liberasurecode[2] as the Reed-Solomon error correcting code library, used in RBC. An implementation detail is that liberasurecode cannot create more than 32 code blocks[3]. The `acs` module provides a small interface to the caller to start and stop the consensus process and also receive results. The `trustchain` module implements the Extended TrustChain data structure. It also provides the essential algorithm necessary to interact with Extended TrustChain such as $\mathsf{new\_tx}(\cdot)$, $\mathsf{new\_cp}(\cdot)$, $\mathsf{agreed\_fragment}(\cdot)$ and so on. Since this is a prototype implementation, we only store the data struc-

[1] https://github.com/kc1212/consensus-thesis-code

[2] https://github.com/openstack/liberasurecode

[3] The 32 code blocks limitation is hardcoded in the source file, see https://github.com/openstack/liberasurecode/blob/0794b31c623e4cede76d66be730719d24debcca9/include/erasurecode/erasurecode.h#L35

ture in memory and not on disk. Finally, the `node` module ties everything together. It implements the the consensus phase, the transaction protocol and the validation protocol.

Every node keeps a persistent TCP connection with every other node. This creates a fully connected network for our experiment. It is certainly not idea in real world scenarios where nodes may have limited resources. But as a prototype, it is sufficient to create a system that has just over a thousand nodes, which is enough for us to experiment on.

5.2 Experimental Setup

There are two asppects of the experimental setup. First is the nature of the experiment. Second is the physical setup. We describe these in turn.

The nature and the goal of the experiment is to run the three protocols—consensus protocol, transaction protocol and validation protocol—simultaneously and analyse the throughput and consensus duration. There are two types of parameters which we must consider. First are the fixed parameters r_{tx} and D. These are selected from emperical evidence, we found that $D = 30$ seconds to be more than enough for all CP blocks to reach consensus. We also found that using $r_{\text{tx}} = 8$ gave us good throughput without putting too much demand on the bandwidth which is must also be used for ACS. These parameters are fixed for all our experiments. The second set of parameters can be seen as the domain, and we run our experiment for every combination of these parameters. Concretely, there are three of them. The number of facilitators $n \; in \{8, 12, \ldots, 32\}$. The population size $N \in \{100, 200, \ldots, 1200\}$. Finally the two modes of transaction. The first mode or the ideal mode is that nodes only transact with their immediate neighbour. This minimises the bandwidth required per validated transaction because agreed fragment can be cached. The second mode is in the other extreme, where every transaction is with a random node out of the N nodes in the system, thus the agreed fragment is unlikely to be cached. Unfortunately, the maximum n is 32 because the limitation in liberasurecode mentioned in Section 5.1. N stops at 1200 is due to our physical setup, which we describe next.

The experiment is run on the DAS-5 (The Distributed ASCI Supercomputer 5). From now on, we use "machines" to refer to DAS-5 nodes and nodes to refer to a running instance in our system. On DAS-5 we use up to 24 machine, for each machine we use 50 nodes. This gives us the aforementioned 1200 number. With this setup, we cannot run more nodes because the every machine only has 65535 ports available (some of them are reserved). But 50 nodes each need 1200 TCP connections which is 60000 TCP connections per machine. Thus we set N to be at most 1200[4].

[4]While it is possible to have many more TCP connections per machine, but it requires additional network interface which is something we do not control on the DAS-5.

To coordinate nodes on many different machine, we employ a discovery server to inform every node the IP addresses and port numbers of every other node. It is only run before the experiment and is not used during the experiment.

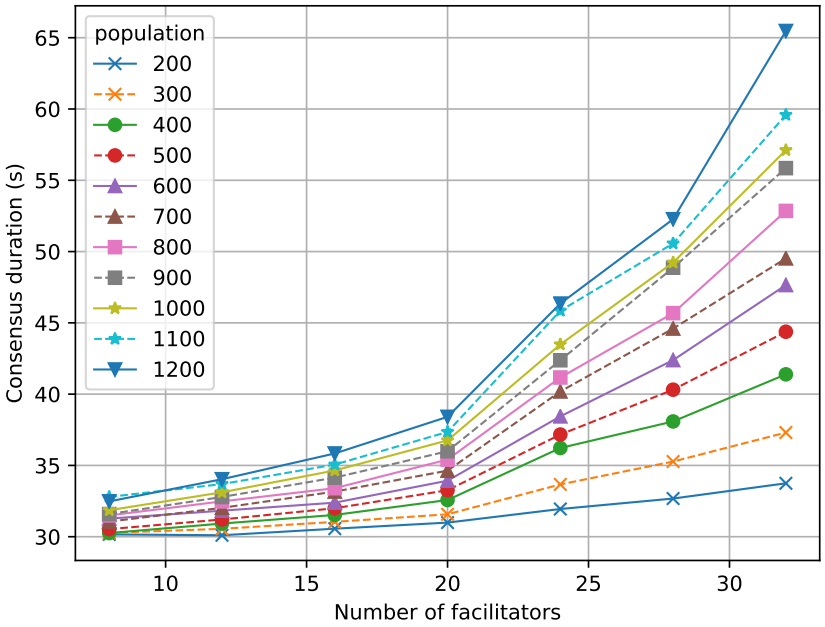
5.3 Evaluation

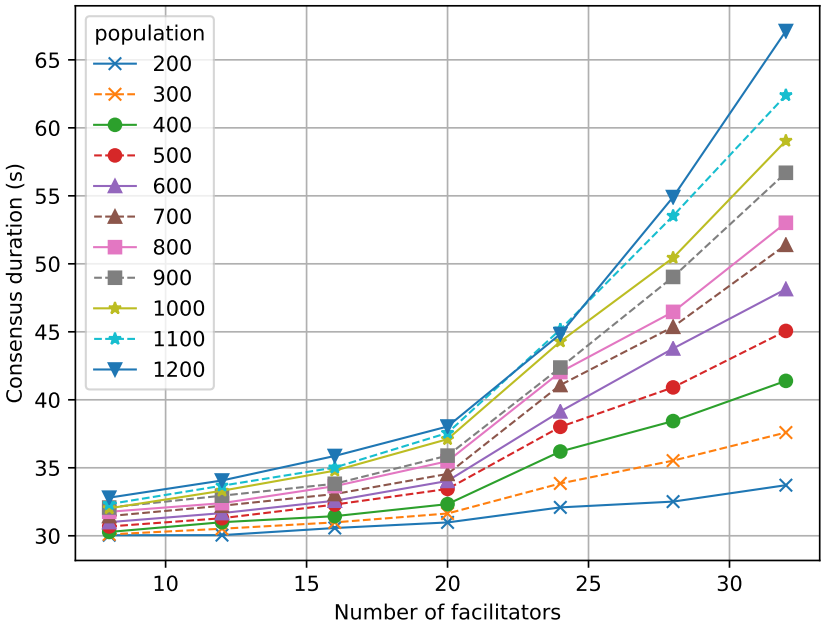
5.3.1 Consensus Duration

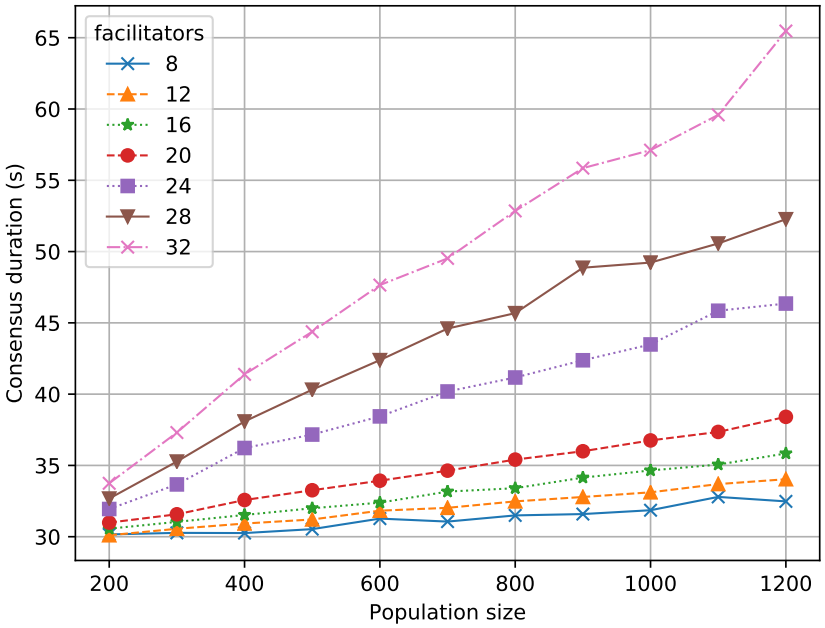
5.3.2 Global Throughput

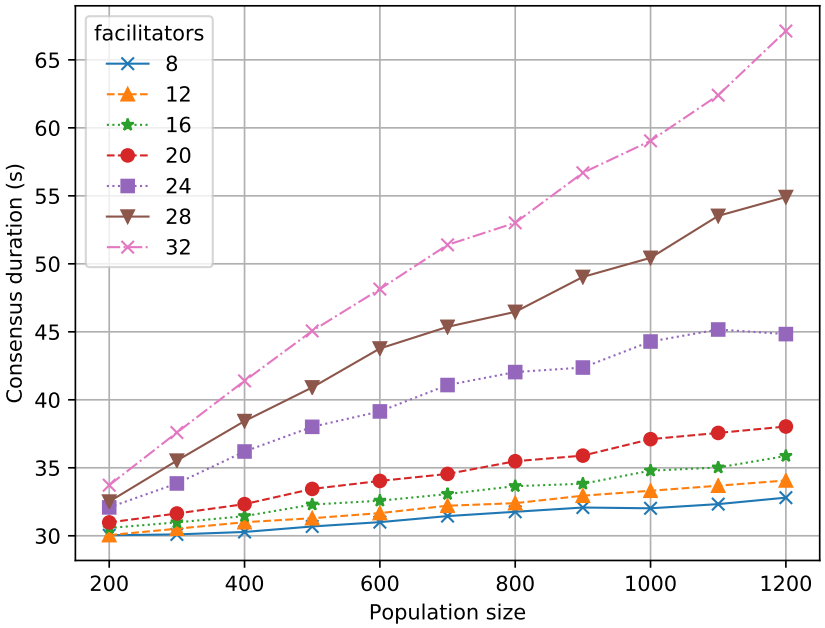
5.4 Evaluation

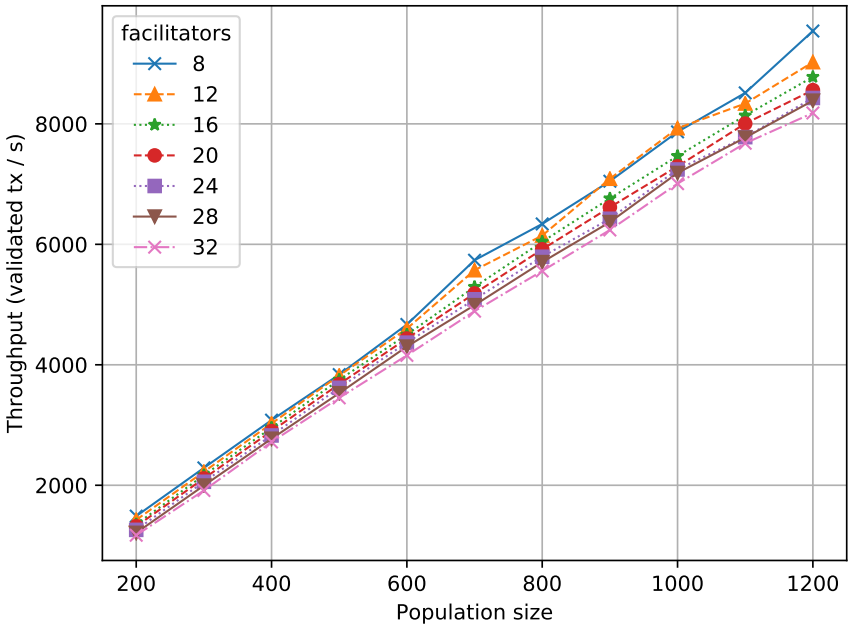
- How fast is the consensus algorithm? Possibly plot graph of time versus the number of nodes.
- Does the promoter registration phase add a lot of extra overhead?
- What's the rate of transaction such that they can be verified "on time", i.e. without a growing backlog?
- Our global validation rate is somewhat equivalent to the transaction rate in other systems. Does the validation rate scale with respect to the number of nodes? In theory it should. Plot validation rate vs number of nodes, we expect it to be almost linear.

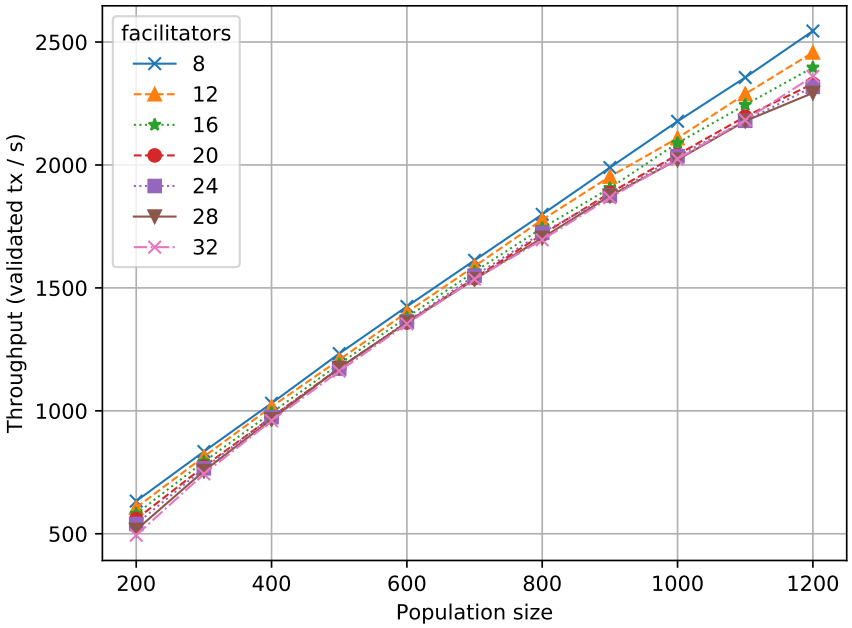

population
200
300
400
500
600
700
800
900
1000
1100
1200
Consensus duration (s)
Number of facilitators
30
35
40
45
50
55
60
65
10
15
20
25
30


population
200
300
400
500
600
700
800
900
1000
1100
1200
Consensus duration (s)
Number of facilitators
30
35
40
45
50
55
60
65
10
15
20
25
30


facilitators
8
12
16
20
24
28
32
Consensus duration (s)
Population size
30
35
40
45
50
55
60
65
200
400
600
800
1000
1200


facilitators
8
12
16
20
24
28
32
Consensus duration (s)
Population size
30
35
40
45
50
55
60
65
200
400
600
800
1000
1200


facilitators
8
12
16
20
24
28
32
Throughput (validated tx / s)
2000
4000
6000
8000
200
400
600
800
1000
1200
Population size


facilitators
8
12
16
20
24
28
32
Throughput (validated tx / s)
500
1000
1500
2000
2500
200
400
600
800
1000
1200
Population size

Chapter 6

Related Work

Having analysed our system both theoretically and experimentally, we dedicate this chapter on comparing our results with related work. Blockchain technology has seen a surge in recent years from both the industry and academia. We classify the various blockchain systems by their consensus approach and divide them into the following categories:

1. classical blockchain systems,
2. classical blockchain with offchain transactions,
3. permissioned systems,
4. hybrid systems and
5. blockchains without global consensus.

A few systems from each of these categories are compared with our design.

6.1 Classical Blockchain Systems

This category represent systems with a probabilistic probabilistic consensus algorithm. That is, transactions never reach consensus with a probability of 1. The typical examples are proof-of-work based systems such as Bitcoin, Ethereum and other Altcoins. In Bitcoin, the level of consensus of a block[1] is determined how deep it is in the Bitcoin blockchain, also called the number of confirmation. The probability of a block being orphaned drops exponentially as the depth increases [16]. Nevertheless, the probability of the highest block being orphaned is non-negligible. The advantage of this type of consensus is that it can be used in a large network and is reasonably secure. Attackers can not out pace honest users in finding new blocks unless they have a

[1]Note that a block in Bitcoin contain many transactions whereas our TX block only contain a single transaction.

majority of the hash power. The disadvantage however is that transactions are never in consensus with a probability of 1—no consensus finality. Also, the performance is limited due to the fact that blocks are of fixed size and are generated at fixed intervals.

Our system significantly improves upon Bitcoin and other classical blockchain systems in performance, scalability and consensus finality. The results described in Chapter 4 and Chapter 5, show that we have horizontal scalability, where more nodes result in more global throughput. Further, we do not have the aforementioned probabilistic behaviour, once some consensus result is decided, it cannot be orphaned, thus our consensus is final. The leadership election is also not ideal in classical blockchain systems. Mining can be seen as a technique to elect a single leader. The leader has full control of what goes into the block thus it may selectively censor transactions. We use ACS, so as long as the CP block is in $n - 2t$ nodes, it is guaranteed to be in consensus.

However, the security aspect falls short of the "honest majority" security model that classical blockchains claim to have[2]. Our system risks going into erroneous state if the inequality $n \geq 3t + 1$ is not satisfied. Classical blockchain systems also have an incentive mechanism, thus they do not depend on altruistic nodes and encourages participation. Our system on the other hand does not have an incentive mechanism because we make no assumptions on the application.

6.2 Offchain Transactions

Offchain transactions make use of the fact that, if two or more parties frequently make transactions, then it is not necessary to store every transaction on the blockchain, only the net settlement is necessary. The best examples are Lightning Network [19] and full duplex channels [9]. These use the Bitcoin blockchain to store the net settlement and a payment channel to conduct offchain transactions.

Offchain transaction systems are implemented using multi-signature addresses [5] and hashed time-locked Bitcoin contracts [4]. If two parties wish to make transactions, they open a time-locked payment channel with a multi-signature Bitcoin address (for two parties it would be a 2-of-2 signature address). Transactions happen off the Bitcoin blockchain and are signed by both parties. These transactions have a timestamp and only contain the net amount since the start of the payment channel. Before the payment channel expires, the latest of such transactions is sent to the multi-signature address. If multiple transactions are sent to the payment channel, the latest one is used. After payment channel is closed, the net transaction is propagated to

[2]Recently it was shown that doing selfish-ming would give the adversary an unfair advantage when she only controls 25% of the mining power [11]

the Bitcoin blockchain.

The advantages of such systems is that they act as add-ons to Bitcoin wish already has a large number of users. Thus, if enough of the network wish for it (by setting a new block version), then a large number of users will instantly benefit from it. It also shares the advantages of Bitcoin such as security and incentives.

On the other hand, offchain transactions also suffers from the problem of Bitcoin. Proof-of-work is still problematic as it consumes an unreasonable amount of power. Further, sidechain transactions are limited only to an exchange of cryptocurrency, it is less general than typical Bitcoin transactions which may be a simple smart contract. Time-locked contracts have a strong dependency on timing, thus disputes may arise when the payment channel is just about to close. Our system is purely asynchronous and make no assumption on timing, in fact we assume the adversary has control of the message delivery time and order.

6.3 Permissioned Systems

This category of systems use Byzantine consensus algorithms (discussed in Section 3.4.1). In essence, they contain a fixed set of nodes, sometimes called validators, that run a Byzantine consensus algorithm to decide on new blocks. This is known as a permissioned system where the validators must be pre-determined. Some examples include Hyperledger and Tendermint. The consensus algorithm used in these two systems is PBFT.

A nice aspect of Byzantine consensus and in particular PBFT is that it can handle much more transactions than classical blockchain systems. But our system has the potential to perform beyond that of PBFT because we represent many transactions with a single CP block, enabling horizontal scalability. Furthermore, since PBFT relies on a leader, it is not censorship resiliant. Our system on the other hand has the benifits of ACS where CP blocks cannot be censored. Finally, our system is able to work in the permissionless setting by simply submitting new CP blocks to the facilitators. It can even be adapted to work in the permissioned by simply removing the luck value computation.

What we wish to have which is in Hyperledger is a smart contract system (also known as chaincodes in Hyperledger). We hope to design and implement smart contracts by adding additional logic to the transaction protocol and the validation protocol. Such functionalities we believe is better to be built into the backbone rather than having it as an add on.

6.4 Hybrid Systems

Hybrid systems are very recent inventions. Just like our work, they are attempts on solving the problems of traditional blockchains. The main characteristic of these systems is that they use a classical blockchain technique, i.e. proof-of-work, to elect a committee and prevent Sybils. And then they use a Byzantine consensus algorithm to actually reach consensus on a set of transactions within the committee. Some examples are SCP [14], ByzCoin [13] and Solidus [1].

Our approach share many similarities with the hybrid systems. First, we also elect a committee (facilitators) to drive consensus. But we do not have proof-of-work for Sybil defence because we believe it is possible to do it efficiently, e.g. using NetFlow [18]. Secondly, our use of CP blocks and ACS is also unique. This creates a much higher throughput and enables censorship resiliance as mentioned earlier. For instance, ByzCoin performs just below 1000 TPS with a thousand nodes whereas we peak at 8000 TPS.

A major side effect of these hybrid systems is that they cannot guarantee correctness when there is a large number of malicious nodes. Our system has the same issue. For SCP, ByzCoin and Solidus, they all have some probability to elect more than n Byzantine nodes into the committee. This problem is especially difficult solve because the committee is always much smaller than the population size which has more than t Byzantine nodes, thus electing more than t nodes into the committee in always a possibility. Classical blockchain do not have this problem because they do not use Byzantine consensus. The permissioned systems work around this problem by trusting validators.

6.5 Blockchains Without Global Consensus

Tangle [20], Corda [12] and the original TrustChain [**trustchain**] do not use global consensus at all. By avoiding global consensus, they are able to achieve extreme scalability. Just like our approach, these blockchains are also application neutral where transactions can contain arbitrary data.

Our system can be considered as the same as these types of blockchains but with a lightweight consensus protocol. Consensus might not be applicable in all applications. But we believe it is important for detecting and preventing fraud. The example in Figure 3.1 on page 9 demonstrates this. If b makes a fork and a and c have no way to communicate (e.g. the adversary may control parts of the network) , then c is tricked to believe that her transaction with b is valid. Only when c sees a conflicting chain is she able to tell that the transaction is invalid. But c does not know the true end-of-chain of b, thus she can never know whether her transaction is valid. This is not possible in our system because b cannot convince c unless he can compute

exponential time algorithms (this is finding the second preimage for a hash function).

However, consensus and validation comes at a cost that do not exist in Tangle, Corda or the original TrustChain. Our transaction rate is affected by the validation protocol in the worst case scenario as we saw in Section 5.3.

Chapter 7

Conclusion

Bibliography

[1] Ittai Abraham et al. "Solidus: An Incentive-compatible Cryptocurrency Based on Permissionless Byzantine Consensus". In: *arXiv preprint arXiv:1612.02916* (2016).

[2] Martin Neil Baily and Douglas J. Elliott. *The US Financial and Economic Crisis: Where Does It Stand and Where Do We Go From Here?* June 2009. URL: https://web.archive.org/web/20100602131359/http://www.brookings.edu/~/media/Files/rc/papers/2009/0615_economic_crisis_baily_elliott/0615_economic_crisis_baily_elliott.pdf (visited on 25/06/2017).

[3] Mihir Bellare and Phillip Rogaway. "Random oracles are practical: A paradigm for designing efficient protocols". In: *Proceedings of the 1st ACM conference on Computer and communications security*. ACM. 1993, pp. 62–73.

[4] Bitcoin Wiki. *Hashed Timelock Contracts*. Nov. 2016. URL: https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts (visited on 20/06/2017).

[5] Bitcoin Wiki. *Multisignature*. Jan. 2017. URL: https://en.bitcoin.it/wiki/Multisignature (visited on 20/06/2017).

[6] Miguel Castro, Barbara Liskov et al. "Practical Byzantine fault tolerance". In: *OSDI*. Vol. 99. 1999, pp. 173–186.

[7] CoinMarketCap. *CryptoCurrency Market Capitalizations*. June 2017. URL: https://coinmarketcap.com/currencies/bitcoin/ (visited on 25/06/2017).

[8] Kyle Croman et al. "On scaling decentralized blockchains". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2016, pp. 106–125.

[9] Christian Decker and Roger Wattenhofer. "A fast and scalable payment network with bitcoin duplex micropayment channels". In: *Symposium on Self-Stabilizing Systems*. Springer. 2015, pp. 3–18.

[10] John R Douceur. "The sybil attack". In: *International Workshop on Peer-to-Peer Systems*. Springer. 2002, pp. 251–260.

[11] Ittay Eyal and Emin Gün Sirer. "Majority is not enough: Bitcoin mining is vulnerable". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2014, pp. 436–454.

[12] Mike Hearn. *Corda: A distributed ledger*. Sept. 2016. URL: https://docs.corda.net/_static/corda-technical-whitepaper.pdf.

[13] Eleftherios Kokoris Kogias et al. "Enhancing bitcoin security and performance with strong consistency via collective signing". In: *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association. 2016, pp. 279–296.

[14] Loi Luu et al. "SCP: A Computationally-Scalable Byzantine Consensus Protocol For Blockchains." In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 1168.

[15] Andrew Miller et al. "The honey badger of BFT protocols". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 31–42.

[16] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008.

[17] Karl J O'Dwyer and David Malone. "Bitcoin mining and its energy footprint". In: (2014).

[18] Pim Otte. "Sybil-resistant trust mechanisms in distributed systems". MA thesis. Delft University of Technology, Dec. 2016. URL: http://resolver.tudelft.nl/uuid:17adc7bd-5c82-4ad5-b1c8-a8b85b23db1f.

[19] Joseph Poon and Thaddeus Dryja. "The bitcoin lightning network". In: (Jan. 2016). URL: https://lightning.network/lightning-network-paper.pdf.

[20] Serguei Popov. *The tangle*. Apr. 2016. URL: https://iota.org/IOTA_Whitepaper.pdf.

[21] Andrew Quentson. *While Bitcoin Price Hits Record Highs, Nearly 100,000 Transactions Are Stuck in a Backlog*. May 2017. URL: https://www.cryptocoinsnews.com/almost-100000-bitcoin-transactions-stuck-in-a-backlog-waiting-to-move/ (visited on 25/06/2017).

[22] Zhijie Ren et al. *Implicit Consensus: Blockchain with Unbounded Throughput*. 2017. eprint: arXiv:1705.11046.

[23] Laura Shin. *Bitcoin Is Mired In A Civil War. Can This Proposal Save It?* Apr. 2017. URL: https://www.forbes.com/sites/laurashin/2017/04/04/bitcoin-is-mired-in-a-civil-war-can-this-proposal-save-it (visited on 25/06/2017).

[24] M. Skala. "Hypergeometric tail inequalities: ending the insanity". In: *ArXiv e-prints* (Nov. 2013). arXiv: 1311.5939 [math.PR].

[25] Pim Veldhuisen. "Leveraging blockchains to establish cooperation". MA thesis. Delft University of Technology, May 2017. URL: http://resolver.tudelft.nl/uuid:0bd2fbdf-bdde-4c6f-8a96-c42077bb2d49.

[26] *Visa Inc. at a Glance*. 2015. URL: https://usa.visa.com/dam/VCOM/download/corporate/media/visa-fact-sheet-Jun2015.pdf (visited on 12/06/2017).

[27] Marko 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.

[28] Roger Wattenhofer. *Principles of Distributed Computing*. 2016. URL: http://dcg.ethz.ch/lectures/podc_allstars/lecture/podc.pdf.

[29] Haifeng Yu et al. "Sybilguard: defending against sybil attacks via social networks". In: *ACM SIGCOMM Computer Communication Review*. Vol. 36. 4. ACM. 2006, pp. 267–278.

Appendix A

Consensus Example

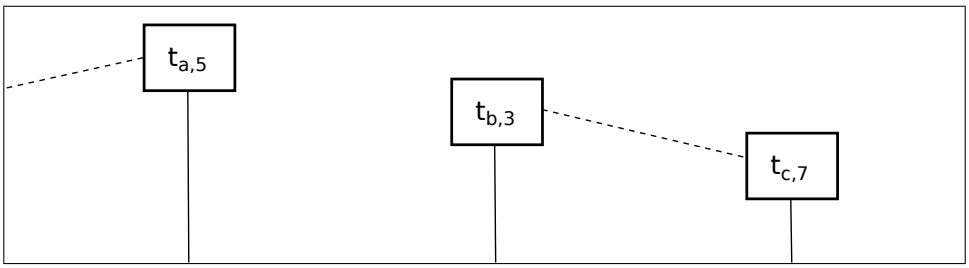


Figure A.1: Initial state

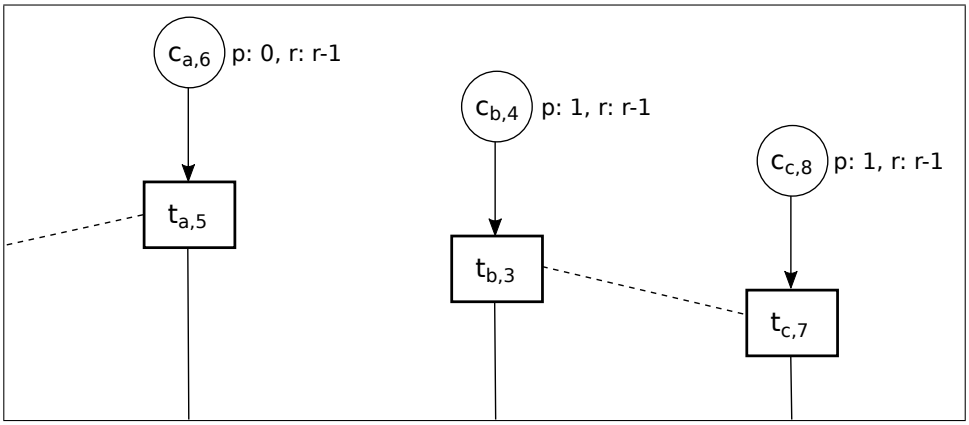


Figure A.2: Initial state

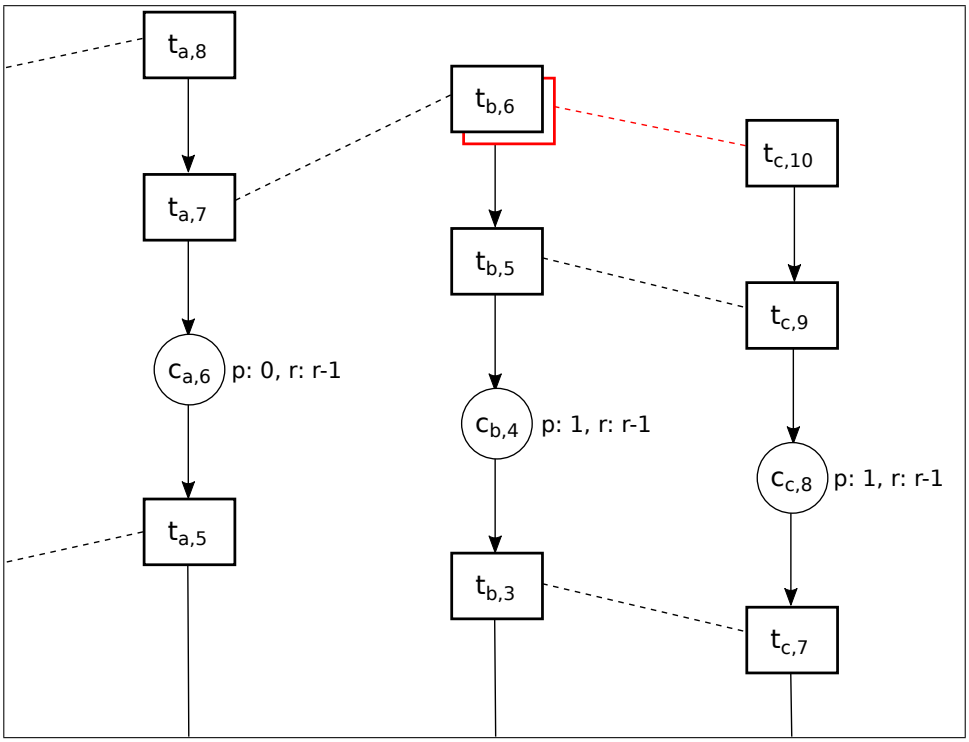


Figure A.3: Initial state

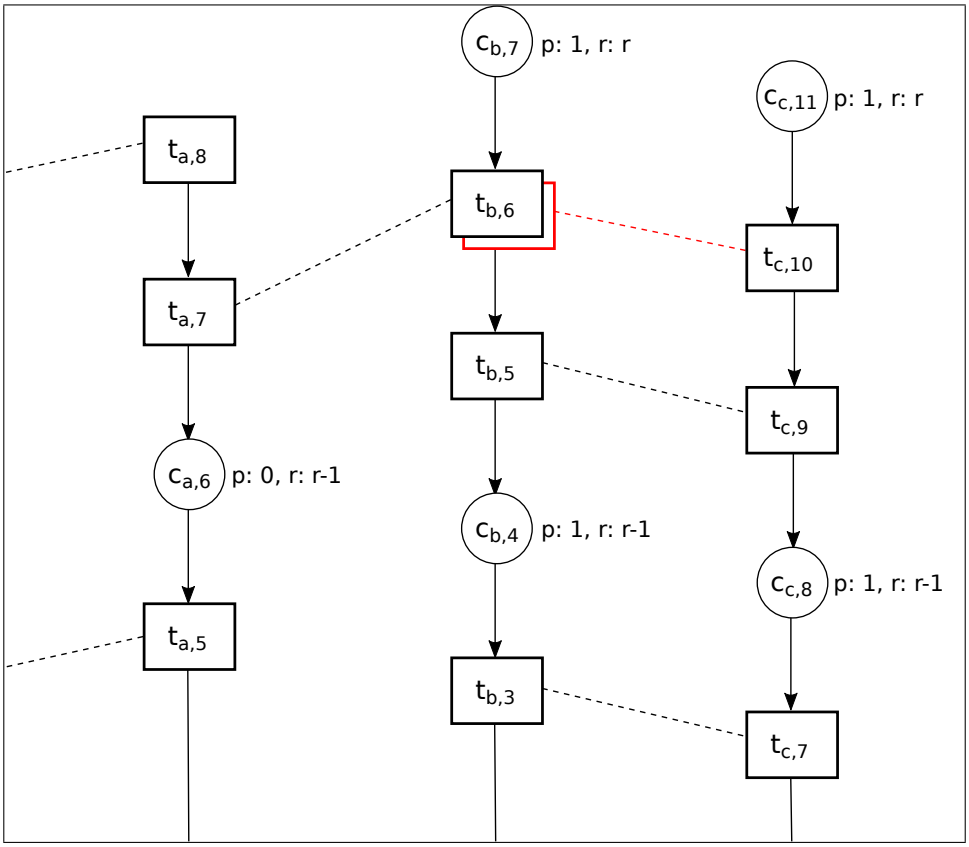


Figure A.4: Initial state