TODO TITLE

TODO AUTHOR

Delft
University of
Technology

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

TODO AUTHOR

12th June 2017

Author
TODO AUTHOR

Title
TODO TITLE

MSc presentation
TODO GRADUATION DATE

Graduation Committee
TODO GRADUATION COMMITTEE   Delft University of Technology

Abstract

TODO ABSTRACT

Preface

TODO MOTIVATION FOR RESEARCH TOPIC

TODO ACKNOWLEDGEMENTS

TODO AUTHOR

Delft, The Netherlands
12th June 2017

Contents

Chapter 1

Introduction

Blockchains is one of the most prominent but also controversial technologies of our time. Never before did we have an incorruptable ledger that can be shared with mutually untrustful parties. The ledger in fact can contain any type of data. Thus enabling applications in myriad of fields, such as finance [**todo**], heath care [**todo**], logistics [**todo**], energy [**todo**] and so on. However, not everyone is convinced on the practicality of blockchain systems. Bram Cohen, the inventor of BitTorrent said "It's getting a lot more media attention than the actual impact it's having so far." [4] Forbes listed a few reasons on why we should be skeptical about blockchains [1], some of them include its accessibility for the layperson and its scalability issues.

Scalability is indeed one of the key challenges we face in blockchain systems today. Bitcoin [**bitcoin**], the largest permissionless[1] blockchain system in terms of market capitalisation [**bitcoinmarketcap**] has a maximum transaction rate of merely 7 transaction per second (TX/s). This is due to the consensus mechanism in Bitcoin, namely proof-of-work (PoW), miners can only create new blocks every 10 minutes and every block cannot be larger than 1 megabyte. Payment processors in use today such as Visa can handle transaction rates in the order of thousands [7]. While Bitcoin may be a revolutionary phenomenon, it clearly cannot be ubiquitous in its current state.

An different approach is to not reach global consensus at all. For instance in TrustChain [**multichain**] and Tangle [**tangle**], nodes in the network only store their personal ledger. Since consensus is left out, nodes can perform transactions as fast as their machine and network allows. The downside of this approach is that it cannot prevent fraud (it is possible to detect fraud). To examplify, a malicious node Mallory may claim she has 3 units of currency to Alice, but in reality Mallory already spent all of it on Bob. If there is no global consensus and Bob and Alice never communicate, then

[1] Explain permissionless

the 3 units that Alice is about to receive is nonexistent.

The scalability property of TrustChain and Tangle are exceptionally desirable. The global consensus mechanism of Bitcoin and many other blockchain systems are also worthwhile for detecting or preventing fraud. These two properties may seem mutually exclusive, but in this work, we demonstrate the opposite. Specifically, we answer the following research question in the affirmative. *Is it possible to design a blockchain fabric that can reach global consensus on the state of the system and also scalable?* We define scalability as a property where if more nodes join the system, then the transaction rate should increases.

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 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.

Using the aforementioned insight, we explore an alternative consensus model for blockchain systems where transactions themself do not reach consensus, but nevertheless verifiable at a later stage by any node in the network. Informally, out model works as follows. Every node stores its own blockchain and every block is one transaction, same as the TrustChain construction. We randomly selected nodes in every round, the selected ones are called facilitators. They reach consensus not on the individual transactions, but on the state of every chain represented by a single digest, we call this state the checkpoint. If a checkpoint of some node is in consensus, then that node can proof to any other node that it holds a set of transactions that computes (form a chain) to the checkpoint. This immediately show that those transactions are tamper-proof.

We begin the detailed discussion by formulating the model **??**. Next, we analyse the correctness, security and performance of our design in **??**, this is where we present our theorems. Implementation and experimental results are discussed in **??**. Finally, we compare our system with other state-of-the-art blockchain systems and conclude in Sections **??** and **??** respectively.

Chapter 2

System Architecture

The primary goal guiding our design is scalability. As mentioned in the Introduction, 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 framework that provides the building blocks of blockchain based applications. Application developers using the framework should be able to create any application they wish. Further, we do not impose on a consensus algorithm, as long as it satisfies the properties of atomic broadcast which we describe in Section 2.1.2.

Due to the nature of our system, we do not explicitly address the Sybil attack [2]. 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 [8]. 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 a application neutral framework. On the other hand, our system also has no restrictions on the Sybil defence technique and application designers can pick the best mechanism for their application.

The third and final goal is security. Our system should be unaffected in the presence of powerful adversaries. Security is often difficult to verify, especially when it is not formalised, therefore we require our design to be provably secure. To summarise, our system design is designed with the following goals in mind.

- Application neutrality,
- scalability and
- security.

We begin the chapter with an intuitive overview of the architecture in Section 2.1. Next, we give the formal description, starting with the model and assumptions in Section 2.2. Then, the three protocols which make up the complete system, namely consensus protocol (Section 2.3), transaction protocol (Section 2.4) and validation protocol (Section 2.5). Finally, the possible extensions are described in Section 2.6.

2.1 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 2.1.4.

2.1.1 Extended TrustChain

Extended TrustChain naturally builds on top of 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. 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. When a node wishes to add a new transaction (TX), a new TX block is generated and is appended to the chain. A TX block must have a valid hash pointer pointing to the previous block and a reference[1] to its *pair*. As a result, a single transaction generates two TX blocks, one on each party's chain. An example of is shown in Figure 2.1.

If every node follows the rules of TrustChain and we only consider hash pointers, then the 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 2.1, 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.

Extended TrustChain

We are now ready to explain the Extended TrustChain, which we abbreviate to ETC. In ETC, we introduce a new type of block—checkpoint (CP) block. In contract to TX blocks, CP blocks do not store transactions or contain

[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.

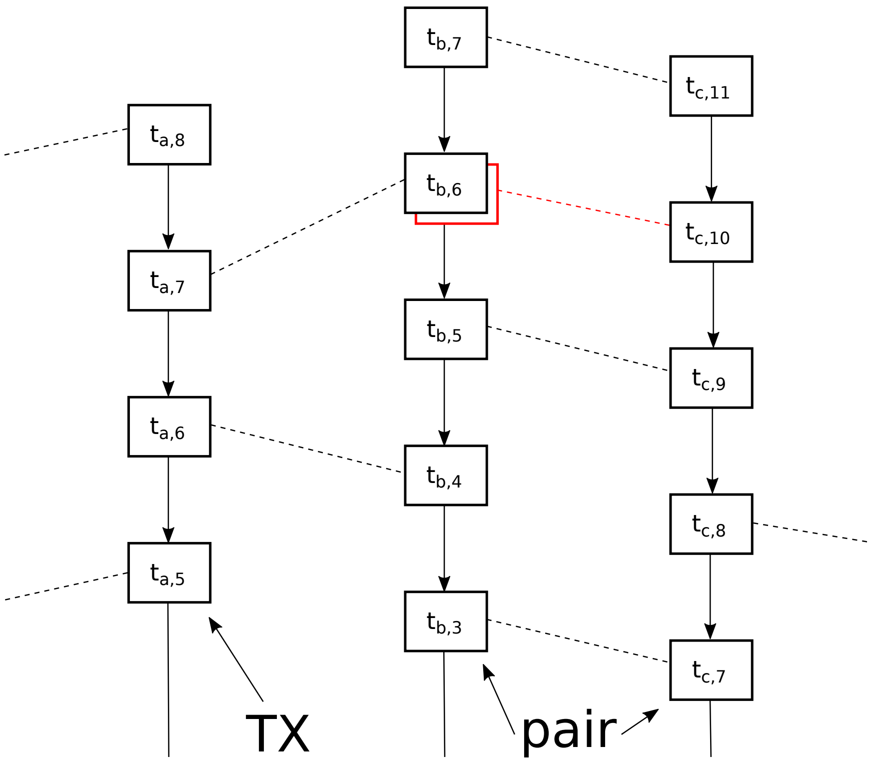

Figure 2.1: Every block is denoted by $t_{i,j}$, where i is the node ID and j is the sequence number of the block. Thus we have three nodes and three corresponding chains in this example. The arrows represent hash pointers and the dotted lines represent references. The blocks at the ends of one dotted line are pairs of each other. The red block after $t_{b,5}$ indicate a fork.

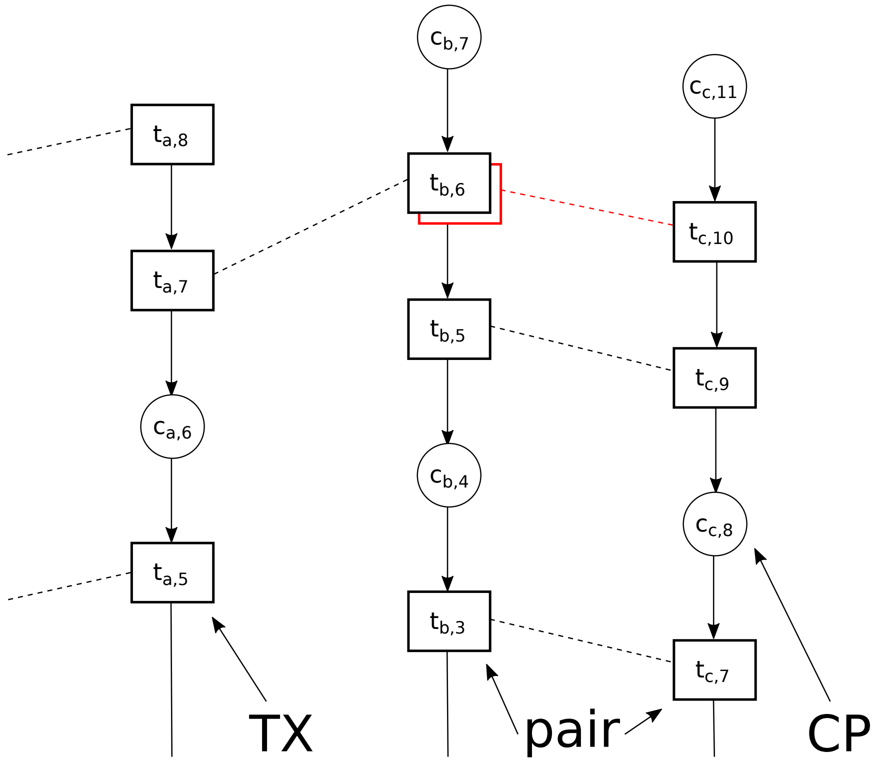

Figure 2.2: The circles represent CP blocks, they also have hash pointers (arrow) but do not have references (dotted line). Note that the sequence number counter do not change, it is shared with TX blocks.

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 2.1.2. A visual representation is shown in Figure 2.2.

2.1.2 Consensus Protocol

Before describing our consensus protocol, we take a brief detour to explain asynchronous common subset primitive and contrast it with other well known primitives. It is a fundamental building block of our consensus protocol.

Asynchronous common subset

TODO give history/background of ACS Asynchronous common subset (ACS) 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. Concretely, ACS needs to satisfy the following properties (adapted from [3]).

Definition 1. *1.* Validity*: If any correct party outputs a set C, then $|C| \geq n - t$ and C contains the input of at least $n - 2t$ parties.*

2. Agreement*: If a correct party outputs C, then every party outputs C.*

3. Totality*: If $n-f$ party receive an input, then all correct parties produce an output.*

ACS has the nice property of censorship resiliance when compared to other consensus algorithms. For instance, Hyperledger and Tender mint uses Practical Byzantine Fault Tolerance (PBFT) 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.

The main drawback with ACS and other BFT protocols is the high message complexity. Typically, such protocols have the 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.

Consensus Protocol

The consensus protocol runs continuously in rounds because a blockchain systems always need to reach consensus on new values, which are CP blocks in our case. This can be seen as running infinitly many rounds of some Byzantine consensus algorithm, starting a new execution immediately after the previous one is completed.

As we mentioned earlier, the high message complexity prohibits us from running a Byzantine consensus algorithm on a large network. Thus, for every round, we randomly select some node—called facilitators—to collect CP blocks and use them as the proposal. The facilitators are elected using a *luck value*, which is computed using $H(\mathcal{C}_r||pk_i)$, where $\mathcal{C}_r$ is the consensus result in round r and pk_i is the public key of i. 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.

2.1.3 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. Upon the counterparty's response, the node checks that the CP blocks are 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 hash collision is hard) to create a different chain that begins and ends with the same two CP blocks but with a different middle section.

2.1.4 Combined Protocol

The final protocol is essentially the concurrent composition of the three aforementioned protocols, all making use the Extended TrustChain data 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, the node may still perform 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.

2.2 Model and Assumptions

For notational clarify, we use the following convention (adapted from [3]) 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.
- Typewrite (e.g. `ack`) denotes message type.

We assume a distributed network where nodes are fully connected, the channels are reliable[2], but messages may be re-ordered and delayed by at

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

most some time Δ, this is sometimes known as a Δ-synchronous network. Nodes are identified by their public key thus we assume there exist a Public Key Infrastructure (PKI). Finally, we use the random oracle model, calls to the random oracle are denoted by $\mathsf{H}(\cdot)$.

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 inequality $n \geq 3t + 1$ must hold.

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 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.

The primary data structure used in our system is 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.

2.2.1 TX 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 which should equal i.
3. $txid$ is a cryptographically secure random number representing the transaction identifier.
4. pk_v is the public key of the counterparty.
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.

2.2.2 CP 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 in round r, the other items are the same as the TX block definition. Note that unlike in our prior work [5], 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 due to every node due to sig_u.

2.2.3 Consensus Result

Our consensus protocol runs in rounds as discussed in Section 2.1. 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.

2.2.4 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

$$a = \operatorname*{arg\,min}_{k, k<i, \texttt{typeof}(b_{u,k})=\gamma} (i-k)$$

$$b = \operatorname*{arg\,min}_{k, k>i, \texttt{typeof}(b_{u,k})=\gamma} (k-i),$$

then $\langle c_{u,a}, c_{u,b} \rangle$ is the *enclosure* of $t_{u,i}$. 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 computed 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 $\mathcal{C}_r$. Similarly, *agreed fragment* is computed using agreed enclosure. We define its function to be $\mathsf{agreed\_fragment}(\cdot)$

The length of the fragment is constrained by L, namely $\forall t |\mathsf{fragment}(t)| \leq L$. The purpose to prevent spam and encourage nodes to create more CP blocks. L should be sufficiently high so that busy nodes are not hindered by it.

TODO agreed fragment must be consecutive

2.3 Consensus Protocol

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

Definition 2. Necessary properties of the consensus protocol
For any $r \in \{x \in \mathbb{Z}_+ : x < \infty\}$, starting at the genesis round (round 0), the system will eventually reach r^{th} round. Further, for every non-genesis round r ($r > 0$), the following properties must be satisfied.

1. Fair lottery*: Every node with a CP block in $\mathcal{C}_{r-1}$, should have an equal probability to be elected as a facilitator in round r.*

2. Consistent facilitators*: If one honest node decides on a list of facilitators in round r, every other node honest decides on the same list.*

3. Validity*: If any correct facilitator outputs a set C, then $|C| \geq n - t$ and C contains the input of at least $n - 2t$ facilitators. Note that the input is a set of CP blocks, so the output is a set of a set of CP blocks.*

4. Agreement*: If a correct facilitator outputs C, then every facilitator outputs C.*

5. Totality*: If $n - f$ facilitators receive an input, then all correct facilitators produce an output.*

Note that the final three properties are on the facilitators rather than on every node, they are the properties ACS. The fair lottery and consistent facilitator are properties on every node, they are prerequites for ACS.

We proceed to describe the protocol step by step. Starting with the bootstrap phase and then moving on to the actual consensus phase.

2.3.1 Bootstrap 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.

Our bootstrap phase runs as follows. First we assume all N nodes start simultaneously. The facilitators for round 0 are hard coded in the program. In practice this could be the machines controlled by the developer. If some node u is not a facilitator, it sends the message $\langle \texttt{cp\_msg}, c_{u,0} \rangle$ to all the facilitators. If u is a facilitator, it does the same, but also waits for some time $\Delta + 1$ to collect messages of type `cp_msg`. After $\Delta + 1$ elapses, it begins the atomic broadcast protocol using the collected CP blocks as the proposal. $\mathcal{C}_0$ is agreed at the end of the protocol. This concludes the bootstrap phase.

The bootstrap phase can be seen as an idealised version of the setup phase combined with the consensus phase, which we describe next.

2.3.2 Setup phase

The setup phase begins immediately after some consensus result is agreed, but not yet disseminate. This could be right after the bootstrap phase. The goal is to reach agreement on a list of new facilitators between every node and should correspond to the fair lottery and consistent facilitators in Definition 2.

The setup phase works as follows. Assume that facilitators of round $r-1$ have just agreed on $\mathcal{C}_{r-1}$. They then 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 $\mathcal{C}_r$.

Continuing, upon receiving $\mathcal{C}_{r-1}$ and at least $n-f$ valid signatures by u, u performs two asks. First, it computes the new facilitators using $\mathsf{get\_facilitator}(C, n)$ (Algorithm 1) and updates its facilitator list to the result. Second, it creates a new CP block using $\mathsf{new\_cp}(\mathcal{C}_{r-1}, r-1)$ (Algorithm 2). This concludes the setup 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.

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

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.

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

2.3.3 Consensus Phase

The consensus phase begins after the new facilitators are selected. Assuming the facilitators are of round r, the goal is to agree on some $\mathcal{C}_r$ between the facilitators. 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 for some duration D and collect messages of type cp_msg, where $D \gg \Delta$. After D is elapsed, it begins the ACS protocol and some $\mathcal{C}_r$ should be agreed upon by the end of it. The facilitators must satisfy the validity, agreement and totality properties from Definition 2. This bring us back to the setup phase and the cycle can be started again.

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 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.

We remark that this procedure is the same as what is described in the bootstrap phase (Section 2.3.1), but the precondition and the waiting time is different. In particular, the bootstrap stage assumed that every node initiated and sent the cp_msg simultaneously, but here we make no such assumption. Furthermore, in the bootstrap phase nodes wait for duration $\Delta + 1$ to collect all the CP blocks, here we wait for some duration D—a system parameter, which is much larger than $\Delta + 1$ to ensure that all honest cp_msg are collected.

2.4 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 [6, 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.

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

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 $txid$

without waiting for the corresponding `tx_resp` messages.

2.5 Validation Protocol

Up to this point, we do not provide a mechanism to detect forks or other forms of tampering or forging, and we cannot detect malicious parties. 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 to be valid.

2.5.1 Validity Definition

A transaction can be in one of three states in terms of validity—*valid*, *invalid* and *unknown*. Given a fragment F, the validity of the transaction t is captured by the function $\mathsf{get\_validity}(t, F)$ (Algorithm 5). The first four conditions (up to Line 22) 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.

2.5.2 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 3.

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.*
4. Unforgeability*: If some transaction is valid, it cannot be forged into an invalid transaction. If some transaction is invalid, it cannot be forged into a valid transaction.*

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.

2.6 Protocol Extensions

(Move to future work?)

Up to this point, we have discussed our protocol in the context of the model and assumptions defined in Section 2.2. In this section, we remove a few assumptions and discuss how our architecture is adapted.

Gossip?

Churn?

Algorithm 5 Function get_validity($t_{u,i}, F_{v,j}$) runs in the private context of u. $t_{u,i}$ is the transaction that u wishes to verify, and $F_{v,j}$ is the corresponding fragment received from v.

```
F_{u,i} ← agreed_fragment(t_{u,i})
if F_{u,i} = ⊥ then
    return unknown
end if                                                        ▷ u has agreed fragment

c_{v,a} ← first(F_{v,j})
c_{v,b} ← 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 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} ← last(F_{u,i})
if c_{u,b} is not created using the same C_r as c_{v,b} then
    return unknown
end if                                                        ▷ correct consensus round

txid, pk_v, m ← t_{u,i}
if number of blocks of txid in F_{v,j} ≠ 1 then
    return invalid
end if                                                        ▷ TX exists

txid', pk'_u, m' ← t_{v,j}
if m ≠ m' ∨ pk_u ≠ pk_u ∨ |F_{v,j}| > L 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 $\mathsf{get\_validity}(t_{u,i}, F_{v,j})$

Chapter 3

Analysis

Up to this point we described our system specification in detail. Of course, specification along does not establish any truths. In this chapter, we prove two aspects of our system. First is correctness, where we show that the consensus protocol and the validation protocol satisfies their desired properties (Definition 2 and Definition 3 respectively). The second is performance, where we prove the lower bound of our throughput and show that it out performs classical blockchain systems.

3.1 Correctness

Our first objective in this section is to establish truths regarding the correctness of our protocol. We do this in two parts. First we use mathematical induction to show that properties in Definition 2 holds for all round. Building on top that, if Definition 2 is true, then we can show that many properties in Definition 3 is true.

Correctness of consensus

$\forall r \in \mathbb{N}$, the following properties must be satisfied.

1. *Agreement*: If one honest node outputs a list of facilitators F_r, every other node honest decides on F_r
2. *Validity*: If any honest node outputs F_r, then $|F_r| > n$ and $|F_r|$ must contain at least $n - t$ honest nodes.
3. *Fairness*: Every node should have an equal probability of becoming a facilitator.
4. *Termination*: At the end of the round, every node outputs some F_r.

3.1.1 Correctness of Consensus

We begin our analysis by establishing the fact that the $\mathsf{get\_facilitator}(\cdot)$ is fair.

Lemma 1. Fairness of facilitator election

Every node with a CP block in $\mathcal{C}_r$, should have an equal probability to be elected as a facilitator.

Proof. This directly follows from the random oracle model. Recall that the luck value is computed using $\mathsf{H}(\mathcal{C}_r, || pk_i)$. Since pk_i is unique for every node that has a CP block in $\mathcal{C}_r$, the output of $\mathsf{H}(\cdot)$ is uniformly random. This implies that the ordered sequence by luck value is uniformly random. $\square$

Using Lemma 1, we show that our consensus protocol satisfied Definition 2.

Lemma 2. Correctness of consensus $\forall r \in \mathbb{N}$, *agreement, validity, fairness and termination holds.*

Proof. We proof by 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 F_1, which indicates the facilitators that are agreed in round 1 and are responsible for driving the ACS protocol in round 2.

For the inductive step, we assume that the properties hold for round r. Then, to start round $r+1$, the honest nodes begin sending CP blocks to F_r. Since the honest region in F_r waits for D and $D \gg \Delta$, the CP blocks of the honest nodes are guaranteed to be received by at least $n-t$ facilitators. The agreement property of ACS (from Definition 1) ensures that the consensus result $\mathcal{C}_{r+1}$ is in consensus. The validity property of ACS ensures that $\mathcal{C}_{r+1}$ contains the input of at least $n-2t$ parties, but this is a quorum containing at least one honest facilitator, thus $\mathcal{C}_{r+1}$ contains the CP blocks of all facilitators. Observe that F_{r+1} is computed using the deterministic function $\mathsf{get\_facilitators}(\cdot)$. Thus agreement of F_{r+1} follows directly from agreement of ACS. Due to the assumption that the adversary cannot corrupt more than t nodes, validity of F_{r+1} also follows from validity of ACS. The fairness property follows from Lemma 1. Finally, the termination property holds because D eventually elapses and then ACS eventually terminates. This completes our proof.

$\square$

3.1.2 Correctness of Validation

Remark: chain structure cannot be satisfied, but can be probabilistically checked

3.2 Performance

Chapter 4

Implementation and Experimental Results

4.1 Implementation

Python. Twisted.

4.2 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.

Bibliography

[1] Jason Bloomberg. *Eight Reasons To Be Skeptical About Blockchain.* 2017. URL: `https://www.forbes.com/sites/jasonbloomberg/2017/05/31/eight-reasons-to-be-skeptical-about-blockchain/` (visited on 12/06/2017).

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

[3] 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.

[4] Justin OConnell. *BitTorrent Creator Cohen Talks Bitcoin, Blockchain.* 2017. URL: `https://www.cryptocoinsnews.com/bram-cohen-talks-bitcoin/` (visited on 09/06/2017).

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

[6] 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`.

[7] *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).

[8] 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

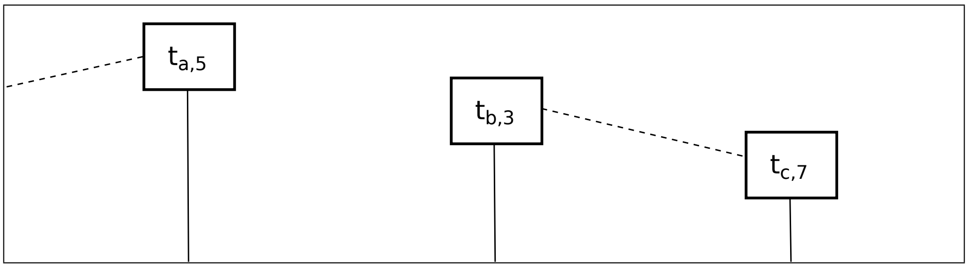

Figure A.1: Initial state

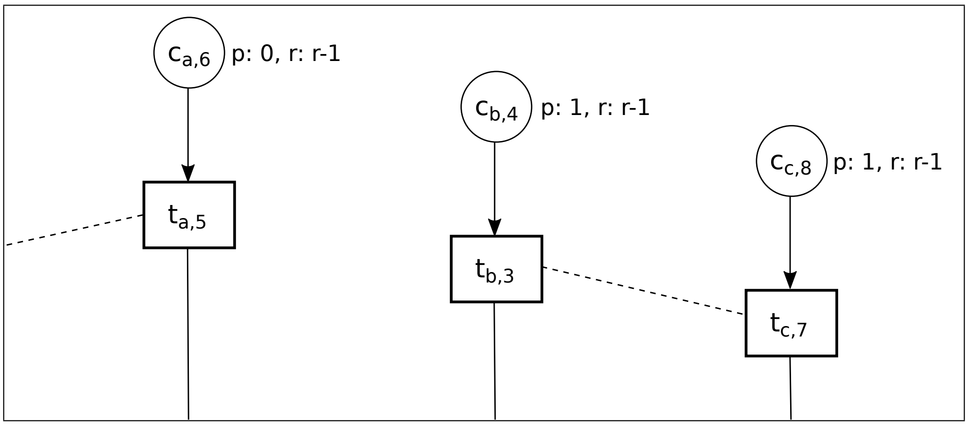

Figure A.2: Initial state

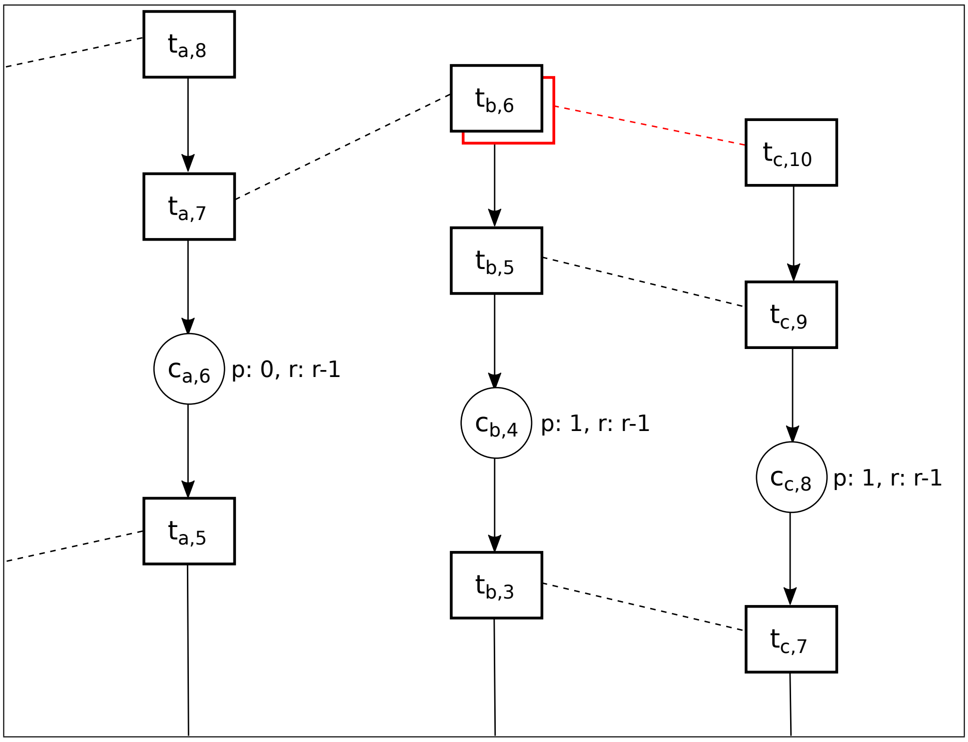

Figure A.3: Initial state

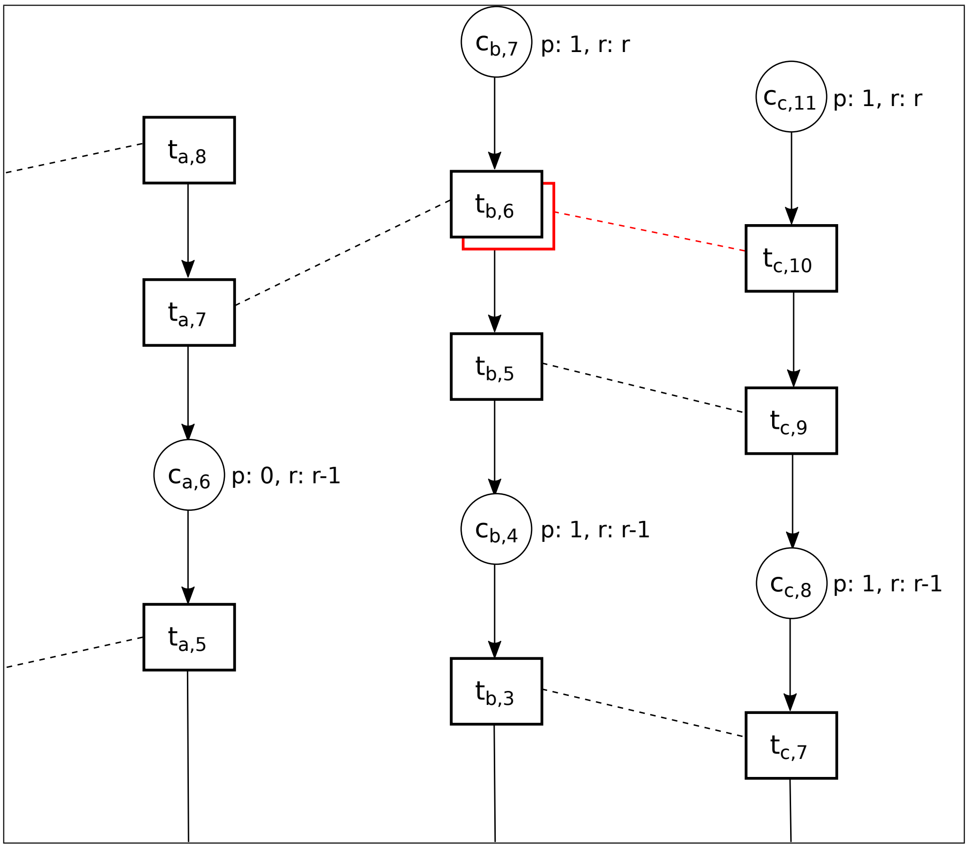

Figure A.4: Initial state