Blockchain Consensus Protocol with Horizontal Scalability

Kelong Cong

Blockchain Consensus Protocol with Horizontal Scalability

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

4th August 2017

Author
Kelong Cong

Title
Blockchain Consensus Protocol with Horizontal Scalability

MSc presentation
31st August 2017

Graduation Committee

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

Abstract

Blockchain systems have the potential to decentralise many traditionally centralised system. However, scalability remains a key challenge. Without a horizontally scalable solution, blockchain system remain unsuitable for ubiquitous use. We design a novel blockchain system which we call CheckpointConsensys. Each node in our system maintain individual hash chains, which only stores transactions that involve the node. Consensus is reached on special blocks called checkpoint blocks rather than on all the transactions. Checkpoint blocks are effectively a hash pointer to the individual hash chains, thus a single checkpoint block may represent a large set of transaction blocks. The consensus protocol does not imply transaction validity. Hence we include a validation protocol which allows nodes to verify that the counterparty correctly recorded the transaction. We implement a CheckpointConsensys prototype and evaluate it experimentally. Our results show strong indication of horizontal scalability even in the worst case, where every transaction is with a randomly selected node. For 1200 nodes, we were able to perform almost 5000 transactions per second, orders of magnitude higher than the 7 transaction per second limit of Bitcoin.

Preface

TODO MOTIVATION FOR RESEARCH TOPIC

TODO ACKNOWLEDGEMENTS

Kelong Cong

Delft, The Netherlands
4th August 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 (root name server), certificate authorities (root CA), 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 resulted 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 [28]. Which, for the first time, gave us a simple banking system in the form of a distributed ledger, also known as blockchain. It needs no central control but still incorruptable with high probability even if there are malicious parties that aim to undermine the system.

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. If C can simultaneously claim that she transferred 5 units to A but also 5 units to B, then the system is inconsistent. Bitcoin and many of its derivatives (e.g. Litecoin[1] and Dogecoin[2]) solve the inconsistency problem with a consensus algorithm. The goal of the algorithm is to reach agreement on a set of valid transactions. This effectively 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 com-

[1] https://litecoin.org
[2] http://dogecoin.org

pete in solving a puzzle. The puzzle is easy to verify but difficult to solve. Only the miner that solves it can generate a valid 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. This property ensures malicious nodes cannot easily rewrite history if they do not have a majority of the network's computing power. Consequently, 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.

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 [32]. Its market cap, at the time of writing, is over 40 billion USD [12]. Many online marketplaces are using Bitcoin, for example Steam[3] and even Amazon[4]. Due to its success, people from many different disciplines began investigating in way to use blockchain technology. This includes finance [43], heathcare [14], logistics [38] and energy [3].

Sadly, as traditional blockchain systems began to gain popularity, their limitations also became apparent. Bitcoin has the infamous 7 transactions per second upper bound [47]. 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 20,000 transactions[5]. A few months ago the backlog even reached 100,000, which meant new transactions would take at least 11 hours to be written to the blockchain [39]. This 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 [41]. 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. It also requires a hard fork (not backward compatible), which risks consensus failure and devaluation. A recent empirical study by Croman et el. [13] has shown that increasing the block size may help. But given the bandwidth and latency constraints, it is not possible to have more than 758 transactions per second. Even worse, 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, which is in the order of tens of thousands transactions per second [46].

What we have described is in fact the current state of Bitcoin. Many proposals exist with the aim to make Bitcoin more scalable. The most

[3] https://store.steampowered.com/

[4] Not directly, but via https://purse.io/.

[5] https://blockchain.info/unconfirmed-transactions/

prominant is Seggregated Witness or SegWit [24]. It moves the signature data to the end of the block and changes the block size to 4 MB. The effect is that the first 1 MB of the block is still backward compatible, thus there is no need for a hard fork. It also solves the transaction malleability problem [8] and paves the way for offchain transactions via Lightning Network [34]. Although SegWit is implemented, it is still uncertain whether the Bitcoin network will adopt it. At the time of writing, there is only 35% of the network signalling for adoptation, this is only a 10% increase since November last year[6]. Thus it may take years if ever for SegWit to be adopted. Even if it is adopted, it cannot reach the transaction rate of centralised systems.

In this work, we take the advice of Croman et el. [13] and Marko [47] 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 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 transactions 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, our system works as follows. Every node stores its own blockchain and every block is one transaction. We randomly select 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 call our protocol CheckpointConsensys.

Our main contributions are the following.

- We formally introduce a blockchain system—CheckpointConsensys. It uses individual hash chains and checkpoints on every node to achieve horizontally scalable for the first time.
- We formally analyse CheckpointConsensys to ensure correctness as

[6] https://blockchain.info/charts/bip-9-segwit

defined in our architecture.

- We implement Checkpiont Consensus with experiment with up to 1200 nodes, our results show strong evidence of horizontal scalability.

We describe and analyse our idea in the remainder of this work. We begin with the problem description in Chapter 2 which describes the state-of-the-art work on making blockchain systems scalable and our research question. In Chapter 3 we give a formal description of our system model and architecture. 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 conclude in 6 and discuss future work.

Chapter 2

Problem description

From the Introduction, we saw that early blockchain systems face a scalability problem. In particular, the use proof-of-work is unlikely to reach a transaction rate suitable for ubiquitous use. Many attempts to fix the scalability problem exist in academia as well as the industry. In this chapter we describe the related work in the form of a mini taxonomy and identify areas for improvement and limitations. Then we state our research question which aims to overcome some of the limitations.

2.1 Related work and a mini taxonomy

Blockchain research has seen a surge in recent years from both the industry and academia, this resulted in a myriad of of blockchain systems with different approaches to scalability. To understand the current state-of-the-art, we classify various blockchain systems by their scalability approach.

2.1.1 Early blockchain systems

This category represents the baseline, which are 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 containing many transactions is determined by 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 [28]. Nevertheless, the probability of the highest block being orphaned is non-negligible.

The advantage of this approach is that it can be used in a large network. In fact, prior to Bitcoin there were no internet-scale consensus protocol. Bitcoin is also fault tolerant, because attackers can not out pace honest users in finding new blocks unless they have a majority of the hash power.

As long as the majority is honest and reasonable measures are taken to prevent the selfish mining attack [18]. The disadvantage however is that transactions are never in consensus with a probability of 1—no consensus finality. Hence double spending is possible for transactions that do not have a large number of confirmations. Furthermore, as we mentioned in the Introduction, the performance is limited due to the fact that blocks are of fixed size and are generated at fixed intervals. Thus early blockchain systems do not achieve horizontal scalability.

2.1.2 Off-chain transactions and payment networks

Off-chain transactions make use of the following fact. If nodes make frequent 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 [34] and Duplex Micropayment Channels [15].

Off-chain transaction systems are implemented using multi-signature addresses [7] and hashed time-locked Bitcoin contracts [6]. In the simiplest case, if two parties wish to make transactions, they open a payment channel in the form of of a multi-signature Bitcoin address (for two parties it would be a 2-of-2 signature address). Each of the party must also deposit some Bitcoins in the multi-signature address, this is called the opening transaction. Both parties keep the channel state which is not broadcasted to the network. The state is updated when the two parties make new transactions. The protocol disincentivises the parties from broadcasting old channel states. If this issue occurs, the counterparty can drain all the Bitcoins in the channel. To close the channel, the parties simply boradcast the latest net settlement to the Bitcoin network, this is called the closing transaction. Effectively, only two transactions need to be broadcasted to the Bitcoin network—the opening and the closing transaction, even if the two parties made thousands or millions of transactions in between. The two-party scheme can be extended to a network of channels, which allows two parties to make off-chain transactions without an open channel as long as they are connected by nodes that do.

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. Some implementation already exist[1], but at the time of writing Segwit is not activated yet which is a prerequisite for Lightning Network.

On the other hand, off-chain transactions also suffers from the problem of Bitcoin. Proof-of-work still consumes an unreasonable amount of power. Further, payment channels complicates user experience. As we mentioned

[1]For example https://github.com/lightningnetwork/lnd.

earlier, each node must deposit some Bitcoins into a multi-signature account. The users must pick a suitable amount; if the deposit is too low it would not allow large transactions, if the deposit is too high the user is locked out of much of its Bitcoins for use outside of the channel. In addition, the user must proactively check whether the counterparty has broadcasted an old channel state so that the user does not lose Bitcoins. Payment channel in theory solves the scalability problem of early blockchain systems, but to the best of our knowledge its exact scalability characteristics are not investigated.

2.1.3 Permissioned systems based on Byzantine consensus

This category of systems use traditional Byzantine consensus algorithms such as PBFT [10]. In essence, they contain a fixed set of nodes (sometimes called validating peer) that run a Byzantine consensus algorithm to decide on new blocks. This is often used in permissioned system where the validators must be pre-determined, for example Hyperledger Fabric [9].

A nice aspect of Byzantine consensus and in particular PBFT is that it can handle much more transactions than classical blockchain systems. PBFT can for example achieve 10,000 TX/s if the number of validating peer is under 16 [27, Section 5.2]. Further, in contrast to proof-of-work, PBFT consensus is final. That is, transaction history cannot be re-written if it is already in the blockchain.

The major drawback of Byzantine consensus based systems is that it does not scale in terms of the number of validating peers. Going back to PBFT, its transaction rate drops to under 5000 TX/s when the number of validating peer is 64 [27, Section 5.2]. Moreover, the validating peers are pre-determined which makes the system unsuitable for the open internet.

2.1.4 Combining proof-of-work with Byzantine consensus

Recent research has developed a class of hybrid systems which uses PoW for committe election, and Byzantine consensus algorithms to agree on transactions. This design is primarily for permissionless system because the PoW leader election aspect prevents Sybils. Some examples are SCP [26], ByzCoin [20] and Solidus [1].

This technique overcomes the early blockchain scalability issue by delegating the transaction validation to the Byzantine consensus protocol (e.g. PBFT in ByzCoin [20]). A major tradeoff of such systems is that they cannot guarantee correctness when there is a large number of malicious nodes (but less than majority). For SCP, ByzCoin and Solidus, they all have some probability to elect more than t Byzantine nodes into the committee, where t is typically just under a third of the committee size (a lower bound of Byzantine consensus [33]). This problem is especially difficult to solve because the committee is always much smaller than the population size which

has more than t Byzantine nodes. Classical blockchain do not have this problem because they do not use Byzantine consensus. Further, due to the fact that these system must reach consensus on all transactions, none of them achieves horizontal scalability.

2.1.5 Sharding

Sharding is a technique to achieve horizontal scalability by grouping nodes into multiple committees or shards. Nodes within a single shard run a Byzantine consensus algorithm to agree on a set of transactions that belong to that specific shard. An intra-shard protocol is needed for transactions that involve nodes from more than one shard. The number of shard grow linears with respect to the total computational power in the network. This scheme achieves horizontal scalability because if every shard commits transactions at the same throughput, then adding more shard would naturally result in a linear increase of global throughput. Examples of blockchain systems that use sharding are Elastico [25] and OmniLedger [21].

The biggest limitation of sharding is that it is only optimal is transactions stay in the same shard. In fact, Elastico cannot atomically process inter-shard transactions. OmniLedger has an inter-shard transaction protocol but choosing a good shard size is difficult. A large shard size would make the system less scalable, because the Byzantine consensus algorithm must be run by a large number of nodes. A small shard size would result in a large number of inter-shard transactions which also hinder scalability. The authors of OmniLedger noted that inter-shard transactions have significantly higher latency compared to the consensus protocol [21]. Furthermore, since no shards can be compromised for the system to function correctly, the adversaries have more opportunites to compromise the system.

2.1.6 Blockchain without global consensus

Tangle [35], Corda [19] and the original TrustChain [31] do not use global consensus at all. In this scheme, every node has their own chain. Transactions between nodes are recorded on their respective chains. This effectively results in a directed acyclic graph. By avoiding global consensus, this approach achieves extremely desirable scalability properties similar to BitTorrent [11].

On the other hand, the application and security is limited. A malicious node can easily lie to other nodes regarding its own chain by simply presenting one version of reality to a set of nodes and another version to a different set. Thus the network will have a conflicting view on the state of the malicious node. Applications such as digital cash can not be implemented on top of such systems because consistency is needed in solving the double spending problem.

2.2 Research question

We saw in Section 2.1 that the majority of the systems cannot achieve horizontal scalability. That is, by adding new nodes into the network, the global transaction rate should increase proportionally. To the best of our knowledge, only two systems—Elastico [25] and OmniLedger [21]—has experimentally demonstrated horizontal scalability. However, these techniques rely on sharding, thus the performance highly depends on transaction characteristics and the shard size parameter because inter-shard transactions are slow. Furthermore, all the systems described in Section 2.1 are complete blockchain systems designed for a concrete application. We wish to design a blockchain consensus protocol that can be used in many applications. For example recording uploads and downloads between peers in Tribler [36, 31] and decentralised market[2]. Hence our research question is the following.

> *Is it possible to design a blockchain consensus protocol that is fault tolerant, scalable and can reach global consensus?*

In order to make sense of our research question, we must first define *blockchain consensus protocol*. Typical blockchain systems are application specific. For example, the application in Bitcoin (any many of its derivatives) is digital cash, the application in Namecoin is domain name registration [29] and the application in Siacoin is cloud storage [30]. Underneath these applications lies the blockchain consensus protocol which has the goal of reaching consensus on transactions or state changes. In the case of Bitcoin it is proof-of-work (PoW). It is not concerned with the structure of the transactions, it works by simply treating transactions as a blob of data and hashes them with the previous block. In this work, we focus on the blockchain consensus protocol rather than any specific application. Because it is the scalability bottleneck and is neccessary for any blockchain based application.

In the remainder of this chapter we expand on our research question and visit each of our requirement in detail. At the end of this chapter we describe some issues that we do not consider as a part of our design, which is the result of not focusing on any specific application.

2.2.1 Global consensus

The first objective is to reach consensus on a global *state* by building on top of decades of Byzantine consensus protocol research [33, 22, 10, 27]. Having consensus is essential in blockchain systems. It stops many types of malicious activities because agreed state must have the consent of the honest nodes in the network. We emphasise the word state because in contrary to traditional blockchains, it does not neccessarily mean the set of all transactions that

[2] https://github.com/Tribler/tribler/blob/devel/Tribler/community/tradechain/__init__.py

has ever happened. The state can be a verifiable representation of all the transactions. As an extreme example, a single digest of the complete history can represent the state. It is even verifiable as long as there are nodes that stores the history.

2.2.2 Security and fault tolerance

The second objective is fault tolerance, where 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. This asppect usually comes for free when using a Byzantine fault tolerant consensus algorithm.

There is another aspect of fault tolerance that is transaction verifiability. All transactions must be eventually verifiable to maintain system integrity. Further, the validation result must be consistent. For example, if two different honest nodes attempts verify the same transaction, it cannot be the case that one nodes thinks that the transaction is valid but the other thinks it is invalid.

2.2.3 Performance and scalability

The last but not least, the final objective is horizontal scalability. We wish to see the performance (global transaction rate) increase as more nodes join the network. BitTorrent [11] is an example of such a system, where peers interact with each other to exchange files without any global bottleneck.

Early blockchain systems may run in a network of thousands of nodes, but the performance do not scale. On the other hand, as we will see in Section 3.4.1, Byzantine consensus algorithms performs well but only when in a small network. In this work, our goal is to overcome the limitations of early blockchain systems and Byzantine consensus algorithms to create a horizontally scalable blockchain.

2.2.4 Limitations

Our work aims to significantly lower the transaction bound. However, the price to pay is that the Sybil attack [16] prevention mechanism moves from the blockchain consensus protocol to the application layer. For example, social network based Sybil defence mechanims use graph structure of real-world relationships [49]. Online marketplaces such as Amazon use the rating of buyer and sellers. Since our system is application neutral, we do not provide an explicit Sybil defence mechanism. 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.

For the same reason, we do not explicitly consider spam or denial of service attacks. Without a concrete application it is difficult to distinguish a super active user from an attacker.

Chapter 3

System architecture

Our system—CheckpointConsensys consist of three protocols—the consensus protocol, the transaction protocol and the validation protocol. It is based on our prior work on TrustChain [31], where every node independently interact with each other via their own blockchain[1].

The goal of this chapter is to detail our system specification. We begin our discussion 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, we discuss a few variations of our main design and their respective tradeoffs in Section 3.7.

3.1 System overview

The architecture is visualised in Figure 3.1. It consist of one data structure—Extended TrustChain, and three protocols. The protocols have distinct roles which are evident from their names. They run concurrently and do not interact with each other. The only synchronisation happens on the Extended TrustChain layer, where all the protocols read and write from it. We first describe each component individually, starting with Extended TrustChain, and then explain how they fit together.

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 [31]. 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.

[1]It is originally called MultiChain but the name has changed to TrustChain

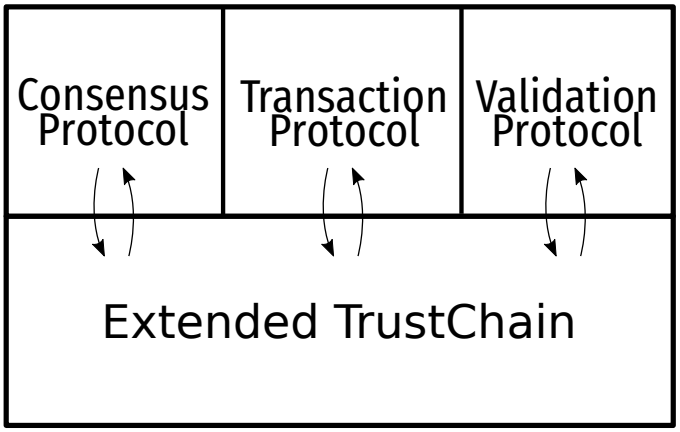


Figure 3.1: Overview of the system architecture

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 for both A and B and is appended to their respective chains. The TX block must have a valid hash pointer pointing to the previous block and a reference[2] to its *pair* on B's chain. An example of is shown in Figure 3.2.

If every node follows the rules of TrustChain and we only consider hash pointers and not references, 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 3.2, 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. 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 3.3.

[2]This is different from the original TrustChain definition found in [31]. 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.

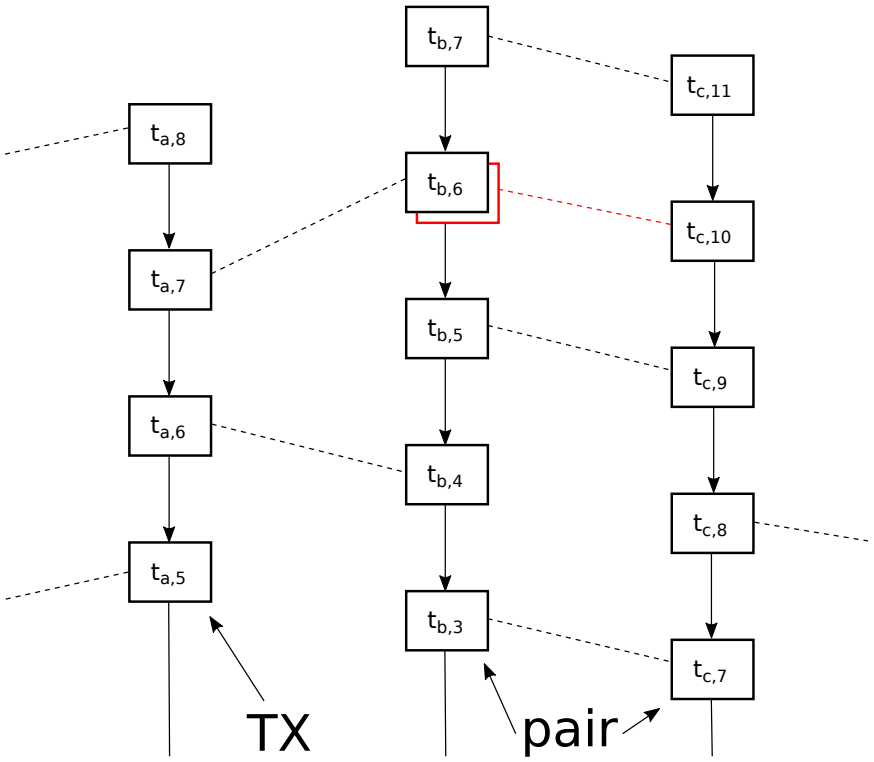


Figure 3.2: Every block is denoted by $t_{i,j}$, where i is the node ID and j is the sequence number of the block. Thus we have three nodes and three corresponding chains in this example. The arrows represent hash pointers and the dotted lines represent references. The blocks at the ends of one dotted line are pairs of each other. The red block after $t_{b,5}$ indicate a fork.

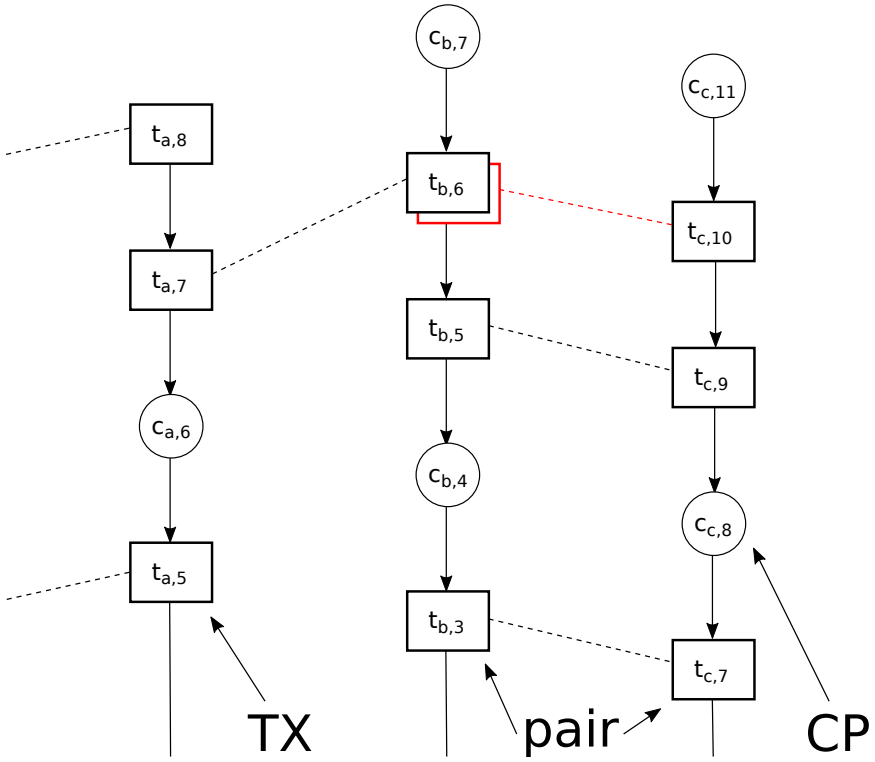


Figure 3.3: The circles represent CP blocks, they also have hash pointers (arrow) but do not have references (dotted line). Note that the sequence number counter do not change, it is shared with TX blocks.

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[3], 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 $\mathsf{H}(\mathcal{C}_r||pk_u)$, where $\mathcal{C}_r$ is the consensus result (which can be seen as the set union of all the CP blocks

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

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. Finally, the elected facilitators run a Byzantine consensus algorithm to agree on a set of CP blocks. The next round begins when the new facilitators are computed from these CP blocks.

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 *agreed fragment* of the TX. An agreed 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.

3.1.4 Combined protocol

The final result 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, 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 lock the chain while the consensus protocol is running and then unlock it afterwards.

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 [27]) for most of 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 stage of our protocol do we make timing assumptions. The adversary has fully control of the delivery schedule and the message ordering of all messages. But they are not allowed to drop messages except for their own[4].

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 (RO) 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. Under the RO model, the probability of successfully computing the inverse of the hash function is negligible with respect to λ [4].

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. This is from the work of Pease, Shostak and Lamport, where they show a network of n nodes cannot reach Byzantine agreement with $t \geq n/3$ [33]. Further, the inequality $N \geq n + t$ must also hold. This is due to our system design, which becomes clear in Section 3.4.3.

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 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. This is realistic because otherwise the adversary can eventually learn all the private keys and sabotage the system.

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

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 $B_u = \{b_{u,i} : i \in \{0, \ldots, h-1\}\}$, where $b_{u,i}$ is the ith block of u, and h is the height of the block (i.e. $h = |B_u|$). 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 all TX blocks in B_u and C_u is the set of all CP blocks is B_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.

Definition 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 pseudo-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.*

Definition 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 Definition 3) in round* r, *the other items are the same as the TX block definition.*

The genesis block in the chain must be a CP block in the form of $c_{u,0} = \langle H(\perp), 0, H(\perp), 0, sig_u \rangle$ *where* $H(\perp)$ *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.*

Definition 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*.*

Here we define an important properties which results from the interleaving nature of CP and TX blocks. It is used in our validation protocol.

Definition 4. Enclosure and agreed enclosure

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

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

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

then $\langle c_{u,a}, c_{u,b} \rangle$ *is the enclosure of* $t_{u,i}$*. Intuitively, that is the two closest CP blocks that surround* $t_{u,i}$*. Note that* $c_{u,a}$ *is the more recent CP block. Also, some TX blocks may not have any enclosure, then its enclosure is* $\perp$*. Agreed enclosure is the same as enclosure with an extra constraint where the CP blocks must be in some consensus result* $\mathcal{C}_r$*.*

Note that the "closest CP blocks" property must also apply to agreed enclosure. Suppose a chain is in the form $\{c_i, c_{i+1}, t_{i+2}, c_{i+3}\}$[5] 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 $\langle c_i, c_{i+3} \rangle$.

Definition 5. Fragment and agreed fragment

If the enclosure of some TX block $t_{u,i}$ *is* $\langle c_{u,a}, c_{u,b} \rangle$*, then its* fragment $F_{u,i}$ *is defined as* $\{b_{u,i} : a \leq i \leq b\}$*. Similarly,* agreed fragment *has the same definition as fragment but using agreed enclosure. For convenience, the function* $\mathit{agreed\_fragment}(\cdot)$ *represents the agreed fragment of some TX block if it exists, otherwise* $\perp$*.*

3.4 Consensus protocol

Our scalable consensus protocol runs on top of Extended TrustChain. It uses an unmodified asynchronous subset consensus (ACS) algorithm as the

[5]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

key building block. 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 6. Properties of the consensus protocol
$\forall r \in \mathbb{N}$, *the following properties must hold.*

1. Agreement*: If one correct node outputs a set of facilitators* $\mathcal{F}_r$*, then every node outputs* $\mathcal{F}_r$

2. Validity*: If any correct node outputs* $\mathcal{F}_r$*, then*

   (a) $|\mathcal{C}_r| \geq N - t$ *and each CP in* $\mathcal{C}_r$ *belong to a different node*[6].

   (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 there are 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 ACS algorithm as it is the primary building block. Then we move on to describe the two phases of our consensus protocol—bootstrap phase and consensus phase.

3.4.1 Background on asynchronous subset consensus

The best way to explain ACS is to contrast it with the typical Byzantine consensus. We adapt the description from [48, Chapter 17].

Definition 7. 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.*

[6] $\mathcal{C}_r$ is a tuple but we abuse the notation here by writing $|\mathcal{C}_r|$ to mean the number of CP blocks in the second element of $\mathcal{C}_r$.

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 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 [27]).

Definition 8. 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 a variant of Practical Byzantine Fault Tolerance (PBFT) [10] 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 [27]. To understand the remainder of this work, the knowledge of Definition 8 is sufficient.

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. As with all distributed systems, there must be a bootstrap phase which sets up the system.

The reader may notice from the system overview (Section 3.1) that the 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 satisfies all the properties in Definition 6. Namely every node has the same set of valid facilitators $\mathcal{F}_1$ that are randomly choosen. This concludes the bootstrap phase. For any future rounds, the consensus phase is used.

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 results 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 6.

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 $N-t$ messages of type `cp_msg` are received. Invalid messages are removed, namely blocks with invalid signatures and 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 to 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, but not in any of the inputs.

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 [27] as our ACS protocol because it is a 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 broad-

cast 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 $\mathsf{new\_cp}(\mathcal{C}_{r+1}, r+1)$ using Algorithm 1. Second, it computes the new facilitators using $\mathsf{get\_facilitator}(\mathcal{C}_{r+1}, n)$ using Algorithm 2, and updates its facilitator set to the result. This concludes the consensus phase and brings us back to the beginning of the consensus phase.

Algorithm 1 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.

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

Algorithm 2 Function $\mathsf{get\_facilitator}(\mathcal{C}_r, n)$ takes the consensus result $\mathcal{C}_r$ and an integer n, then sorts the CP blocks C by the luck value (the λ-expression), and outputs the smallest n elements.

$\langle r, C \rangle \leftarrow \mathcal{C}_r$
$\mathsf{take}(n, \mathsf{sort\_by}(\lambda x.\mathsf{H}(\mathcal{C}_r || pk \text{ of } x), C))$

Our protocol has some similarities with synchronisers [48, Chapter 10] because it is effectively a technique to introduce synchrony in an asynchronous environment. If we consider the facilitators as a collective authority, then it can be seen as a synchroniser that sends pulse messages (in the form of `cons_msg` and `cons_sig`) to indicate the start of a new clock pulse. Every node then sends a completion messages (in the form of `cp_msg`) to the new collective authority to indicate that they are ready for the next pulse.

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 [45, 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
  $\langle \_, \_, txid, pk_v, m, \_ \rangle \leftarrow t_{v,j}$
  $\mathsf{new\_tx}(pk_u, m, txid)$
  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
  $\langle \_, \_, txid, pk_v, m, \_ \rangle \leftarrow t_{v,j}$
  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 $\mathsf{get\_validity}(t, F_u, F_v)$ (Algorithm 5). The first four conditions (up to Line 3) 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 check for

tampering or missing blocks, if any of these misconducts are detected, then the TX is invalid.

We stress that the *unknown* state means that the verifier does not have enough information to continue with the validation protocol. If enough information is available at a later time, then the verifier will output either *valid* or *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 $\mathsf{get\_validity}(t_{u,i}, F_{u,i}, F_{v,j})$ is not necessarily u[7] Any node w may call $\mathsf{get\_validity}(t_u u, i, F_{u,i}, F_{v,j})$ 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.

Algorithm 5 Function $\mathsf{get\_validity}(t_{u,i}, F_{u,i}, F_{v,j})$ validates the transaction represented by $t_{u,i}$. We assume $F_{u,i}$ is always correct and contains $t_{u,i}$. $F_{v,j}$ is the corresponding fragment received from v.

---

if $F_{v,j}$ is not a fragment created in the same round as $F_{u,i}$ **then**
    return *unknown*

$\langle \_, \_, txid, pk_v, m, \_ \rangle \leftarrow t_{u,i}$
if number of blocks of $txid$ in $F_{v,j} \neq 1$ **then**
    return *invalid*

$t_{v,j} \leftarrow$ the TX block with $txid$ in $F_{v,j}$
$\langle \_, \_, txid', pk_u, m', \_ \rangle \leftarrow t_{v,j}$
if $m \neq m'$ **then**
    return *invalid*

if $t_{u,i}$ is not signed by $pk_u \vee t_{v,j}$ is not signed by pk_v **then**
    return *invalid*
return *valid*

---

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

3.6.2 Validation protocol

Our validation protocol is designed to classify transactions according to the aforementioned validity definition. The protocol is a simple response and request protocol shown in 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 9. Properties of the validation protocol

1. 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*.*
2. Validity*: The validation protocol outputs the correct result according to the aforementioned validity definition.*
3. Liveness*: Any valid (invalid) transactions is marked as valid (invalid) eventually.*

Algorithm 6 Validation protocol which runs in the context of u

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 $txid$
  if $F_{u,i}$ and $F_{v,j}$ are available and $F_{u,i}$ is the agreed fragment of $t_{u,i}$ **then**
    set the validity of $t_{u,i}$ to $\mathsf{get\_validity}(t_{u,i}, F_{u,j}, F_{v,j})$

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 Design variations and tradeoffs

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 explore a few design variations which we can make, some of them require a relaxed version of our original model. They enable better performance, allow us to apply our design in the fully permissionless setting and improves privacy.

3.7.1 Using epidemic protocol to reduce communication cost

One of the final steps in our consensus protocol is to broadcast the consensus result and signatures to every other node (Section 3.4.3). While this guarantees delivery (since we assumed reliable channel), it is wasteful. For example, if every facilitator is honest, a node would receive n consensus results which are identical when only one is necessary.

An optimisation we can do is to use an epidemic protocol [17] (also known as gossiping) instead of our broadcast approach. Typical epidemic protocols works as follows. Every node buffers every message it receives up to some buffer size b. Then it forwards the messages t number of times. Every time the message is sent to f random neighbours, f is often called the fan-out. The upside of using epidemic protocol is that the communication cost is distributed more evenly between nodes. This is especially true with a lazy push approach where the node who just received a message would push the message ID (e.g. digest) to its f random neighbours, and only push the full message if the neighbour explicitly requests the message [23]. With this, nodes typically only need to receive one consensus result message instead of n.

A down side of epidemic protocol is that it usually takes $O(\log N)$ time to infect the whole network, whereas broadcasting uses constant time. Another downside of some epidemic protocols (e.g. eager push) is that it is difficult to guarantee delivery. It is especially true when the parameters are not choosen correctly in a network that is only partially connected (but every node is nevertheless reachable). If the delivery cannot be guaranteed, then we cannot guarantee liveness in our consensus protocol because a future facilitator may miss the memo. Picking parameters are difficult in practice because the network configuration is unknown and the total number of nodes might also be unknown.

3.7.2 Using timing assumption in the permissionless setting

Our model is purely asynchronous, where we make no timing assumptions anywhere in the protocol. In many applications however, it is often fine to make timing assumptions. For example, TCP relies on timeout for its retransmission and the `nLockTime` property in Bitcoin transactions makes the transaction unspendable until some time in the future (either Unix time or block height) [5]. One limitation of our system is that we use the parameter N in our algorithms, which makes it unsuitable for the permissionless environment where users can join and leave at will. In this section we show how making a timing assumption would allow us to operate in the permissionless setting.

At the start of our consensus phase (Section 3.4.3), facilitators must wait for $N - f$ `cp_msg` messages. This is the only place where we used N as a

parameter. To introduce timing, instead of waiting for $N - f$ messages, we wait for some time D, such that D is sufficiently long for honest nodes to send their CP blocks to the facilitators. Consequently, this removes the need for a PKI because the collected CP blocks may be from nodes that nobody has seen in the past. However, choosing the parameter D is difficult and depends on a number of factors such as the network condition, message size, and so on. Overestimating it would make agreed fragments much longer than usual, which increases communication costs for validation. Underestimating it would lead to unfairness where users with a poor internet connection will have a lower chance to be selected as a facilitator in the next round. Nevertheless, there is a significant gain for making the timing assumption and that is the ability to operate in the permissionless setting which we explain next.

Suppose a new node wish to join the network and the facilitators are known (this can be done with a public registry). It simply sends its latest CP block to the facilitators. Then, in the next round the node will have a chance to become a facilitator just like any existing node. To leave the network, nodes simply stop submitting CP blocks. There is a subtlety here which happens when the node is elected as a facilitator in the following round. In this case, the node must fulfill its obligation by completing the consensus protocol, but without proposing its own CP block, before leaving. Otherwise the $n \geq 3t + 1$ condition may be violated.

3.7.3 Privacy preserving validation protocol using compact blocks

Our approach already has privacy preserving features in comparison to early blockchain systems. That is, the transactions for each node are only revealed during the validation protocol. Hence if two nodes never directly or indirectly interact with each other, their transactions are never revealed.

We can take our privacy-preserving property one step further by introducing another level of hash pointer indirection. The result is shown in Figure 3.4.

Concretely, we introduce an additional block type, namely compact block. Such blocks only have three attributes,

1. Seq—the sequence number its corresponding block,
2. Digest—the digest of its corresponding block, and
3. Prev—the digest of the previous compact block.

Each compact block has a corresponding full block (either a CP block or a TX block). The relationship is uniquely identified with Seq or Digest. Recall that our original validation protocol requires the nodes to send the full agreed fragment. With compact blocks, it is only necessary to send the

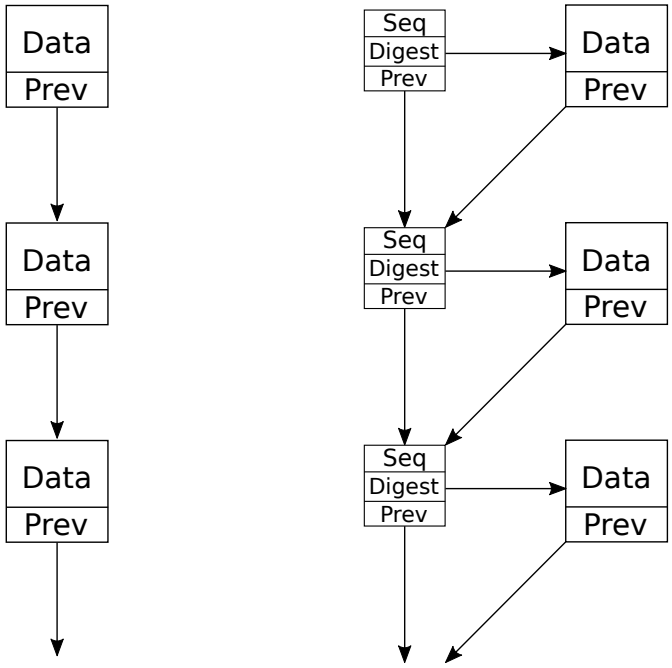


Figure 3.4: The chain on the left represent direct chaining, where the digest in "Prev" is simply the digest of the previous block. The chain on the right uses compact blocks, represented by the smaller squares. Compact blocks also form a chain as before, but they each have a hash pointer to the full block, identified by "Seq" and "Digest".

compact version of the agreed fragment. The validation then proceeds in a similar fashion, provided that the pair of the to-be-validated TX block is known.

The space saving of this approach of course depends on the size of the full blocks. If the full blocks are on average 500 bytes (which is the typical size of Bitcoin transactions [44]), and the compact blocks are $32 + 32 + 8 = 72$ bytes (SHA256 digests are 32 bytes each and we use a 64 bit integer to represent the sequence number), then the saving in communication cost is 86%.

3.7.4 Optimising validation protocol using cached agreed fragments

One more way to improve the efficiency of the validation protocol is to use a single agreed fragment to validate multiple transaction. Concretely, upon receiving an agreed fragment from node A, rather than validating a single transaction, we attempt to validate all transactions in the unknown state performed with A in that fragment.

For a node, the benefit of this technique is maximised when it only transacts with one other node. In this case, the communication of one fragment is sufficient to validate all transactions in that fragment. In the opposite extreme, if every transaction that the node makes is with another unique node, then the caching mechanism would have no effect.

3.7.5 Global fork detection

The validation algorithm guarantees that there are no forks within a single agreed fragment. This is sufficient for some applications such as proving the existance of some information. However, for applications such as digital cash where every block depends on one or more previous blocks, our scheme is not suitable. For such applications we need to guarantee that there are no forks from the genesis block leading up to the TX block of interest.

There are a variety of approaches to do global fork detection. First and the easiest solution is to simply ask for every agreed fragment from the one containing the genesis block up to the one that the to-be-proven TX block is in. If the hash pointers are correct and all the CP blocks are in consensus, then the verifier can be sure that there are no forks. We use this approach in our prior work on Implicit Consensus [40]. It may sound inefficient at first, but nodes employ caching to minimise communication costs. In our work we call this effect "spontaneous sharding".

The second approach is probabilistic but with only a constant communication overhead. For a node, observe that if all of its agreed fragment has a transaction with a honest node, then the complete chain is effectively validated in a distributed maner. The only way for an attacker to make

a fork is to make sure that the agreed fragment containing the fork has no transactions with honest nodes. Such malicious behaviour is prevented probabilisticaly using a challenge-response protocol as follows. Suppose node A wish to make a transaction with node B. A first sends a challenge to B asking it to proof that it holds a valid agreed fragment between some consensus round specified by A. If B provides a correct proof, then they run the transaction protocol as usual. If B provides an invalid proof or refuses to respond, then A would refuse to make the transaction. The probability that an honest node catches out a malicious node is

$$p = \frac{f}{r},$$

where f is the number of bad agreed fragments and r is the latest round number. If there are more nodes (say n) trying to make transactions with the malicious node, then the probability that the malicious node gets caught at least once follows a binomial distribution

$$\Pr[X > 0] = \sum_{k=1}^{n} \binom{n}{k} p^k (1-p)^{n-k}.$$

Chapter 4

Analysis of correctness and performance

Up to this point we described our system specification in detail. Of course, specification alone does not establish any truths. In this chapter, we evaluate our system analytically. First we show it has the desired properties. That is, the properties of the consensus protocol and the validation protocol (Definition 6 and Definition 9 respectively) should hold. Then we analyse the performance, especially the throughput, and show that it out performs classical blockchain systems. Finally, we consider the case where the $n \geq 3t + 1$ assumption is violated, and show the effect of it in our system.

4.1 Correctness in the presense of faults

Our first objective is to show that Definition 6 holds for our consensus protocol. Then, building on top of it, we show Definition 9 holds for the validation protocol. The resulting theorem shows that only using CP blocks in the consensus algorithm implies consensus on TX blocks.

4.1.1 Correctness of the consensus protocol

We begin our analysis by establishing truths on the four properties in Definition 6, namely agreement, validity, fairness and termination for an arbitrary round. 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 8), 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 at least $n-t$ honest nodes in $\mathcal{F}_{r+1}$ which follows from our assumption, 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 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, and then 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 6 holds when assuming the existance of some $\mathcal{F}_r$. Thus, to proof the whole of Definition 6, 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 6).*

Proof. We proof using mathematical induction.

In the base case ($r = 1$), agreement, validity fairness and termination follows directly from the bootstrap protocol. 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 the validation protocol

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 the properties except liveness in Definition 9 can be satisfied.

Theorem 2. *The validation protocol satisfies agreement and validity properties.*

Proof. We proof the agreement property by contradiction. Without loss of generality, suppose for some transaction with TX block t, node u decides *valid* but node v decides *invalid*. Then there exist a fragment $F' = \{\ldots, t', c'\}$ which u received that contains a valid pair of t—t'. There also exist a fragment $F'' = \{\ldots, t'', c''\}$ 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 3. Due to Theorem 1, we have $c' = c''$, otherwise the result would be *unknown*. Since $c'(= c'') = \langle \mathsf{H}(t'), \ldots \rangle$ we must have $\mathsf{H}(t') = \mathsf{H}(t'')$ and $t' \neq t''$ (because t'' is invalid). In other words, the sender of F'' must be able to create some t'' that has the same digest as t'. But this is only possible if the adversary can compute the inverse of $H(\cdot)$ with non-negligible probability. Thus we have a contradiction and this completes the agreement proof. Our protocol uses Algorithm 5 which is also the validity definition, this completes the validity proof. □

Theorem 2 shows that agreement on CP blocks would lead to agreement on TX blocks when the nodes are running the validation protocol. Like in our prior work [40], we call this behaviour implicit consensus. One of

the main advantages of our scheme 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.

A stronger version of the validity definition exists. That is, if two nodes are making transactions with each other, then the resulting transaction is always valid in addition to our current validity definition. Under our purely asynchronous model, we cannot guarantee this stronger version. Since the adversary can delay any message for any amount of time, it can make sure all `tx_req` messages are delivered in a round later than the round which the message is sent. Effectively, the transaction pair would always be in different rounds and the validation protocol would not output *valid*. We believe in a relaxed model, i.e. a weakly synchronous model, a stronger validity definition is possible.

4.1.3 Impossibility of liveness

Theorem 2 is a major result that allows significantly improved performance over traditional blockchain systems, but it does not have all the properties of a typical Byzantine consensus protocol. Now we show a negative result, where the liveness property of Definition 9 cannot be attained. Meaning that trasanctions with adversaries cannot always be validated.

Lemma 5. *There exist a valid transactions that cannot be validated eventually.*

Proof. Suppose nodes u and v correctly perfomed the TX protocol which resulted a transaction. Then when u wants to validate it, it does so by sending `vd_req` message to v. v can act maliciously and ignore all `vd_req` message, then the transaction 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 analysis

This section aims to analytically answer the scalability aspect of 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 the consensus protocol, and then the bandwidth requirement for a single transaction. We build on top of those results to analyse the global throughput.

4.2.1 Communication complexity of the consensus protocol

The consensus protocol can bee seen as three parts, so the communication complexity is the sum of these parts. The first part is when every node sends their CP block to all the facilitators, which is $O(Nn)$ since there are N nodes and n facilitators. Or simply $O(N)$ if we consider n as a constant.

The second part is ACS. The communication complexity of ACS is $O(n^2|v| + \lambda n^3 \log n)$ [27], where $|v|$ is the size of largest message and λ is the security parameter (described in Section 3.2). In our system, we wish to understand the scalability property. 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 $|v| = kN$, where k 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.

The third and final part is the dissemination, where the facilitators broadcast the consensus result along with their signatures. For the same reason as the first part, this is also $O(N)$. Thus the combined communication complexity when n is a constant is $O(N)$.

4.2.2 Duration of the consensus protocol

In order to make arguments on bandwidth or throughput in our purely asynchronous model, which are concepts that depend on time, we must make some assumptions regarding our model to make arguments on the duration of the consensus protocol. Note that it is not the same as the time complexity typically used in distributed systems. In analysis of distributed systems, time complexity is often in terms of the 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 [27]. 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, we add the following to our computational model. For every unit of communication, we assume they take some non-negligible but constant time to perform. Hence, from Section 4.2.1 and the fact that the consensus protocol uses a constant number of rounds, it follows that the consensus protocol has a duration of $O(N)$.

4.2.3 Communication cost for transactions

With the analysis on the duration of the consensus protocol, we are ready to analyse the amount of data required to be transmitted over a link per transaction, which we call the communication cost per transaction. To create and then validate a transaction, the communication cost per transaction 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

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

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

4.2.4 Linear global throughput

Using our results so far, we are able to analyse the global throughput. First we clarify the bandwidth definition, which is "the data rate at which a network link or a network path can transfer" [37]. Now, suppose every node has some fixed bandwidth C per link, N links and are making transactions at r_{tx} per second. Then we have the inequality

$$NC \geq r_{\text{tx}} l,$$

where l is the length of the the agree fragment as before. The inequality suggests that the rate for which transactions and validations are made cannot exceed the total bandwidth of all the links.

We note that the inequality does not hold if the node is only transacting with a subset of the population. This is because it cannot use all the bandwidth available in all the links. In the extreme case, if the node is only transacting with one other node, then it can only use the bandwidth of one link which is only C. However, if that is the case, we can intelligently cache the `vd_resp` messages as described in Section 3.7.4. For this work, we analyse the worst case where every node transacts with a random node from the population, and a new `vd_resp` must be sent for every `vd_req` message.

Consider the case where the system is making use of all the bandwidth, i.e. $NC = r_{\text{tx}} l$. Recall that l is of $O(N)$, that means LHS and RHS both grow linearly with respect to N. Hence, there exist some constant r_{tx} that makes use of all the available bandwidth regardless of N. Finally, if every node in

the network is transacting at a constant rate, then the global throughput (in terms of transactions per second) is linear w.r.t. the population size N. If the system is not making use of all the bandwidth. We also maintain a constant transaction rate by the same argument. Thus a global throughput is also linear w.r.t. N. We verify both of these claims experimentally in Chapter 5.

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 (malicious) and $N - t$ are white (honest). If n balls are drawn uniformly at random without replacement, what is the probability that more than $\lfloor \frac{n-1}{3} \rfloor$ 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 [42]. 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 4.1. 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 with "faster-than-exponential fall-off" [42], 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. We can use tail inequality to bound the probability of picking more than $\lfloor \frac{n-1}{3} \rfloor$ black balls when only $n\alpha$ are black where $0 \leq \alpha \leq \lfloor \frac{n-1}{3} \rfloor / n$. The tail inequality is

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

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

$$\tau = \frac{\lfloor \frac{n-1}{3} \rfloor + 1}{n} - \alpha$$

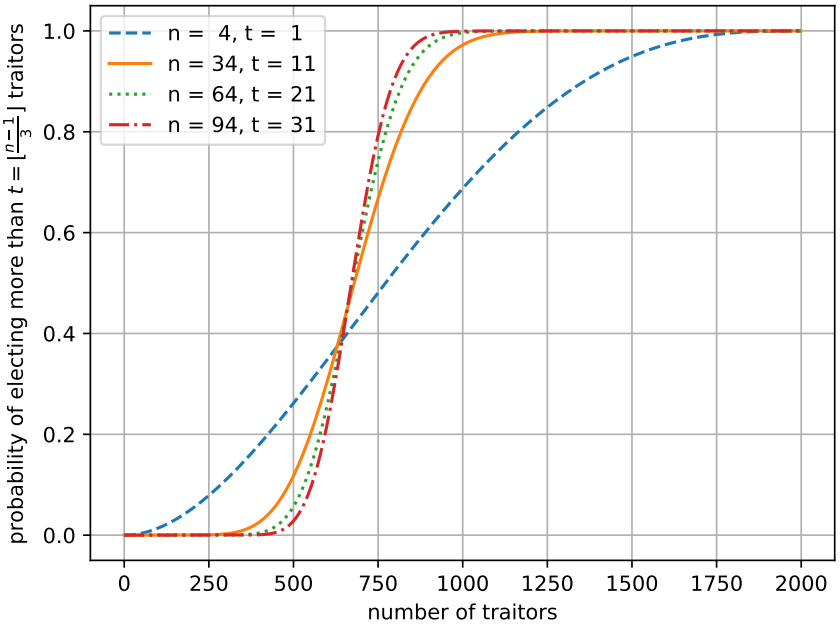


Figure 4.1: Plot of the probability of selecting more than $\lfloor \frac{n-1}{3} \rfloor$ black balls for different value of t with N fixed at 2000.

Putting τ back into the tail inequality we get the following bound.

$$\Pr[X \geq \lfloor \frac{n-1}{3} \rfloor + 1] \leq e^{-2\left(\frac{\lfloor \frac{n-1}{3} \rfloor + 1}{n} - \alpha\right)^2 n}$$

The bound is not tight, but it is useful for for picking parameters. For some fixed n, since $0 \leq \alpha \leq \lfloor \frac{n-1}{3} \rfloor / n < (\lfloor \frac{n-1}{3} \rfloor + 1)/n$, the probability is maximum when the squared term is minimum at $\alpha = \lfloor \frac{n-1}{3} \rfloor / n$. The probability is minimum when the squared term is maximum at $\alpha = 0$. Hence, if n is known, then we can pick a α such that the probability becomes small. On the other hand, if α is fixed, but small, we may increase n to achieve the same. To put this into perspective, suppose $n = 1000$ and $\alpha = 1/5$, i.e. a fifth of the nodes are malicious. Then the probability to draw more black balls than the threshold is only 2.6×10^{-16}.

Chapter 5

Implementation and experimental results

To build confidence of the horizontal scalability claim from our analytical results, we must verify the same property experimentally. Hence, we give an open source prototype implementation of our system. Our implementation runs on a network with over 1000 node. Each node send transactions to each other autonomously, which allows us to evaluate our analytical claims.

We begin this chapter with a description of the implementation in Section 5.1. Then, we move on to describing our experimental setup in Section 5.2 which includes a discussion on the system parameters and the physical infrastructure which we use. Finally, Section 5.3 and onwards follows the same structure as the Performance analysis (Section 4.2), where we analyse the performance experimentally rather than analytically and compare both results.

5.1 Implementation

The prototype implementation can be found on GitHub.

`https://github.com/kc1212/consensus-thesis-code`

It implements the three protocols and the Extended TrustChain discussed in Chapter 3. We also implement two optimisations—privacy preserving validation protocol using compact blocks (Section 3.7.3) and optimised validation protocol using cached agreed fragments (Section 3.7.4). It is written in the event driven paradigm, using the Python programming language[1]. We use the Twisted[2] library for networking.

The structure of the implementation is primarily made up of three modules—`acs`, `trustchain` and `node`. `acs`, as its name suggests, implements ACS.

[1] `https://www.python.org/`
[2] `https://twistedmatrix.com/`

ACS uses erasure code in one of its subprotocols (reliable broadcast). Thus we use the liberasurecode[3] library for its Reed-Solomon error correcting code functionality. An implementation detail is that liberasurecode cannot create more than 32 code blocks[4], we discuss the effect of this in Section 5.2. The `acs` module provides a small interface to the caller to start and stop the consensus process and also retrieve results. The `trustchain` module implements the Extended TrustChain data structure. It also provides the essential algorithm necessary to interact with Extended TrustChain such as $\mathsf{new\_tx}(\cdot)$, $\mathsf{new\_cp}(\cdot)$ and $\mathsf{agreed\_fragment}(\cdot)$. Finally, the `node` module ties everything together. It implements the the consensus protocol, 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 ideal in real world scenarios where nodes may have limited resources (e.g. sockets). But as a prototype, it is sufficient to run a network of over a thousand nodes and experiment with it.

Finally, the cryptography primitives we use are SHA256 for hash functions and Ed25519 for digital signatures. Both of which are provided by libnacl [5].

5.2 Experimental setup

The the goal of the experiment is to run the three protocols—consensus protocol, transaction protocol and validation protocol—simultaneously and analyse the communication cost, the consensus duration and the global throughput. We investigate these properties under the following four parameters.

1. The transaction rate r_{tx} per node. This is not comparable to the others because it is fixed at 2 TX/s. Nevertheless, it is a good value because, as we show later, it hits a bottleneck in extreme cases which helps us understand the limitations of our design.

2. The number of facilitators $n \in \{4, 8, \ldots, 32\}$. The maximum n is 32 because the limitation in liberasurecode mentioned in Section 5.1, but our results give a good dindication of how our system may function for larger numbers of n.

3. The population size $N \in \{200, 300, \ldots, 1200\}$. N stops at 1200 is due to our physical setup, which we describe next.

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

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

[5] https://pypi.python.org/pypi/libnacl

4. The two modes of transaction. The first mode is the *fixed-neighbour* mode where nodes only transact with their immediate neighbour which are predetermined. This minimises the data volume per validated transaction because agreed fragment can be cached. The second mode is in the other extreme which we call the *random-neighbour* mode, where every transaction is with a random node out of the N nodes in the system. It is unlikely that the agreed fragments can be reused.

The experiment is run on the DAS-5[6]. 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 30 machine, for each machine we use up to 40 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 (minus the reserved ones). But 40 nodes each need 1200 TCP connections which is 48000 TCP connections per machine and that is inching close to the limit. 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. Nevertheless, running the system with 1200 nodes gives a good indication of its scalability properties as we show later.

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.

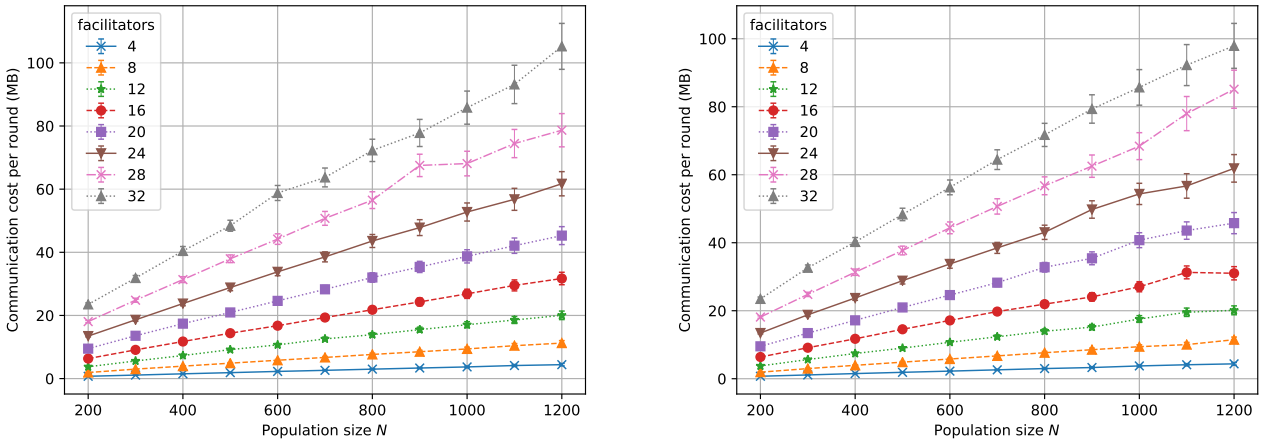
Finally, since Bitcoin transactions are approximately 500 bytes [44], we use a uniformly random transaction size sampled between 400 and 600 bytes.

5.3 Communication cost for the consensus protocol

The remainder of this chapter follows the same structure as the performance analysis in Section 4.2. We check our analytical results experimentally and demonstrate horizontal scalability.

Figure 5.1 show the relationship between the communication cost of the consensus protocol per round and the population size. The most important aspect is that these results show a linear increase. This reinforces our analytical result in Section 4.2.1. Note that regardless of whether the transactions are performed with a random neighbour or with a fixed neighbour, the magnitude of the communication cost are similar. Both peak at about 100 MB. This is expected because the consensus protocol is decoupled from the transaction protocol and the validation protocol. Finally, the rate for

[6] https://www.cs.vu.nl/das5/



(a) Transactions are with fixed neighbours.

(b) Transactions are with random neighbours.

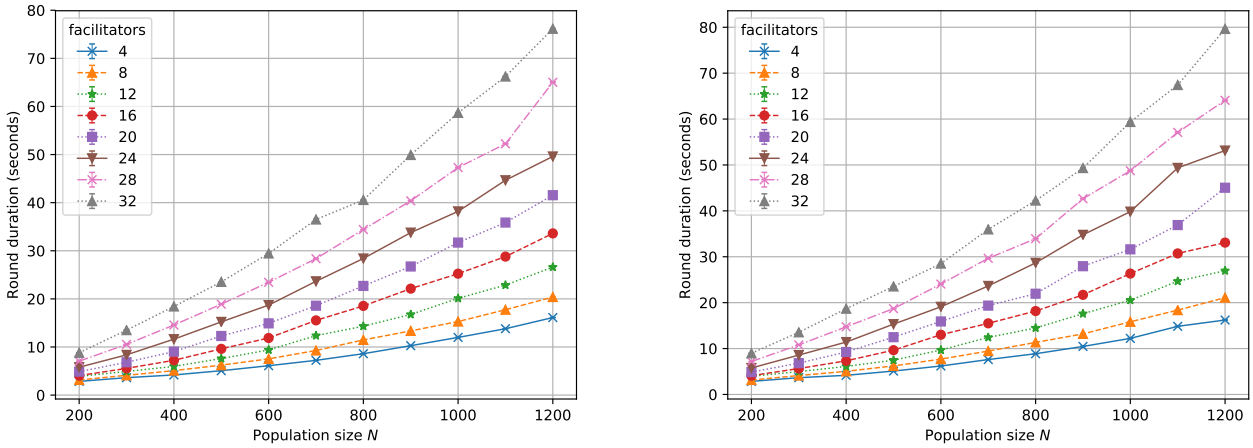
Figure 5.1: Communication cost for the consensus protocol per round increases linearly with population size. The error bars are larger for higher population size or higher number of facilitators is because rounds take longer thus they are repeated fewer times.

which the communication cost increases is higher when the number of facilitators are higher. This is also expected because the ACS algorithm has a n^2 term in its communication complexity.

We are also interested in how communication costs translate to time. Hence, for the same experiment we record the duration in seconds and the result is shown in Figure 5.2. Interestingly, the duration is not entirely linear. We attribute this behaviour to the extra time needed to hash the CP blocks in the consensus result to compute the luck value. Since if N increases, every node must also perform more hash operations. These result do not conform to analytical result in Section 4.2.2. Nevertheless, the difference is minor and there are ways to optimise the luck value computation. For example, the luck value can be computed by the facilitators and are sent with the consensus result. Then the non-facilitator nodes simply use it if there are enough signatures.

5.4 Communication cost for transaction and validation protocols

We argued that the communication cost per verified transaction is of $O(N)$ (Section 4.2.3). To verify the argument, we plot the relationship between the communication cost of for every validated transaction and population



(a) Transactions are with fixed neighbours.

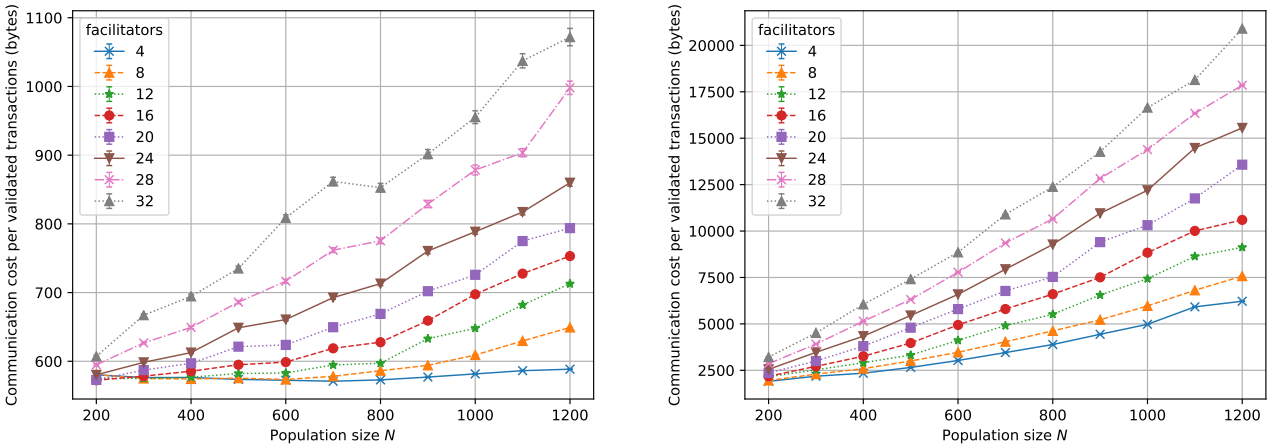
(b) Transactions are with random neighbours.

Figure 5.2: Round duration do not increase linearly with the population size. This is likely due to the additional hashing operation required for larger consensus result.

size in Figure 5.3. We observe a near linear relationship, which is due to the near linear relationship of the communication duration mentioned before. Again, we believe the difference is minor and it is possible to remove the extra overhead.

More interestingly, there is a large difference in communication cost between the two modes of transaction. When transacting with only one neighbour, the communication cost is low because only one agreed fragment needs to be communicated for every round in order to validate all transactions of that round. This is because agreed fragments are cached. On the other hand, if every node is transacting with a random node, then it is likely the case that one agreed fragment needs to be communicated for every transaction. Hence the communication cost we see in Figure 5.3b is much higher than in Figure 5.3a.

Some fluctuations exist in Figure 5.3a, this is due to our caching mechanism. We send validation requests at the same rate as transactions. Upon receiving an (remote) agreed fragment, the caching mechanism inspects all the transactions in the agreed fragment and attempts to verify as many as it can, rather than just the transaction in the original validation request. However, it may be the case that the agreed fragments arrive later than the validation request interval. Then it is possible to have sent two or more validation requests for some transactions in the same local agreed fragment. In this case, the remote would respond with two or more of the same agreed fragments, which results in extra (wasted) communication cost, and this is



(a) Transactions are with fixed neighbours.

(b) Transactions are with random neighbours.

Figure 5.3: Communication cost per verified transaction has similar (near linear) trend as Figure 5.2. Fluctuation for the fixed-neighbour mode exists because the cache mechanism is unpredictable.

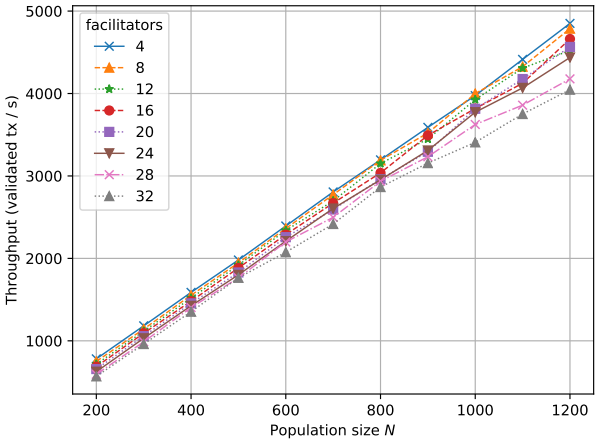
the source of the fluctuation seen in Figure 5.3a. The result in Figure 5.3b reinforces our argument. It is a lot more stable because for every transaction it is almost always the case that an agreed fragment is needed to validate it.

5.5 Linear global throughput

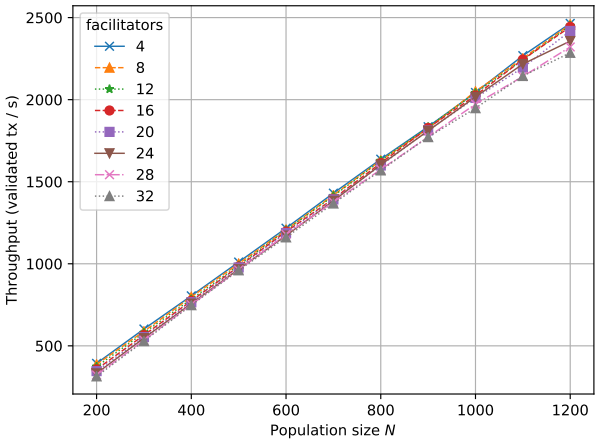
Finally, the global throughput results are shown in Figure 5.4. Evidently, the throughput has a linear relationship with the population size. This result is a strong indication of the horizontal scalability which we aimed to achieve. It also matches with our analytical result.

Note that the throughput decreases slightly as the number of facilitators increases. This is due to the additional communication cost for running ACS with a high number of facilitators. That is, if the network is congested then the nodes may not have enough bandwidth to send validation responses timely.

For Figure 5.4a, the magnitude of our throughput may not be self-evident at first glance. Recall that we fixed our r_{tx} to 2, but how is it possible to have around 4800 transactions per second for 1200 nodes (which is 4 TX/s)? This is due to the way validated transactions are calculated. Transactions are between two parties, hence if every node makes two transactions per second, every node also expects to receive two transactions per second. Hence, for every node, the TX blocks are created at 4 per second. Validation requests are sent at the same rate, which explains the magnitude.



(a) Transactions are with fixed neighbours.

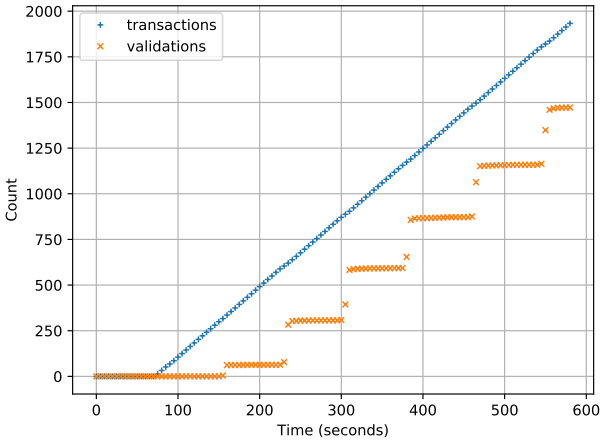


(b) Transactions are with random neighbours.

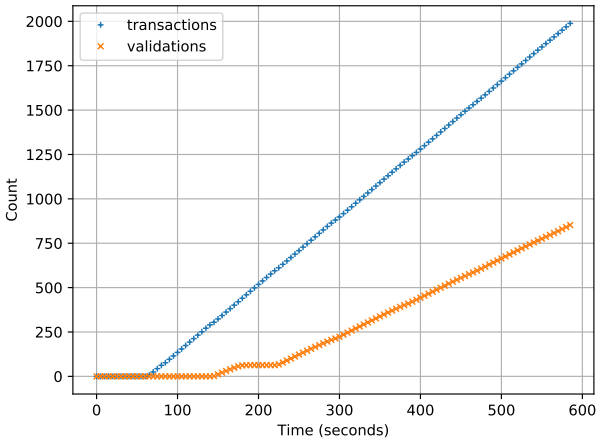
Figure 5.4: Global throughput increases as the population increases when every node transact at the same rate. Using fixed neighbours results in a higher throughput because of the caching mechanism.

The difference in magnitude between Figure 5.4a and Figure 5.4b is caused by the caching mechanism mentioned earlier. If a new agreed fragment needs to be transmitted to validate every transaction then it puts a toll on the network infrastructure. The low transaction rate in Figure 5.4b is caused by fact that the network infrastructure cannot keep up with our demand.

We demonstrate the bottleneck issue from a different perspective in Figure 5.5. The graph is plotted by counting the number of transactions and the number of validated transactions every 5 seconds for one node running in a network of 1200 nodes and 32 facilitators. In Figure 5.5a, the number of validated transaction changes as a step function. This means that transactions are validated in bursts, and the validation protocol can "keep up" with the transactions. Note that the horizontal "lines" where no new validated transactions are made are on average 76 seconds, this matches with the round duration result in Figure 5.2a. On the other hand in Figure 5.5b, the validation protocol clearly cannot "keep up" with the rate which the transactions are made. As a result, the global throughput is lower when transacting with random nodes than only with neighbours.



(a) Transactions are with fixed neighbours.



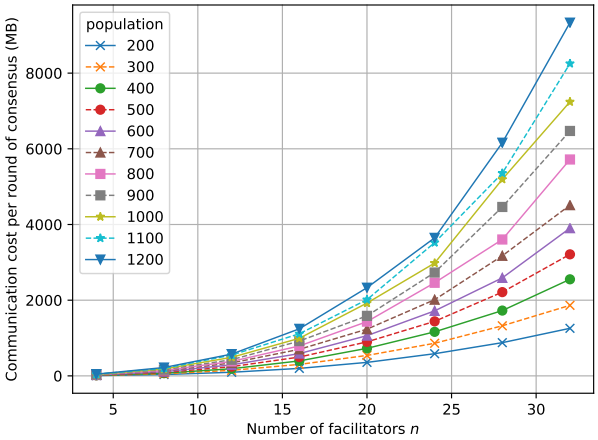
(b) Transactions are with random neighbours.

Figure 5.5: A limitation of our system is when nodes are transacting with many random nodes, then the network cannot keep up with the large number of agreed fragments that need to be communicated between nodes. Hence the slope of validated transactions grows at a slower rate in Figure 5.5b.

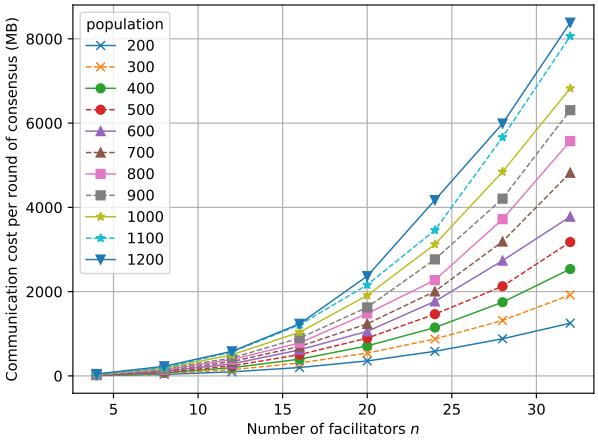
5.6 Communication cost with varying number of facilitators

Up to this point we focused on the effect of communication cost, throughput and so on with respect to the population size. In this section we consider a varying number of facilitators, which gives us an insight on our system performance may perform when the number of facilitators are higher than 32.

Figure 5.6 shows the communication cost of the consensus protocol for varying numbers of facilitators. There is strong evidence that the communication cost grows polynomially. We expect this because of there are polynomial terms in the ACS communication complexity—$O(n^2|v| + \lambda n^3 \log n)$. This also means that the rounds would take longer to complete and transactions would take longer to verify. On the other hand, we get better fault tolerance. Thus a linear increase in fault tolerance (t) costs a polynomial increase in communication cost.



(a) Transactions are with fixed neighbours.



(b) Transactions are with random neighbours.

Figure 5.6: The communication cost of the consensus protocol increase polynomially with respect to the number of facilitators as expected from the ACS communication complexity.

Chapter 6

Conclusion

In this work we introduced an application neutrual blockchain system which we call CheckpointConsensys. We add checkpoint block to the existing TrustChain data structure for use in our consensus protocol. The round based consensus protocol uses ACS as a building block to reach consensus on checkpoint blocks. The consensus result lets nodes elect new facilitators and create new checkpoint blocks. Node make transactions via the transaction protocol, which is adapted from an existing work called True Halves. We introduce a validation protocol which ensure that if agreed fragments for some transaction exists, then nodes reach agreement on the validity of that transaction. The novelty of the validation protocol is that it uses point-to-point communication, i.e. nodes only validate the transactions of interest, this enables our horizontal scalability property.

The research question we asked in Chapter 2 is the following.

> *Is it possible to design a blockchain consensus protocol that is fault tolerant, scalable and can reach global consensus?*

We answer it in the affirmative. Fault tolerance is guaranteed if $n \geq 3t + 1$ by using ACS as a building block. While t may be small compared to the population size N, we show that the probability for the system to fail is low even when $n \geq 3t + 1$ does not hold as long as the proportion of malicious nodes is not close to a third of N. For example, if there the population size is 1000 and 20% of the nodes are malicious, the probability for a round to potentially to fail is bounded below 2.6×10^{-16}. This probability bound would decrease as the population size increases. Horizontal scalability property is demonstrated both analytically and experimentally. Unlike sharding protocols, the property holds regardless of transaction characteristics and needs no parameter selection. Finally, we achieve global consensus on transactions via consensus on checkpoint blocks.

This work is the first step in building a new paradigm for blockchain consensus protocol. It has the potential to efficiently cultivate trust on the

internet in the presence of faults without a central authority. We hope to improve our design by building a concrete application on top of it.

6.1 Future work

While our system has excellent scalability properties, it is not without limitations. Much of it is from the fact that it does not have a concrete application. We do not attempt to prevent the Sybil attack, spam or DoS because the accuracy of the defence depends on the understanding of the application. For instance, without any application it would be impossible to distinguish between a very active node from a spammer because the content of the transaction (m) carry no meaning.

Our experiment is carried out in a somewhat idealised world (DAS-5). To understand our system further, we hope to experiment on a real platform with real users such as Tribler [36]. For instance, the TrustChain in Tribler stores the number of bytes uploaded and downloaded between users, but it has no global consensus. It would be useful to integrate our consensus protocol into Tribler to prevent misreporting attacks. More importantly it gives us an opportunity to observe the characteristics of our system in the real world and how it evolves with new features or alterations.

The fault tolerance property is adaquate for permissioned systems where nodes have relatively more trust between each other. However, it is not adaquate for permissionless systems, which usually aim to tolerate fault that involve a minority of the network, which can be just shy of 50%. For future work, we propose the following two ways to improve fault tolerance. First, use a reputation mechanism to select facilitators rather than simply selecting them randomly so that faulty nodes are less likely to be selected. This technique of course heavily depends on the application, for P2P file sharing systems the reputation score may be the number of bytes contributed to the network. Second, instead of using ACS we use a proof-of-stake (PoS) scheme which means that nodes with more stake in the system gets more vote. In an ideal PoS scheme, the system can tolerate any fault if the majority of the stake holders are honest.

Finally, as mentioned in Section 3.7.2, we wish to evaluate our system (analytically and experimentally) in the permissionless settings. This would enable our protocol to be applied to a much wider range of applications.

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] James Basden and Michael Cottrell. *How Utilities Are Using Blockchain to Modernize the Grid*. Harvard Business Review, Mar. 2017. URL: https://hbr.org/2017/03/how-utilities-are-using-blockchain-to-modernize-the-grid (visited on 04/07/2017).

[4] 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.

[5] Bitcoin Project. *Bitcoin Developer Guide*. URL: https://bitcoin.org/en/developer-guide (visited on 04/07/2017).

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

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

[8] Bitcoin Wiki. *Transaction Malleability*. Aug. 2015. URL: https://en.bitcoin.it/wiki/Transaction_Malleability (visited on 25/06/2017).

[9] Christian Cachin. 'Architecture of the Hyperledger blockchain fabric'. In: *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*. 2016.

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

[11] Bram Cohen. 'Incentives build robustness in BitTorrent'. In: *Workshop on Economics of Peer-to-Peer systems*. Vol. 6. 2003, pp. 68–72.

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

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

[14] Reenita Das. *Does Blockchain Have A Place In Healthcare?* Forbes, May 2017. URL: https://www.forbes.com/sites/reenitadas/2017/05/08/does-blockchain-have-a-place-in-healthcare/ (visited on 04/07/2017).

[15] 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.

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

[17] Patrick T Eugster et al. 'Epidemic information dissemination in distributed systems'. In: *Computer* 37.5 (2004), pp. 60–67.

[18] 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.

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

[20] 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.

[21] Eleftherios Kokoris-Kogias et al. 'OmniLedger: A Secure, Scale-Out, Decentralized Ledger.' In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 406.

[22] Leslie Lamport, Robert Shostak and Marshall Pease. 'The Byzantine generals problem'. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3 (1982), pp. 382–401.

[23] Joao Leitao, Jose Pereira and Luis Rodrigues. 'Epidemic broadcast trees'. In: *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*. IEEE. 2007, pp. 301–310.

[24] Eric Lombrozo, Johnson Lau and Pieter Wuille. *Segregated Witness (Consensus layer)*. Dec. 2015. URL: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki (visited on 25/06/2017).

[25] Loi Luu et al. 'A secure sharding protocol for open blockchains'. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 17–30.

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

[27] 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.

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

[29] Namecoin Developers. *Namecoin*. URL: https://www.namecoin.org/ (visited on 25/06/2017).

[30] Nebulous Inc. *Your decentralized private cloud*. URL: https://sia.tech/ (visited on 25/06/2017).

[31] Steffan Norberhuis. *MultiChain: A cybercurrency for cooperation*. Dec. 2015. URL: http://resolver.tudelft.nl/uuid:59723e98-ae48-4fac-b258-2df99d11012c.

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

[33] Marshall Pease, Robert Shostak and Leslie Lamport. 'Reaching agreement in the presence of faults'. In: *Journal of the ACM (JACM)* 27.2 (1980), pp. 228–234.

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

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

[36] Johan A Pouwelse et al. 'TRIBLER: a social-based peer-to-peer system'. In: *Concurrency and computation: Practice and experience* 20.2 (2008), pp. 127–138.

[37] Ravi Prasad et al. 'Bandwidth estimation: metrics, measurement techniques, and tools'. In: *IEEE network* 17.6 (2003), pp. 27–35.

[38] Giulio Prisco. *Walmart Testing Blockchain Technology for Supply Chain Management*. Bitcoin Magazine, Dec. 2016. URL: https://bitcoinmagazine.com/articles/walmart-testing-blockchain-technology-for-supply-chain-management/ (visited on 04/07/2017).

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

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

[41] Laura Shin. *Bitcoin Is Mired In A Civil War. Can This Proposal Save It?* Forbes, 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).

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

[43] Alex Tapscott and Don Tapscott. *How Blockchain Is Changing Finance*. Harvard Business Review, Mar. 2017. URL: https://hbr.org/2017/03/how-blockchain-is-changing-finance (visited on 04/07/2017).

[44] TradeBlock. *Analysis of Bitcoin Transaction Size Trends*. Oct. 2015. URL: https://tradeblock.com/blog/analysis-of-bitcoin-transaction-size-trends (visited on 14/07/2017).

[45] 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.

[46] *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).

[47] 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.

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

[49] 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 protocol example

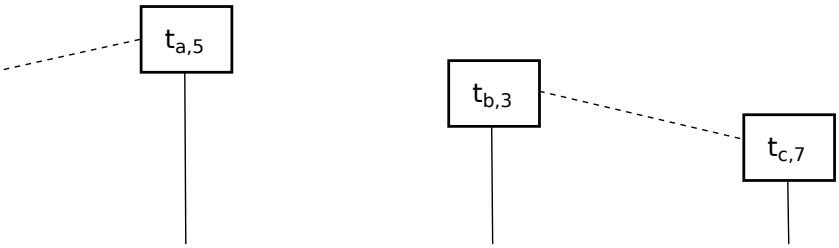


Figure A.1: We begin in a state where some facilitators $\mathcal{F}_{r-2}$ just agreed on the consensus result $\mathcal{C}_{r-1}$ but have not yet propogated it yet.

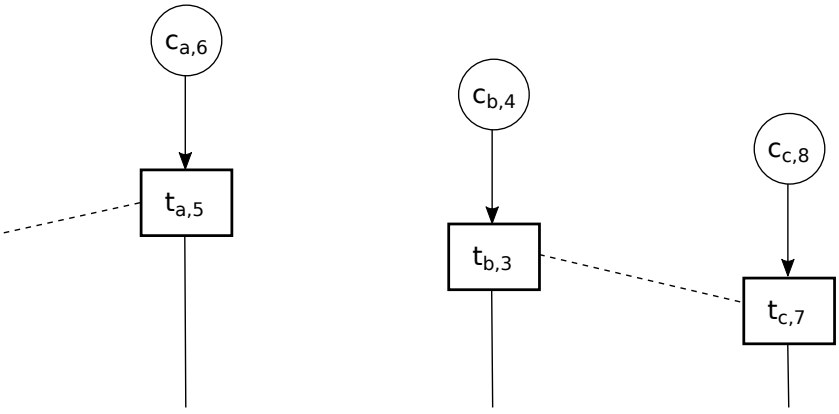


Figure A.2: The facilitators propogates $\mathcal{C}_{r-1}$. Upon receiving it, the nodes perform two tasks; (1) elect new facilitators $\mathcal{F}_{r-1}$ by selecting the first n nodes ordered by $\mathsf{H}(\mathcal{C}_{r-1}||pk)$, and (2) create new CP blocks—$c_{a,6}, c_{b,4}$ and $c_{c,8}$. $\mathsf{H}(\cdot)$ is a cryptographically secure hash function and pk is the public key of the respective node.

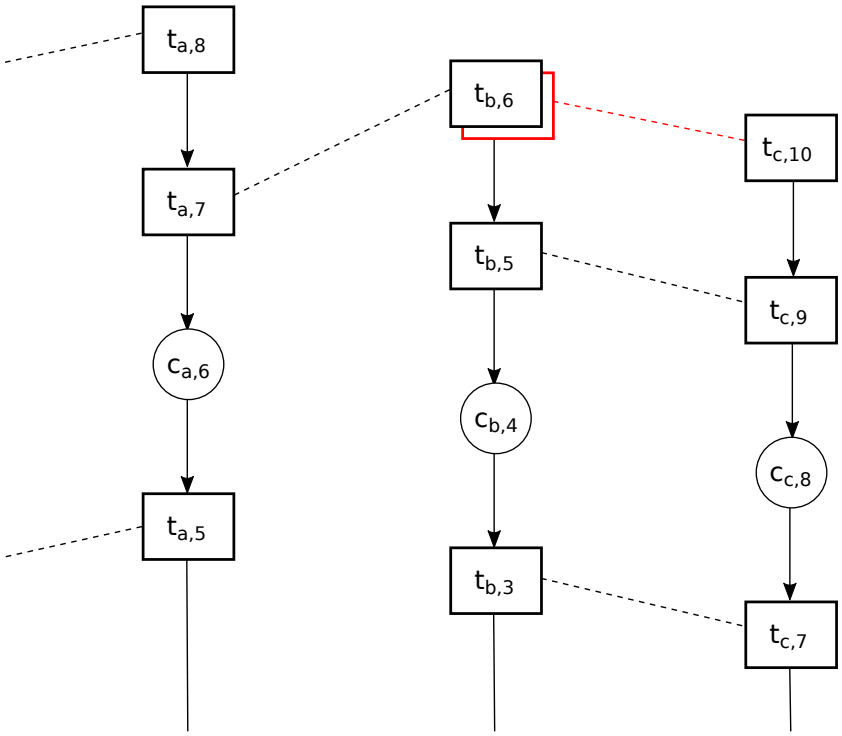


Figure A.3: Nodes now send their new CP blocks to the new facilitators $\mathcal{F}_{r-1}$. While $\mathcal{F}_{r-1}$ is trying to reach consensus on a set of CP blocks, nodes carry on making transactions as usual (concurrently). We remark that the to-be-agreed consensus result $\mathcal{C}_r$ is created using CP blocks from the previous round—$c_{a,6}, c_{b,4}$ and $c_{c,8}$.

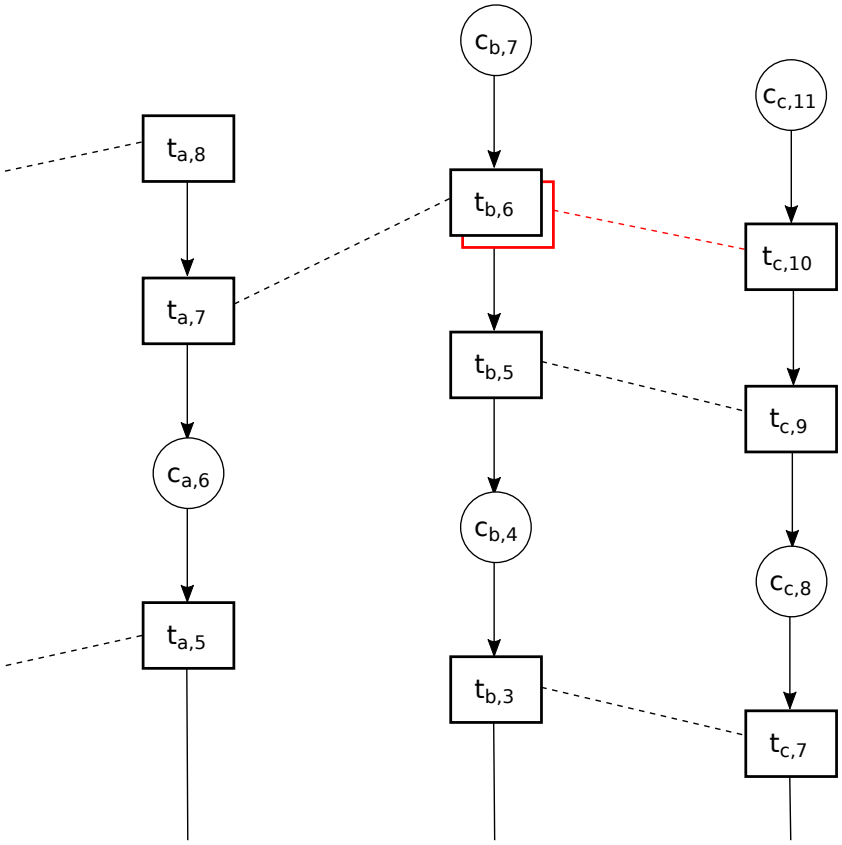


Figure A.4: Finally, when $\mathcal{F}_{r-1}$ decides on $\mathcal{C}_r$ (which should contain $c_{a,6}, c_{b,4}$ and $c_{c,8}$) and disseminates it. Nodes compute new facilitators $\mathcal{F}_r$ and create new CP blocks.