A Blockchain Consensus Protocol With Horizontal Scalability

Kelong Cong

TUDelft
Delft
University of
Technology

A 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

23rd August 2017

Author
  Kelong Cong (4520009)

Title
  A 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 systems. However, scalability remains a key challenge. Without a horizontally scalable solution, where performance increases by adding more nodes to the system, blockchain systems remain unsuitable for ubiquitous use. We design a novel blockchain system called CHECO. Each node in our system maintains a personal hash chain, which only stores transactions that the node is involved in. A consensus is reached on special blocks called checkpoint blocks rather than on all the transactions. Checkpoint blocks are effectively a hash pointer to the personal hash chains; thus a single checkpoint block may represent an arbitrarily large set of transactions. We introduce a validation protocol so that any node can check the validity of any transaction. Since it is a point-to-point protocol, we achieve horizontal scalability. We analytically evaluate our system and proof a number of highly desirable correctness properties. Further, we give a free and open-source implementation of CHECO and evaluate it experimentally. Our results show a strong indication of horizontal scalability.

Preface

Since early 2013, the time when I mined my first Bitcoin, I became cautious of the security and privacy issues surrounding centralised systems and became interested in building a decentralised society. Following the Snowden revelations, the interest grew into a desire that has stayed with me ever since. Today, many blockchain systems are mature, but still not yet ready for ubiquitous use due to the scalability limitation. Many of us still do not use Bitcoin to purchase everyday goods, online or offline. This limitation, as well as my ambitions, led me to this thesis, with the goal of creating a truly horizontally scalable solution.

Kelong Cong

Delft, The Netherlands
23rd 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 to 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 root name server in the domain name system, the root certificate authority in our public key infrastructure, 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 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 [1] and led to the European debt crisis.

Ironically, during the midst of the financial crisis, Satoshi Nakamoto published the Bitcoin whitepaper [2]. 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 incorruptible with high probability even if there are malicious parties that aim to undermine the system.

The primary innovation is the consensus mechanism which prevents double spending but also incentivises anyone (with adequate hardware) to participate and keep the network 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* (PoW). Where miners (parties that run the Bitcoin network) collect transactions and compete 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.

---

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

The miner is also rewarded with new coins and transaction fees. It is important to note that every block 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 overall it has grown into an enormous system. Its power consumption is as high as the Republic of Ireland [3]. Its market capitalisation, at the time of writing, is over 60 billion USD and growing (see Figure 1.1). Many online marketplaces are using Bitcoin. For example, Steam[3] and

Figure 1.1: Market capitalisation and price of Bitcoin, graph is extracted from [4].

Amazon[4]. Due to its success, people from many different disciplines began investigating in ways to use blockchain technology. This includes finance [5], health care [6], logistics [7] and energy [8].

Sadly, as traditional blockchain systems began to gain popularity, their limitations also became apparent. Bitcoin has the infamous 7 transactions per second upper bound [9], which is due to the fact that blocks are fixed to 1 MB and 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]. The backlog even reached 100,000 in May 2017, which meant new transactions would

---

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

take at least 11 hours to be written to the blockchain [10]. This issue has plagued the Bitcoin community for some time and is the root cause for the block size debate, which some call a civil war [11]. Parties which are for an 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 propagate through the network. It also requires a hard (non-backwards compatible) fork, which risks consensus failure and devaluation. A recent empirical study by Croman et al. [12] has shown that increasing the block size or the block interval may help. But given the bandwidth and latency constraints of today's network, it is not possible to have more than 27 transactions per second. To achieve transaction rates higher than that, it will lead to unfairness for nodes that do not have a good network connection. Thus, fundamental changes are necessary to run at the scale of centralised payment processors such as Visa, which is in the order of tens of thousands transactions per second [13].

The Bitcoin community understands the issue and many proposals exist with the aim to make Bitcoin more scalable. The most prominent one is Segregated Witness or SegWit [14]. It moves the signature data to the end of the block and uses a variable block size up to 4 MB. The effect is that the first 1 MB of the block is still backwards compatible, thus there is no need for a hard fork. It also solves the transaction malleability problem [15] which paves the way for off-chain transactions via the Lightning Network [16]. However, some fear SegWit is not sufficient and wish to see a hard fork that uses an "Adjustable Blocksize Cap" that increases the maximum block size even further to 8 MB [17]. The ongoing civil war resulted in a Bitcoin fork, called Bitcoin Cash [18]. At the time of writing, it is too early to speculate how these proposals and forks may affect the scalability properties of Bitcoin. But the conclusion that we can draw from the scalability debate is that the Bitcoin design is fundamentally not scalable, and it is highly non-trivial to "patch" the problem.

Moreover, Bitcoin is far from distributed. It is more accurate to say Bitcoin is made in China [19]. The combination of cheap electronics and electricity has made China one of the most profitable place to mine Bitcoins. At the time of writing, China holds about 80% of the hashing power [20], effectively putting it in control of the 60 billion-dollar ecosystem. If the government decides to flex its totalitarian muscles, then it would spell the end of the ecosystem.

1.1 Contributions

In this work, we take the advice of Croman et al. [12] and Vukolić [9] and rethink the blockchain architecture. Our primary insight came from observing the differences between how value transfer work in the real world and how they work in early blockchain systems. 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 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 owner would report the incident to some central authority, e.g., the police. On the contrary, in blockchain systems, every transaction, regardless of whether it is valid or invalid, are sent to the miners, which can be seen as a collective authority. This consequently leads 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 themselves do not reach consensus, but nevertheless verifiable at a later stage by any node in the network. A verified transaction is guaranteed to be tamper-proof and irrefutable. Our main contributions are the following.

- We formally introduce a blockchain system—CHECO[6]. It uses individual hash chains and checkpoint blocks on every node to achieve horizontally scalable in the general case for the first time.

- We formally analyse CHECO to ensure correctness as defined in our system architecture.

- We provide an implementation and then experiment with up to 1200 nodes, our results show strong evidence of horizontal scalability.

1.2   Document structure

We begin with the problem description in Chapter 2, it first describes the state-of-the-art in scalable blockchain systems and their shortcomings, which leads to 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 Chapter 6 and discuss future work.

---

[6]Derived from *che*ckpoint *co*nsensus.

Chapter 2

Problem description

We saw that early blockchain systems face a scalability problem. In particular, the use PoW 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 and identify their limitations. Then, we state our research question which aims to overcome some of these.

2.1 Related work

Blockchain research has seen a surge in recent years from both the industry and academia, this resulted in a myriad 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. A summary is given in Table 2.1.

Table 2.1: Summary of the scalability properties of many blockchain systems. Scalability gets better from left to right.

Not scalable	Somewhat scalable	Limited horizontal scalability	Full horizontal scalability
Bitcoin [2] Ethereum [23] etc.	Hyperledger [21] ByzCoin [24] Solidius [26]	Elastico [22] OmniLedger [25]	CHECO (this work)

2.1.1 Early blockchain systems

This category represents the baseline, which are systems with a probabilistic consensus algorithm. That is, transactions never reach consensus with a probability of 1. Given sufficient computational power, adversaries are able to undo any transaction. The typical examples are PoW based systems such as Bitcoin, Ethereum and other Altcoins. In Bitcoin, nodes compete in solving puzzles. The first node solves the puzzle has the privilege to create a new block. The level of consensus of

a block containing many transactions is determined by how deep it is in the Bitcoin blockchain, which is also called the number of confirmation. The probability of a block being orphaned drops exponentially as the depth increases [2]. 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 was no Internet-scale consensus protocol. Bitcoin is also fault-tolerant because attackers can not outpace honest users in finding new blocks unless they have a majority of the computational power (assuming reasonable measures are taken to prevent the selfish mining attack [27]). 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 Chapter 1, 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 scale.

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 [16] and Duplex Micropayment Channels [28].

Off-chain transaction systems are implemented using multi-signature addresses [29] and hashed time-locked Bitcoin contracts [30]. In the simplest case, if two parties wish to make frequent transactions, they open a payment channel in the form 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, which 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 occurs, the counterparty can drain all the Bitcoins in the channel. To close the channel, the parties simply broadcast the latest net settlement to the Bitcoin network, which 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. For example, if a channel exists between node A and B, and a channel exists between node B and C, then A can pay C via the intermediary B. The result is that there is no need to create a payment channel with every node in the network. If some key nodes, acting as payment hubs, are well-connected, then nodes can make off-chain transactions to a large number of other nodes by opening only one channel with a payment hub.

The advantage of such systems is that they act as add-ons to Bitcoin, which

6

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 implementations 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, payment channel complicates user experience and leads to centralisation. As we mentioned earlier, each node must deposit some Bitcoins into a multi-signature account, so the users are responsible for picking 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. Furthermore, creating channels with sufficient balance and also keeping it online to act as a router is expensive. A casual user is not capable of such tasks, thus it leads to centralisation. Payment channels, in theory, solve 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 uses traditional Byzantine consensus algorithms such as PBFT [31]. 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 technique is often used in permissioned systems where the validators must be predetermined, for example, Hyperledger Fabric [21].

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 [32, Section 5.2]. Further, in contrast to PoW, 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 [32, Section 5.2]. Moreover, the validating peers are predetermined which makes the system unsuitable for the open internet.

2.1.4 Combining PoW with Byzantine consensus

Recent research has developed a class of hybrid systems which uses PoW for committee election, and Byzantine consensus algorithms to agree on transactions. Some examples are ByzCoin [24] and Solidus [26].

This technique overcomes the early blockchain scalability issue by delegating the transaction validation to the Byzantine consensus protocol (e.g., PBFT in Byz-Coin [24]) while keeping the permissionless nature by using PoW. A major tradeoff

---

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

7

of such systems is that they cannot guarantee correctness when there is a large number of malicious nodes (but less than a majority). For 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 difficult to solve because the committee is always much smaller than the population size which typically has more than t Byzantine nodes. Classical blockchain does not have this problem because they do not use Byzantine consensus. Further, due to the fact that these systems 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, also known as 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 inter-shard protocol is needed for transactions that involve nodes from more than one shard. The number of shards grows linearly 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 [22] and Omni-Ledger [25].

The biggest limitation of sharding is that it is only optimal if 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. Furthermore, using smaller shards is a security risk. Every shard is fault-tolerant up to at most a third of the shard size. If the total number of malicious nodes remain fixed, then as the shard size decreases, it is more probably for a shard to fail.

2.1.6 Blockchain without global consensus

Tangle [34], Corda [35] and the original TrustChain [36] avoid the need of global consensus. In these schemes, every node has their own hash 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 [37].

On the other hand, the application and security properties are 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. To the best of our knowledge, only the two systems described in Section 2.1.5 have experimentally demonstrated horizontal scalability. However, these techniques rely on sharding; thus the performance highly depends on the transaction characteristics and the shard size parameter. 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, the accounting of detailed internet traffic in Tribler [36], [38] and decentralised market in Tribler[2]. Hence our research question is the following.

> *How can we design a blockchain consensus protocol that is fault-tolerant, horizontally scalable, and able to reach global consensus?*

To make sense of our research question, we must first explain *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 [39], and the application in Siacoin is cloud storage [40]. 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 PoW, because it is not concerned with the structure or the semantics of the transactions. In this work, we focus on the blockchain consensus protocol rather than any specific application because it is the scalability bottleneck and is necessary for any blockchain based application.

In the remainder of this section, we expand on our research question and visit each aspect in detail. At the end of this section, we describe some issues that out of the scope of this work, which is the result of not focusing on any particular application.

Global consensus

Having consensus is essential in blockchain systems. It stops many types of malicious activities because the agreed state, or a representation of it, must have the consent of the honest nodes in the network. For example, if two honest nodes attempt to verify the same transaction, it cannot be the case that one node thinks that the transaction is valid, but the other thinks it is invalid, as long as the number of malicious nodes is below the threshold that the system can tolerate.

---

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

Fault-tolerance

Similar to many blockchain systems, 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 omitting messages to colluding with each other to undermine the whole system. As a result, just like cryptocurrencies, our transactions should be tamper-proof and irrefutable.

Horizontal scalability

To enable ubiquitous use, we demand horizontal scalability. That is, the global transaction rate should increase as more nodes join the network. BitTorrent [37] is an example of such a system, where peers interact with each other to exchange files without any global bottleneck.

Limitations

Since our consensus protocol is application neutral, it does not attempt to prevent the Sybil attack [41]. A good defence mechanism requires application specific information. For example, a social network based Sybil defence mechanism uses random walks in real-world social graphs [42]. Online marketplaces such as Amazon use the rating of buyer and sellers. Cryptocurrencies use PoW to decide on new blocks. For the same reason, we do not explicitly consider spam or denial of service attacks because, without a concrete application, it is hard to distinguish a super active user from an attacker. For this work, we assume that it is the role of the application developer to find a reputation mechanism or a trust model suitable for defending against such attacks.

Chapter 3

System architecture

In this chapter, we break away from the early blockchain paradigm and introduce a novel system that is application neutral, horizontally scalable and will never have a block size debate. In particular, our system consists of three protocols—the consensus protocol, the transaction protocol and the validation protocol. It is based on our prior research on TrustChain [36] at Delft, where every node independently interacts with each other via their own hash chain[1].

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, and then the three protocols which make up the complete system. Finally, we discuss a few variations of our design and their respective tradeoffs in Section 3.7.
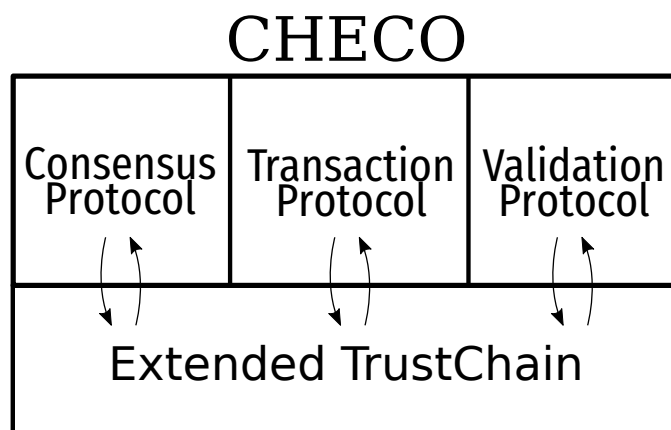
3.1 System overview

CHECO

Consensus Protocol	Transaction Protocol	Validation Protocol
	Extended TrustChain	

Figure 3.1: The four components of CHECO.

---

[1]TrustChain is originally called MultiChain.

The architecture of CHECO is visualised in Figure 3.1. It consists 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 synchronization 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.

TrustChain and Extended TrustChain

We explain the standard TrustChain first and then the Extended TrustChain. Our description of the standard TrustChain has a minor difference when compared to the description in [36], we remark the difference when it occurs.

In TrustChain, every node has a personal hash chain. Initially, the chain only contains a genesis block generated by the nodes themselves. When node A wishes to make a 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, either by accident or for malicious intent, 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 $t_{b,5}$ is spent twice.

Extended TrustChain introduces a new block type—checkpoint (CP) block. The primary function of the CP block is to represent the state of a node, in the form of a hash pointer. A collection of CP blocks from all nodes represents the state of the whole system. Therefore, nodes need to reach consensus on CP blocks using some consensus protocol in order to reason about the state, which we explain next. A visual representation is shown in Figure 3.3.

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

Consensus protocol

The consensus protocol can be seen as a technique of running infinitely many times of a Byzantine consensus algorithm[3], starting a new execution immediately after

---

[2]This definition is different from the original TrustChain definition found in [36]. In the original definition, 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.

[3]More accurately it is an asynchronous common subset (ACS) algorithm, which we describe in Section 3.4.1.
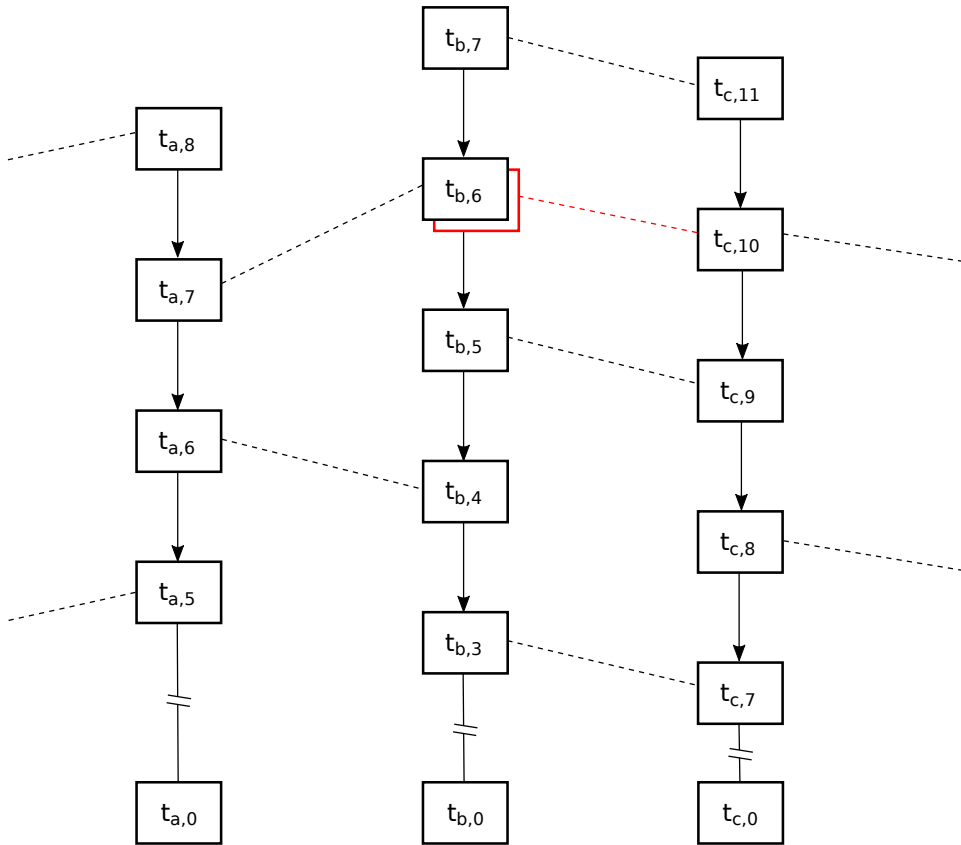
Figure 3.2: A TrustChain example with three nodes. Every block is denoted by $t_{i,j}$, where i is the node ID and j is the sequence number of the block. The arrows represent hash pointers and the dotted lines represent references. The blocks at the ends of the dotted lines are pairs of each other. The red block behind $t_{b,6}$ indicates a fork. Blocks of index 0 (e.g. $t_{b,0}$) are genesis blocks.

Figure 3.3: An example of Extended TrustChain. The circles represent CP blocks, they also have hash pointers (arrow) but do not have references (dotted line). Note that the sequence number counter does not change, it is shared with TX blocks. CP blocks of index 0 (e.g. $c_{b,0}$) are genesis blocks.

the previous one is completed. Nodes create new CP blocks at the end of every execution. This approach is necessary because blockchain systems always need to reach consensus on new values proposed by the nodes in the system, or CP blocks in our case.

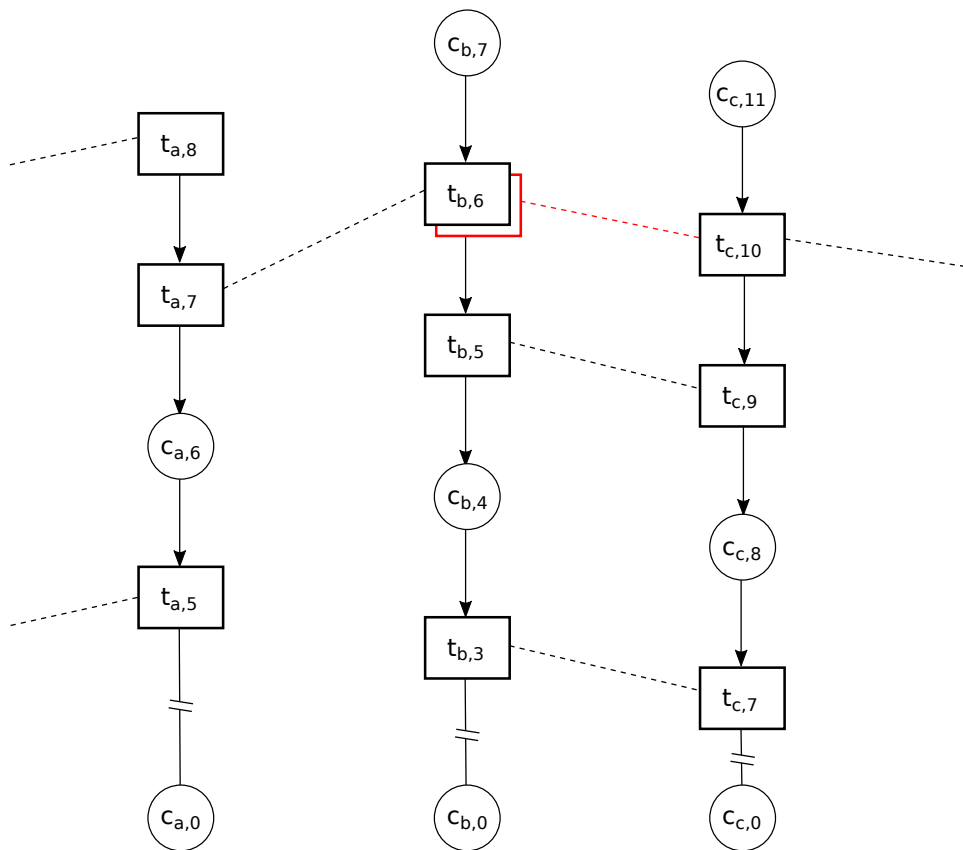The communication complexity of Byzantine consensus algorithms typically grows polynomially w.r.t the number of nodes, which prohibits us from running it on a large network. For example, one of the most recent works tested with up to 104 nodes and the algorithm took almost 6 minutes to complete [32]. Thus, at the beginning of every Byzantine consensus algorithm execution, we randomly elect a set of nodes—called facilitators—to collect CP blocks from every other node and use those blocks as the input to the Byzantine consensus algorithm, and then run it. After the algorithm completes, the facilitators output a set of CP blocks which we call the consensus result, which is then propagated to the network. Using the result, nodes are allowed to create new CP blocks, and then the next algorithm execution begins.

Transaction and validation protocols

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 malicious behaviour such as tampering the TrustChain. Thus, we need a validation protocol to counteract such behaviour. When a node wishes to validate one of its transactions, it asks the counterparty for the *agreed fragment* of the transaction. Which is a fragment of the counterparty's chain beginning and ending with CP blocks but contains the TX block belonging to that transaction, and the CP blocks must be in consensus. Upon the counterparty's response, the node checks whether the CP blocks are in some consensus result and some other conditions. The transaction is valid if these conditions are satisfied. Intuitively, this works because it is hard (because a cryptographically secure hash function is collision resistant) to create a different chain that begins and ends with the same two CP blocks.

The complete system

The complete system is essentially the concurrent composition of the three protocols above; all making use the Extended TrustChain data structure.

Our sub-protocol design gives us the highly desirable 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 state 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 explanation of CHECO. For notational clarity, we use the following convention (adapted from [32]) 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 the concatenation 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 full control of the delivery schedule and the ordering of all messages.

Further, we assume there exists 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 $\mathsf{H}(\cdot)$ is negligible with respect to λ [43].

In our model, we consider N nodes, which is the population size. n of them are facilitators, t nodes are malicious and the inequality $n \geq 3t + 1$ must hold. This inequality is from the work of Pease, Shostak and Lamport, where they show a network of n nodes cannot reach Byzantine agreement for $t \geq n/3$ [33]. Further, the inequality $N \geq n + t$ must also hold so that there are enough honest nodes in the network to run the consensus protocol; the exact reason becomes apparent 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 change 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 restriction is realistic because otherwise the adversary can eventually learn all the private keys and sabotage the system.

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 use $b_{u,h-1}$ 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 $T_u \cup C_u = B_u$ and $T_u \cap C_u = \varnothing$. The notation $b_{u,i}$ is generic over the block type.

Definition 1. Transaction block

The TX block is a six-tuple, i.e.

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

We describe each item in turn.

1. *$H(b_{u,i-1})$ 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 concatenation 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 TX block

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

there exist one and only one pair

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

if they are created by honest nodes running the TX protocol. 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 and genesis block

The CP block is a five-tuple, i.e.

$$c_{u,i} = \langle H(b_{u,i-1}), seq_u, H(\mathcal{C}_r), r, sig_u \rangle,$$

where $\mathcal{C}_r$ is the consensus result (which we describe next 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

$$c_{u,0} = \langle H(\bot), 0, H(\bot), 0, sig_u \rangle,$$

where $H(\bot)$ 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, where the first round is defined to be 1 and it is incremented after every execution of the consensus protocol. The consensus result, output of the consensus protocol, 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.

Now we define an important property 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

- *$c_{u,a}$ is the closest CP block to $t_{u,i}$ with a lower sequence number and*

- *$c_{u,b}$ is the closest CP block to $t_{u,i}$ with a higher sequence number,*

then $\langle c_{u,a}, c_{u,b} \rangle$ is the enclosure of $t_{u,i}$. Some TX blocks may not have any enclosure, then their enclosure is $\bot$. Agreed enclosure is the same as enclosure with an extra constraint where the CP blocks must be in some consensus result $\mathcal{C}_r$.

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 agreed_fragment$(t_{u,i})$ outputs the agreed fragment of $t_{u,i}$ if it exists, otherwise $\bot$.

3.4 Consensus protocol

Our consensus protocol Π_c runs on top of the Extended TrustChain. It uses an unmodified asynchronous common subset (ACS) algorithm as the key building block. The objectives of the protocol are to allow honest nodes to make progress (in the form of creating new CP blocks), compute correct the consensus result in every round and have an unbiased election of facilitators. We formally define the desired properties below.

Definition 6. TrustChain consensus protocol

A TrustChain consensus protocol is correct if the following holds for every round r.

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) $|\mathcal{C}_r| \geq N - t$[4]*, and*

    (b) $|\mathcal{F}_r| = n$.

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 are similar to the typical Byzantine consensus properties, 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 a failure in one round will lead to failures in subsequent rounds.

Before describing the protocol in detail, we take a brief detour to give background on ACS 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 common subset

The best way to explain ACS is to contrast it with the typical Byzantine consensus. We adapt the description from [44, 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 following.

1. Agreement: *If a correct node outputs v, then every correct node outputs v.*

---

[4] $\mathcal{C}_r$ 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 $\mathcal{C}_r$.

2. Validity: *The decision value must be the input value of a correct node.*

3. Termination: *All correct nodes terminate in finite time.*

A node under Byzantine fault means that it can have arbitrary behaviour, for example, not sending a message or colluding with other Byzantine nodes to undermine the entire system. Note that the decision is on a single value, which is in contrast to ACS that 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 [32]).

Definition 8. Asynchronous common subset

There are n nodes, of which at most t might experience Byzantine fault. Node i starts with a 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 correct 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. Termination: *If $n - t$ nodes receive an input, then all correct nodes produce an output.*

ACS has the nice property of censorship resilience when compared to other consensus algorithms. In PBFT [31], a leader is elected, if the leader is malicious but follows the protocol, then it can selectively filter values. In contrast, every party in ACS are involved in the proposal phase, and it guarantees that if $n - 2t$ parties propose the same value, then it must be in the agreed output. In other words, 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 [32]. 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.

To bootstrap, imagine that there is some bootstrap oracle that initiates the correct program on every node, meaning that it satisfied the properties in Definition 6. In practice, the bootstrap oracle is most likely the software developer that sets up the system and assigns the facilitators of round 1 in a fair way. This concludes the bootstrap phase. For any future rounds, the consensus phase is used.

3.4.3 Consensus phase

For any node u, the consensus phase begins when $\mathcal{F}_r$ is available and the latest block is $c_{u,h-1}$. Note that $\mathcal{F}_r$ indicates the facilitators that were elected using results of round r and are responsible for driving the ACS algorithm 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 u is not the facilitator, it sends $\langle \texttt{cp\_msg}, c_{u,h-1} \rangle$ to all the facilitators. Second, if u is a facilitator, it waits until $N - t$ messages of type $\texttt{cp\_msg}$ are received. Invalid messages are removed. That is blocks with invalid signatures and blocks signed by the same key. With the sufficient number of $\texttt{cp\_msg}$ 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 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.

The core of the consensus phase is the ACS algorithm. While any algorithm that solves the ACS problem will work, we use a simplification of HoneyBadgerBFT [32] as our ACS algorithm because it is a consensus algorithm 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.

Continuing, when $\mathcal{F}_r$ finish the ACS execution and reach agreement on $\mathcal{C}_{r+1}$, they immediately broadcast two messages to all the nodes—first the consensus message $\langle \texttt{cons\_msg}, \mathcal{C}_{r+1} \rangle$, and second the signature message $\langle \texttt{cons\_sig}, r, sig \rangle$. The reason for sending $\texttt{cons\_sig}$ is the following. The 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 $n - t$ valid signatures, u performs two tasks. First, it creates a new CP block using $\textsf{new\_cp}(\mathcal{C}_{r+1})$ (Algorithm 1). Second, it computes the new facilitators using $\textsf{get\_facilitator}(\mathcal{C}_{r+1}, n)$ (Algorithm 2), and

updates its facilitator set to the result. This concludes the consensus phase and brings us back to the state at the beginning of the consensus phase, so a new round can be started.

Our protocol has some similarities with synchronizers [44, 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 synchronizer 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.

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

Algorithm 1 Function $\mathsf{new\_cp}(\mathcal{C}_r)$ runs in the context of the caller u. It creates a new CP block and appends it to u's chain.

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

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.

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

3.5 Transaction protocol

The TX protocol Π_t, shown in Algorithm 4, is run by all nodes. It is also known as True Halves, first described informally by Veldhuisen [45, Chapter 3.2]. Nodes that wish to initiate a transaction calls $\mathsf{new\_tx}(pk_v, m, txid)$ (Algorithm 3) with the intended counterparty v identified by pk_v and message m. $txid$ should be a uniformly distributed random value, i.e. $txid \in_R \{0,1\}^{256}$. Then the initiator sends $\langle \texttt{tx\_req}, t_{u,h} \rangle$ to v.

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

Algorithm 3 Function 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.

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

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

Upon $\langle \texttt{tx\_req}, t_{v,j} \rangle$ from v
  $\langle \_, \_, txid, pk_v, m, \_ \rangle \leftarrow t_{v,j}$
  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 $txid$

3.6 Validation protocol

The consensus protocol and the transaction protocol do not provide a mechanism to detect tampering. The validation protocol aims to solve this issue. The protocol is also a request-response protocol. It assumes that nodes store all the consensus results as well as the corresponding signatures that they are aware of. Further, we assume nodes that recovered from a crash or a corruption are able to recover its missed cons_msg and cons_sig messages, which can be achieved by querying other nodes. 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—*valid*, *invalid* and *unknown*. Given a fragment $F_{v,j}$, the validity of the TX block $t_{u,i}$ with its corresponding fragment $F_{u,i}$ is captured by the function get_validity($t_{u,i}, F_{u,i}, F_{v,j}$) (Algorithm 5). Note that $t_{u,i}$ and $F_{u,i}$ are assumed to be valid, otherwise the node calling the function would have no point of reference. This assumption is reasonable in practice because typically the caller is u, so it knows its own TX block and the corresponding agreed fragment. If the caller is not u, it can always query for the agreed fragment that contains the transaction of interest from u.

Algorithm 5 works as follows. Before Line 3, we 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*. Such case is possible because the result of agreed_fragment($\cdot$) would be $\perp$ for new transactions. The next three 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 omitted TX blocks. But in reality, it was the malicious sender who did not follow the protocol. Thus, in such cases, the whole transaction is marked as invalid.

Algorithm 5 Function 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 correct and contains $t_{u,i}$. $F_{v,j}$ is the corresponding fragment received from v.

1: **if** $F_{v,j}$ is not a fragment created in the same round as $F_{u,i}$ **then**
2:     **return** *unknown*
3:
4: $\langle \_, \_, txid, pk_v, m, \_\rangle \leftarrow t_{u,i}$
5: **if** number of blocks of $txid$ in $F_{v,j} \neq 1$ **then**
6:     **return** *invalid*
7:
8: $t_{v,j} \leftarrow$ the TX block with $txid$ in $F_{v,j}$
9: $\langle \_, \_, txid', pk_u, m', \_\rangle \leftarrow t_{v,j}$
10: **if** $m \neq m' \lor txid \neq txid'$ **then**
11:     **return** *invalid*
12:
13: **if** $t_{u,i}$ is not signed by $pk_u \lor t_{v,j}$ is not signed by pk_v **then**
14:     **return** *invalid*
15: **return** *valid*

3.6.2 Validation protocol

Our validation protocol Π_v, shown in Algorithm 6, is designed to classify transactions according to the aforementioned validity definition. If u wishes to validate some TX with ID $txid$ and counterparty v, it sends $\langle$vd_req$, txid\rangle$ to v. The desired properties of the validation protocol are as follows.

Definition 9. TrustChain validation protocol
A TrustChain validation protocol is correct if the following properties hold.

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

same conclusion or output unknown.

2. Validity*: The validation protocol outputs the correct result according to the validity definition above.*

3. Liveness*: Any valid (invalid) transaction is marked as valid (invalid) eventually.*

Algorithm 6 Validation protocol Π_v 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 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 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, 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, 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. Furthermore, realistic networks do not offer a broadcast primitive, i.e. firewalls may block incoming messages from certain nodes.

An improvement is to use an epidemic protocol [46] instead, also known as gossip protocol. Typical epidemic protocols work 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 an epidemic protocol is that the communication cost is distributed more evenly between nodes. It 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 [47]. With this, nodes typically only need to receive one consensus result message instead of n.

A down side of an epidemic protocol is that it usually takes $O(\log N)$ time to infect the whole network, whereas broadcasting is constant time. Another downside of some epidemic protocols (e.g., eager push) is that it is difficult to guarantee delivery. It is particularly the case when the parameters are not chosen 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. However, in many applications, 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) [48]. A 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 $N-t$ `cp_msg` messages, which is the only place where we used N as a parameter. To introduce timing, instead of waiting for $N-t$ 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 many 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
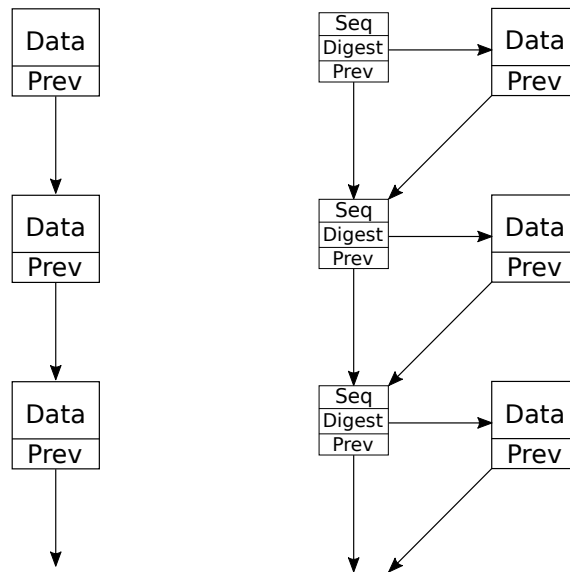
Figure 3.4: The chain on the left represents direct chaining of full blocks, where the digest in "Prev" is simply the digest of the previous full block. The chain on the right uses compact blocks, represented by the smaller squares. Compact blocks also form a chain as before, but they each have a hash pointer to the full block, identified by "Seq" and "Digest".

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 but still wish to leave. In this case, the node must fulfil 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 block-chain 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 of the corresponding full block,

2. Digest—the digest of the corresponding full 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 agreed fragment containing the full blocks. With compact blocks, it is only necessary to send the compact version of the agreed fragment. The validation then proceeds similarly, provided that the pair of the to-be-validated TX block is known.

The space saving of this approach 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 [49]), 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 transactions. Concretely, for node A, upon receiving an agreed fragment from node B, rather than validating a single transaction, A attempts to validate all transactions performed with B, which are in the unknown state but also in that fragment.

The benefit of this technique is maximised when a node 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   Total fork detection

The validation algorithm guarantees that there are no forks within a single agreed fragment, which is sufficient for some applications such as proving the existence 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.

We offer two approaches to do total fork detection. First and the easiest solution is to ask for the complete hash chain of the counterparty. The verifier can be sure that there are no forks if the following conditions hold.

1. The hash pointers are correct.

2. All the CP blocks are in consensus.

3. The TX of interest is in the chain.

We use this approach in our prior work on Implicit Consensus [50]. Nodes employ caching to minimise communication costs, and we call this effect spontaneous sharding.

The second approach is probabilistic but with only a constant communication overhead over our current design. For a node, observe that if all of its agreed fragments has a transaction with an honest node, then the complete chain is effectively validated in a distributed manner. The only way for an attacker to make a fork is to ensure that the agreed fragment containing the fork has no transactions with honest nodes. Such malicious behaviour is prevented probabilistically 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 prove that it holds a valid agreed fragment between some consensus round specified by A. If B provides a correct and timely response, then they run the transaction protocol as usual. Otherwise, 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 (denoted by the random variable X) follows a binomial distribution, i.e.

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

Chapter 4

Analysis of correctness and performance

In this chapter, we evaluate our system analytically to check whether it correctly implements the TrustChain consensus protocol (Definition 6) and the TrustChain validation protocol (Definition 9). Further, we analyse the performance and in particular the throughput to understand the scalability characteristics. 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 analysis

Our first objective is to show that our consensus protocol Π_c described in Section 3.4 is, in fact, a TrustChain consensus protocol defined in Definition 6. Then, building on top of it, we show that our validation protocol Π_v described in Section 3.6 conforms to the TrustChain validation protocol defined in Definition 9. 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 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. *If for an arbitrary round r, $\mathcal{F}_r$ 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 of $\mathcal{F}_r$. For each facilitator, 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). No honest facilitator can 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 receive $N - t$ valid CP blocks. At the end of ACS, $\mathcal{C}_{r+1}$ is created and then 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 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. $\square$

Lemma 2. *If for an arbitrary round r, $\mathcal{F}_r$ is known by all correct nodes and any correct node outputs $\mathcal{F}_{r+1}$, then (a) $|\mathcal{C}_{r+1}| \geq N - t$ and (b) $|\mathcal{F}_{r+1}| = n$.*

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 $|\mathcal{C}_{r+1}| \geq N - t$ and this proves (a). Since $N - t \geq (n + t) - t = n$ and get_facilitators$(\mathcal{C}_{r+1}, n)$ outputs n items, so $|\mathcal{F}_{r+1}| = n$ and this proves (b). $\square$

Lemma 3. *If for an arbitrary round r, $\mathcal{F}_r$ 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. As a result, the output of get_facilitators$(\mathcal{C}_{r+1}, n)$ is a random permutation of all the nodes in $\mathcal{C}_{r+1}$ truncated to n. Thus, every node has the same probability of becoming a facilitator. $\square$

Lemma 4. *If for an arbitrary round r, $\mathcal{F}_r$ 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 disseminated to all the nodes. $\square$

From Lemmas 1, 2, 3 and 4, we have shown that the four properties of Definition 6 holds when assuming the existence of some $\mathcal{F}_r$. Now we need to proof these properties under the universal quantifier on r. We do this using mathematical induction.

Theorem 1. Π_c *implements a TrustChain consensus protocol (Definition 6).*

Proof. We prove 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 represents the set of facilitators agreed in round 1, are responsible for driving the ACS protocol in round 2.

For the inductive step, we assume that the four 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. $\qquad\square$

4.1.2 Correctness of the validation protocol

Π_c ensures consensus on CP blocks and facilitators, which makes it the building block for reaching consensus on transactions. In this section, we build on top of Theorem 1 to show that our validation protocol Π_v described in Section 3.6 has the agreement and validity properties of a TrustChain validation protocol defined in Definition 9.

Theorem 2. Π_v *satisfies the agreement and validity properties of a TrustChain validation protocol (Definition 9).*

Proof. We proof the agreement property by contradiction. Recall that $F_{u,i}$ and $t_{u,i}$ used in Algorithm 5 are valid. Without loss of generality, suppose for some transaction with TX block t, node u decides *valid* but node v decides *invalid*. Then there exists a fragment $F' = \{\ldots, t', c'\}$ which u received that contains a valid pair of t—t'. There also exists a fragment $F'' = \{\ldots, t'', c''\}$ which v received that does not contain or contains an invalid pair—t''. In both cases, the 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.

Π_v directly uses Algorithm 5, which is also the validity definition, so the validity property is also satisfied. $\qquad\square$

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 [50], 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 transaction throughput 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 of Theorem 2 is

unforgeability. That is, no adversary can create two chains, $F = \{\ldots, t, c\}$ and $F' = \{\ldots, t', c\}$, with correct hash pointers and the same end of chain c with non-negligible probability.

A stronger version of the validity definition exists. That is, if two honest nodes make a transaction, then the transaction state 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 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 problem. Now we show a negative result, where the liveness property of Definition 9 cannot be attained in Π_v, meaning that transactions made with adversaries cannot always be validated.

Lemma 5. *There exists a valid transaction that cannot be eventually validated.*

Proof. Suppose nodes u and v correctly performed the TX protocol to create 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 from u, and then the transaction can never be validated. $\square$

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 needs to comply with the protocol.

4.2 Performance analysis

This section aims to answer whether our system is horizontally scalable analytically. We begin by looking at the communication and time complexity of the consensus protocol Π_c, 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

Π_c can be 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 $O(n^2|v| + \lambda n^3 \log n)$ [32], where $|v|$ is the size of the 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 is $O(N)$ for a constant n.

4.2.2 Duration of the consensus protocol

To make arguments on the duration of Π_c, the bandwidth or throughput in our purely asynchronous model, which are concepts that depend on time, we must make additional assumptions. Note that the duration we are interested in is not the same as the time complexity typically used in distributed systems. In the analysis of distributed algorithms, time complexity is often in terms of the number of rounds. For example, ACS runs in a constant number of rounds because of its sub-protocols—reliable broadcast and binary Byzantine consensus—also run in a constant number of rounds [32]. 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 our results in Section 4.2.1 and the fact that Π_c runs in a constant number of rounds, it follows that Π_c has a duration of $O(N)$.

4.2.3 Communication cost for transactions

Geared with the results above, 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 result 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 of 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 result 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). This behaviour is verified experimentally in Chapter 5.

4.2.4 Linear global throughput

Using our results so far, we can 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" [51]. Now, suppose every node has N links which connect to every other node, every link has a fix bandwidth C and every node makes r_{tx} transactions per second. Then we have the inequality

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

where l is the length of the agreed 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, which 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. Hence, 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. $NC = r_{tx}l$. Recall that l is of $O(N)$, which means LHS and RHS both grow linearly with respect to N. Hence, there exists some r_{tx} that makes use of all the available bandwidth regardless of N. Finally, if every node in the network is transacting at r_{tx} per second, then the global throughput (in terms of transactions per second) is linear w.r.t. the population size N. We also maintain a constant transaction rate by the same argument if the system is not making use of all the bandwidth. Both claims are verified experimentally in Chapter 5.

4.3 Effect of a highly adversarial environment

Our last study considers the effect when the number of adversaries is more than t. This is useful because in practice it is difficult to guarantee that t satisfies $n \geq 3t + 1$, 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 [52]. 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.
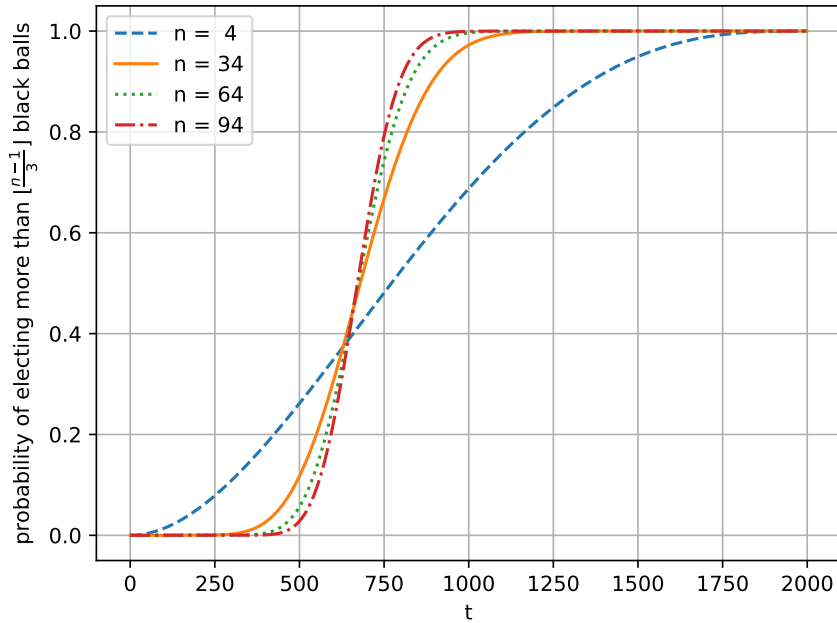
Figure 4.1: Probability of selecting more than $\lfloor \frac{n-1}{3} \rfloor$ black balls for different value of t with N fixed at 2000.

On the other hand, due to the fact that hypergeometric distributions have light tails with "faster-than-exponential fall-off" [52], 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

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

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 picking parameters for a desired level of fault tolerance. If n is fixed, 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 $n = 100$ and $\alpha = 1/10$, i.e. 10% of the nodes are malicious, then the probability to draw more black balls than the threshold is 9.9×10^{-6}.

Chapter 5

Implementation and experimental results

We evaluate our system experimentally to understand its scalability characteristics as well as its limitations. Concretely, we give a free and open-source prototype implementation that runs on a network with over 1000 node. Nodes in our implementation make transactions autonomously with each other, and then we collect the results for analysis.

This chapter begins with a description of the implementation in Section 5.1. Then, we describe 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 Section 4.2, where we analyse the performance experimentally.

5.1 Implementation

The prototype implementation can be found on GitHub.

https://github.com/kc1212/checo

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. ACS uses erasure code in one of its sub-protocols (reliable broadcast). Thus we

---

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

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, which is hard-coded in the source file[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 new_tx($\cdot$), new_cp($\cdot$) and agreed_fragment($\cdot$). Finally, the `node` module ties everything together. It implements the consensus protocol, the transaction protocol and the validation protocol.

Every node keeps a persistent TCP connection with every other node, which 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 goal of the experiment is to run CHECO and analyse the communication cost, the consensus duration and the global throughput. We investigate these properties under the following parameters.

1. 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 in Section 5.6 give a good indication of how our system may function for a larger n.

2. The population size $N \in \{200, 300, \ldots, 1200\}$. The reason why N only goes up to 1200 is due to our physical setup, which we describe below.

3. The two modes of transaction. The first mode is the *fixed-neighbour* mode where nodes only transact with one predetermined node. It 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 performed with a random node out of the N nodes in the system. It is unlikely that the agreed fragments can be reused.

---

[3] https://github.com/openstack/liberasurecode
[4] https://github.com/openstack/liberasurecode/blob/0794b31c623e4cede76d66be730719d24debcca9/include/erasurecode/erasurecode.h#L35
[5] https://pypi.python.org/pypi/libnacl

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 machines, for each machine we use up to 40 nodes, which 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 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 machines, 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, the transaction rate r_{tx} is fixed at 2 TX/s for every node. This value, as we show later, creates a bottleneck in extreme cases which helps us understand the limitations of our design. Since Bitcoin transactions are approximately 500 bytes [49], 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 shows 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, which 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 is similar. Both peak at about 100 MB, which is expected because the consensus protocol is decoupled from the transaction protocol and the validation protocol. Finally, the rate for which the communication cost increases is higher when the number of facilitators is higher, which is also expected because the ACS algorithm has a polynomial term w.r.t. n 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 results 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
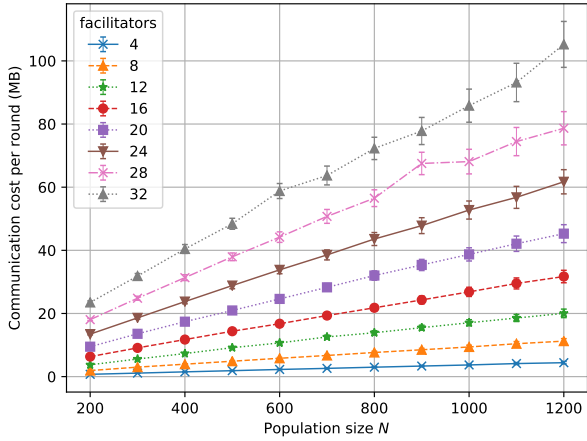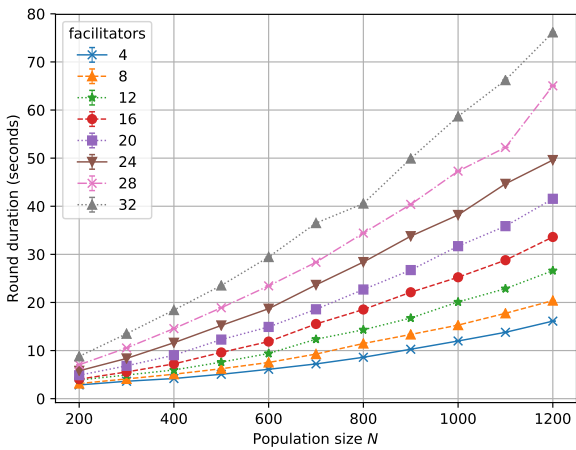

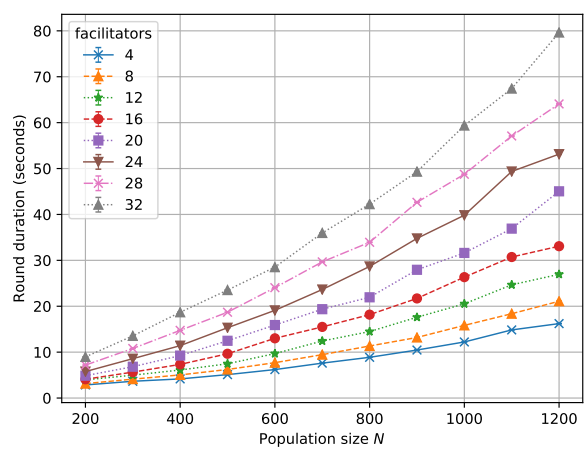(a) Transactions are with fixed neighbours.
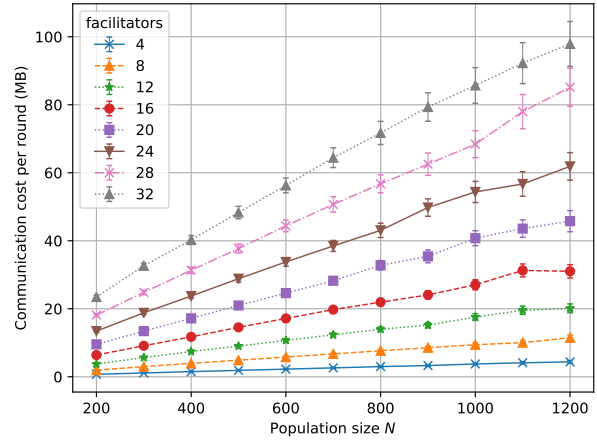
(b) Transactions are with random neighbours.

Figure 5.1: Experimental result of the communication cost per round for the consensus protocol. 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.
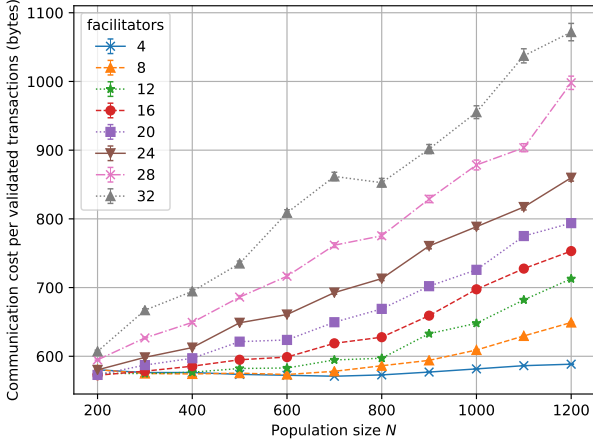

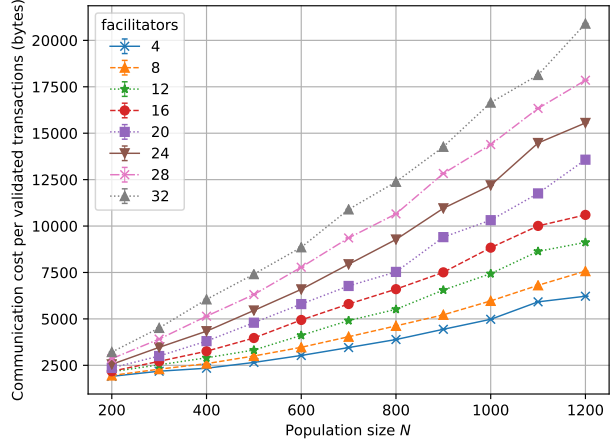(a) Transactions are with fixed neighbours.

(b) Transactions are with random neighbours.

Figure 5.2: Experimental result of the round duration per round for the consensus protocol.

(a) Transactions are with fixed neighbours.

(b) Transactions are with random neighbours.

Figure 5.3: Experimental result of the communication cost per verified transaction. It has a similar, nearly linear, trend as Figure 5.2. Fluctuation for the fixed-neighbour mode exists because the cache mechanism is unpredictable.

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 size in Figure 5.3. We observe a nearly linear relationship, which is due to the nearly 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 to validate all transactions of that round. 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, which is due to our caching mechanism. We send validation requests at the same rate as transactions. Upon receiving a

(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 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 a new 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 effect 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 (4 TX/s)? This result 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.

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 the fact that the network infrastructure cannot keep up with our demand. In practice, we do not expect such behaviour to occur as it is possible to cache agreed fragments.

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 Fig-
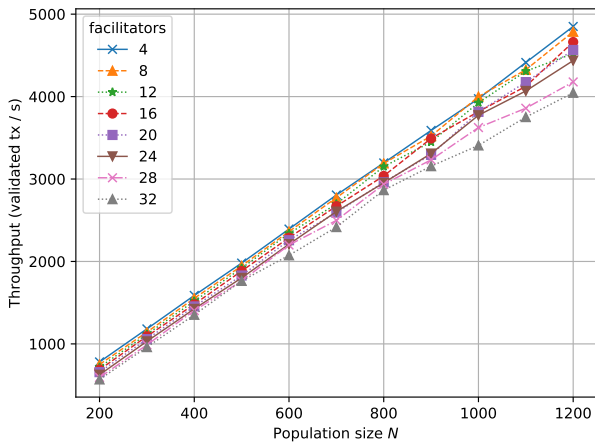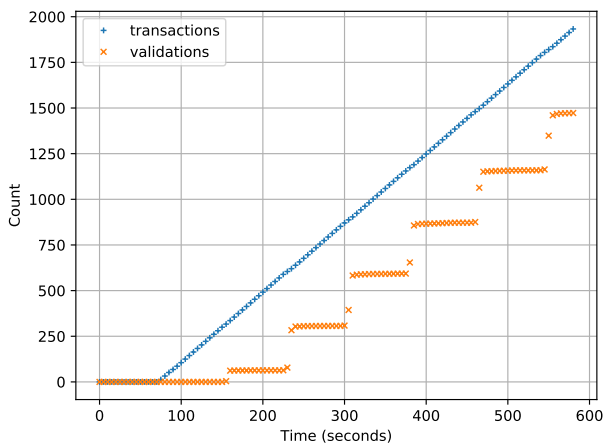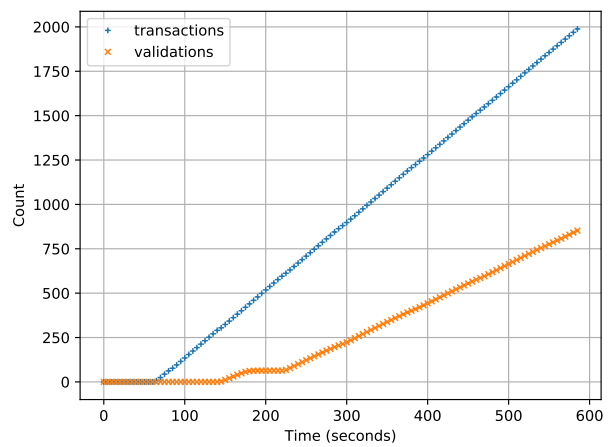
(a) Transactions are with fixed neighbours.
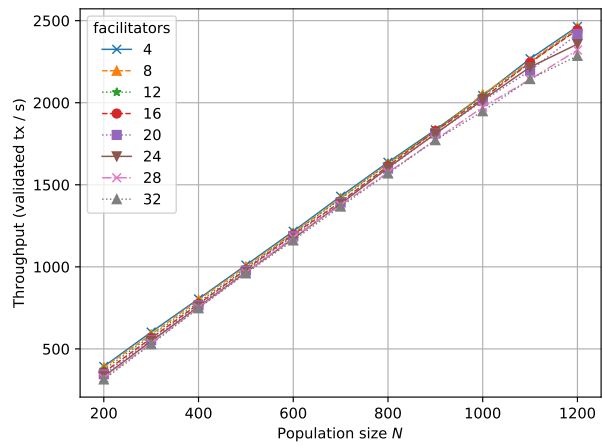
(b) Transactions are with random neighbours.

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 through-put because of the caching mechanism.

(a) Transactions are with fixed neighbours.

(b) Transactions are with random neighbours.

Figure 5.5: Demonstration of the network bottleneck. Note that the slope of vali-dated transactions grows at a slower rate in Figure 5.5b than in Figure 5.5a.

ure 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.

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 into our system performance may perform when the number of facilitators is higher than 32.

Figure 5.6 shows the communication cost of the consensus protocol for varying numbers of facilitators. There is substantial evidence that the communication cost grows polynomially. We expect this behaviour because of there are polynomial terms in the ACS communication complexity—$O(n^2|v|+\lambda n^3 \log n)$, which 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.

The same experiment viewed from the perspective of time is shown in Figure 5.7. Similarly, the duration increases polynomially for a linear increase in fault tolerance.
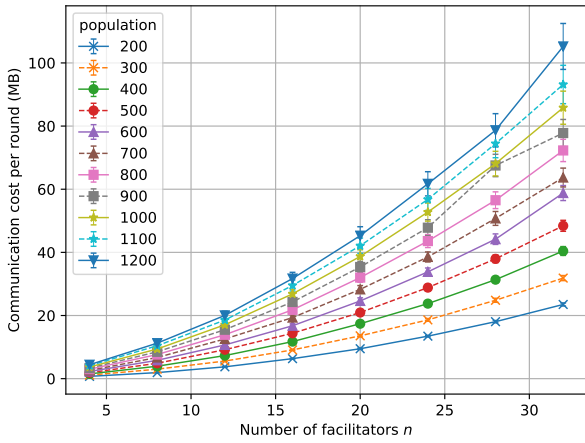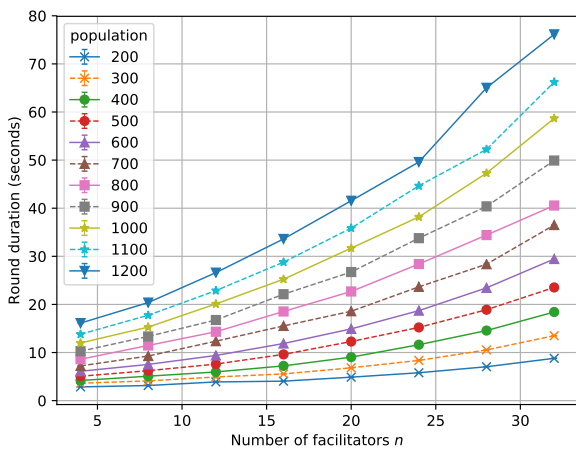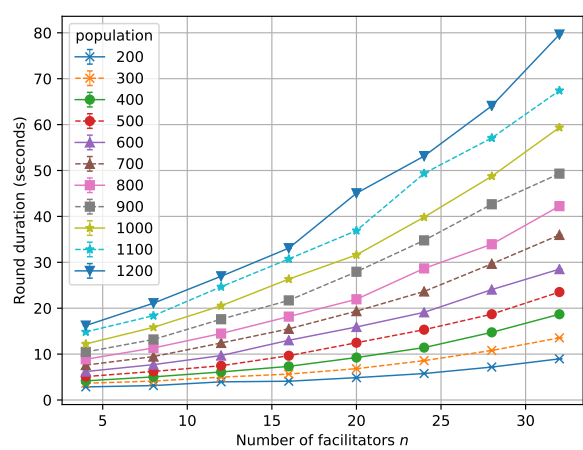
(a) Transactions are with fixed neighbours.
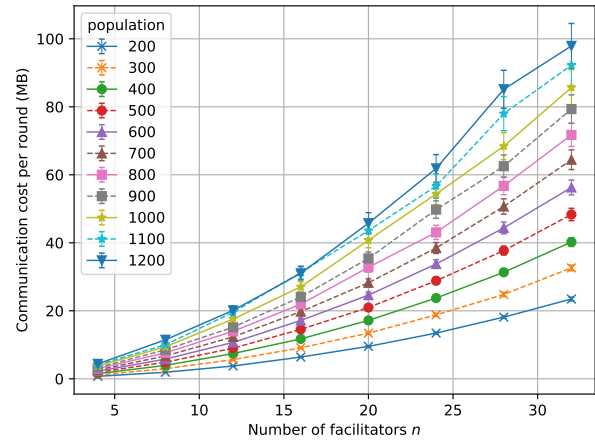
(b) Transactions are with random neighbours.

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

(a) Transactions are with fixed neighbours.

(b) Transactions are with random neighbours.

Figure 5.7: The consensus duration increases polynomially with respect to the number of facilitators.

Chapter 6

Conclusion

In this work, we introduced an application neutral blockchain system which we call CHECO. We add checkpoint block to the existing TrustChain data structure to capture the system state. The round based consensus protocol uses ACS as a building block to reach consensus on checkpoint blocks. The consensus result, which is a set of checkpoint blocks, lets nodes elect new facilitators and create new checkpoint blocks. The existing work on True Halves is formalised to act as our transaction protocol. Finally, we introduce a validation protocol which ensures 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.

This work answers the research question stated in Section 2.2.

How can we design a blockchain consensus protocol that is fault-tolerant, horizontally scalable, and able to reach global consensus?

Concretely, 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. The 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 conducted in a somewhat idealised world (DAS-5). We hope to study our prototype in a more realistic environment. For example, DeterLab [53] lets users simulate network delay, which is essential for discovering issues that may only occur in hostile environments. To understand our system further, we hope to experiment with a real platform with real users such as Tribler [38]. For instance, the standard TrustChain in Tribler is used for accounting internet traffic 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 adequate for permissioned systems where nodes have relatively more trust between each other. However, it is not adequate for permissionless systems, which usually aim to tolerate fault that involves 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 (e.g., Ouroboros [54]) which means that nodes with more stake in the system get more vote. In an ideal PoS scheme, the system can tolerate any fault if the majority of the stakeholders are honest.

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

Bibliography

[1] M. N. Baily and D. J. Elliott. (Jun. 2009). The us financial and economic crisis: Where does it stand and where do we go from here?, [Online]. Available: 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).

[2] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

[3] K. J. O'Dwyer and D. Malone, 'Bitcoin mining and its energy footprint', 2014.

[4] CoinMarketCap. (Jun. 2017). Cryptocurrency market capitalizations, [Online]. Available: https://coinmarketcap.com/currencies/bitcoin/ (visited on 05/08/2017).

[5] A. Tapscott and D. Tapscott, *How blockchain is changing finance*. Harvard Business Review, Mar. 2017. [Online]. Available: https://hbr.org/2017/03/how-blockchain-is-changing-finance (visited on 04/07/2017).

[6] R. Das, *Does blockchain have a place in healthcare?* Forbes, May 2017. [Online]. Available: https://www.forbes.com/sites/reenitadas/2017/05/08/does-blockchain-have-a-place-in-healthcare/ (visited on 04/07/2017).

[7] G. Prisco, *Walmart testing blockchain technology for supply chain management*. Bitcoin Magazine, Dec. 2016. [Online]. Available: https://bitcoinmagazine.com/articles/walmart-testing-blockchain-technology-for-supply-chain-management/ (visited on 04/07/2017).

[8] J. Basden and M. Cottrell, *How utilities are using blockchain to modernize the grid*. Harvard Business Review, Mar. 2017. [Online]. Available: https://hbr.org/2017/03/how-utilities-are-using-blockchain-to-modernize-the-grid (visited on 04/07/2017).

[9] M. Vukolić, 'The quest for scalable blockchain fabric: Proof-of-work vs. bft replication', in *International Workshop on Open Problems in Network Security*, Springer, 2015, pp. 112–125.

[10] A. Quentson, *While bitcoin price hits record highs, nearly 100,000 transactions are stuck in a backlog.* CryptoCoinsNews, May 2017. [Online]. Available: https://www.cryptocoinsnews.com/almost-100000-bitcoin-transactions-stuck-in-a-backlog-waiting-to-move/ (visited on 25/06/2017).

[11] L. Shin, *Bitcoin is mired in a civil war. can this proposal save it?* Forbes, Apr. 2017. [Online]. Available: 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).

[12] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, 'On scaling decentralized blockchains', in *International Conference on Financial Cryptography and Data Security*, Springer, 2016, pp. 106–125.

[13] (2015). Visa inc. at a glance, [Online]. Available: https://usa.visa.com/dam/VCOM/download/corporate/media/visa-fact-sheet-Jun2015.pdf (visited on 12/06/2017).

[14] E. Lombrozo, J. Lau and P. Wuille. (Dec. 2015). Segregated witness (consensus layer), [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki (visited on 25/06/2017).

[15] Bitcoin Wiki. (Aug. 2015). Transaction malleability, [Online]. Available: https://en.bitcoin.it/wiki/Transaction_Malleability (visited on 25/06/2017).

[16] J. Poon and T. Dryja, 'The bitcoin lightning network', Jan. 2016. [Online]. Available: https://lightning.network/lightning-network-paper.pdf.

[17] BitcoinCash.org. (2017). Bitcoin cash, [Online]. Available: https://www.bitcoincash.org/ (visited on 04/08/2017).

[18] A. van Wirdum, *The birth of bch: The first crazy days of "bitcoin cash"*. Bitcoin Magazine, Aug. 2017. [Online]. Available: https://bitcoinmagazine.com/articles/birth-bch-first-crazy-days-bitcoin-cash/ (visited on 04/08/2017).

[19] T. Swanson. (May 2014). Bitcoins: Made in china, [Online]. Available: http://www.ofnumbers.com/wp-content/uploads/2014/05/Bitcoins-Made-in-China.pdf (visited on 18/08/2017).

[20] J. Tuwiner. (Jul. 2017). Bitcoin mining pools, [Online]. Available: https://www.buybitcoinworldwide.com/mining/pools/ (visited on 18/08/2017).

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

[22] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert and P. Saxena, '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.

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

[24] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser and B. Ford, '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.

[25] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly and B. Ford, 'Omniledger: A secure, scale-out, decentralized ledger.', *IACR Cryptology ePrint Archive*, vol. 2017, p. 406, 2017.

[26] I. Abraham, D. Malkhi, K. Nayak, L. Ren and A. Spiegelman, 'Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus', *ArXiv preprint arXiv:1612.02916*, 2016.

[27] I. Eyal and E. G. Sirer, 'Majority is not enough: Bitcoin mining is vulnerable', in *International Conference on Financial Cryptography and Data Security*, Springer, 2014, pp. 436–454.

[28] C. Decker and R. Wattenhofer, 'A fast and scalable payment network with bitcoin duplex micropayment channels', in *Symposium on Self-Stabilizing Systems*, Springer, 2015, pp. 3–18.

[29] Bitcoin Wiki. (Jan. 2017). Multisignature, [Online]. Available: `https://en.bitcoin.it/wiki/Multisignature` (visited on 20/06/2017).

[30] ——, (Nov. 2016). Hashed timelock contracts, [Online]. Available: `https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts` (visited on 20/06/2017).

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

[32] A. Miller, Y. Xia, K. Croman, E. Shi and D. Song, 'The honey badger of bft protocols', in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2016, pp. 31–42.

[33] M. Pease, R. Shostak and L. Lamport, 'Reaching agreement in the presence of faults', *Journal of the ACM (JACM)*, vol. 27, no. 2, pp. 228–234, 1980.

[34] S. Popov, *The tangle*, Apr. 2016. [Online]. Available: `https://iota.org/IOTA_Whitepaper.pdf`.

[35] M. Hearn, *Corda: A distributed ledger*, Sep. 2016. [Online]. Available: `https://docs.corda.net/_static/corda-technical-whitepaper.pdf`.

[36] S. Norberhuis, *Multichain: A cybercurrency for cooperation*, Dec. 2015. [Online]. Available: http://resolver.tudelft.nl/uuid:59723e98-ae48-4fac-b258-2df99d11012c.

[37] B. Cohen, 'Incentives build robustness in bittorrent', in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.

[38] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. Epema, M. Reinders, M. R. Van Steen and H. J. Sips, 'Tribler: A social-based peer-to-peer system', *Concurrency and computation: Practice and experience*, vol. 20, no. 2, pp. 127–138, 2008.

[39] Namecoin Developers. (). Namecoin, [Online]. Available: https://www.namecoin.org/ (visited on 25/06/2017).

[40] Nebulous Inc. (). Your decentralized private cloud, [Online]. Available: https://sia.tech/ (visited on 25/06/2017).

[41] J. R. Douceur, 'The sybil attack', in *International Workshop on Peer-to-Peer Systems*, Springer, 2002, pp. 251–260.

[42] H. Yu, M. Kaminsky, P. B. Gibbons and A. Flaxman, 'Sybilguard: Defending against sybil attacks via social networks', in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 36, 2006, pp. 267–278.

[43] M. Bellare and P. 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.

[44] R. Wattenhofer, *Principles of distributed computing*, 2016. [Online]. Available: http://dcg.ethz.ch/lectures/podc_allstars/lecture/podc.pdf.

[45] P. Veldhuisen, 'Leveraging blockchains to establish cooperation', Master's thesis, Delft University of Technology, May 2017. [Online]. Available: http://resolver.tudelft.nl/uuid:0bd2fbdf-bdde-4c6f-8a96-c42077bb2d49.

[46] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec and L. Massoulié, 'Epidemic information dissemination in distributed systems', *Computer*, vol. 37, no. 5, pp. 60–67, 2004.

[47] J. Leitao, J. Pereira and L. Rodrigues, 'Epidemic broadcast trees', in *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, IEEE, 2007, pp. 301–310.

[48] Bitcoin Project. (). Bitcoin developer guide, [Online]. Available: https://bitcoin.org/en/developer-guide (visited on 04/07/2017).

[49] TradeBlock. (Oct. 2015). Analysis of bitcoin transaction size trends, [Online]. Available: https://tradeblock.com/blog/analysis-of-bitcoin-transaction-size-trends (visited on 14/07/2017).

[50] Z. Ren, K. Cong, J. Pouwelse and Z. Erkin, *Implicit consensus: Blockchain with unbounded throughput*, 2017. eprint: `arXiv:1705.11046`.

[51] R. Prasad, C. Dovrolis, M. Murray and K. Claffy, 'Bandwidth estimation: Metrics, measurement techniques, and tools', *IEEE network*, vol. 17, no. 6, pp. 27–35, 2003.

[52] M. Skala, 'Hypergeometric tail inequalities: ending the insanity', *ArXiv e-prints*, Nov. 2013. arXiv: `1311.5939 [math.PR]`.

[53] J. Mirkovic and T. Benzel, 'Teaching cybersecurity with deterlab', *IEEE Security & Privacy*, vol. 10, no. 1, pp. 73–76, 2012.

[54] A. Kiayias, A. Russell, B. David and R. Oliynykov, 'Ouroboros: A provably secure proof-of-stake blockchain protocol', in *Annual International Cryptology Conference*, Springer, 2017, pp. 357–388.

Appendix A

Consensus protocol example

Visualisations of the different stages in the consensus protocol is shown in Figures A.1 to A.4. The captions detail every step.
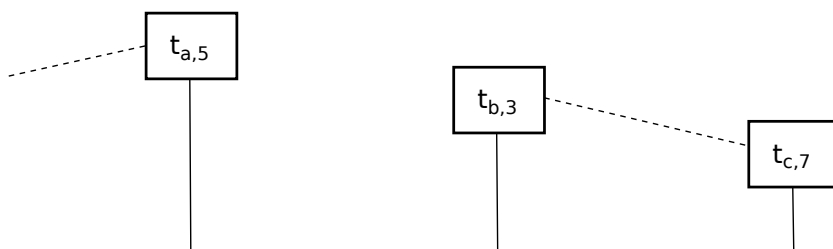

Figure A.1: We consider three nodes and begin in a state where some facilitators $\mathcal{F}_{r-2}$ just agreed on the consensus result $\mathcal{C}_{r-1}$ but have not yet propogated it yet.
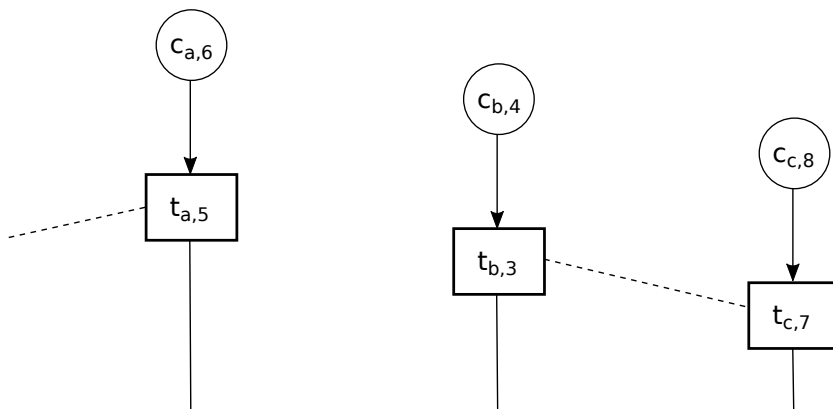

Figure A.2: The facilitators propogates $\mathcal{C}_{r-1}$. Upon receiving it, the nodes perform two tasks: (1) elect new facilitators $\mathcal{F}_{r-1}$ by selecting the first n nodes ordered by $\mathsf{H}(\mathcal{C}_{r-1}||pk)$, and (2) create new CP blocks—$c_{a,6}, c_{b,4}$ and $c_{c,8}$. For example, $c_{a,6} = \langle \mathsf{H}(t_{a,5}), 6, \mathsf{H}(\mathcal{C}_{r-1}), r-1, sig_a \rangle$.
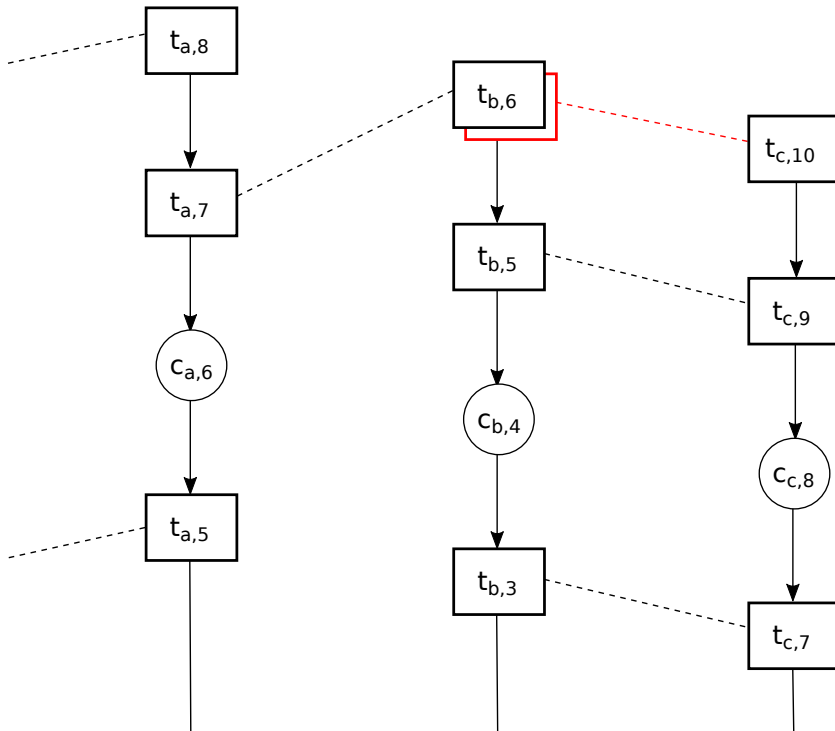
Figure A.3: Nodes now send their new CP blocks to the new facilitators $\mathcal{F}_{r-1}$. While $\mathcal{F}_{r-1}$ is trying to reach consensus on a set of CP blocks, nodes carry on making transactions as usual (concurrently). We remark that the to-be-agreed consensus result $\mathcal{C}_r$ is created using CP blocks from the previous round—$c_{a,6}$, $c_{b,4}$ and $c_{c,8}$.
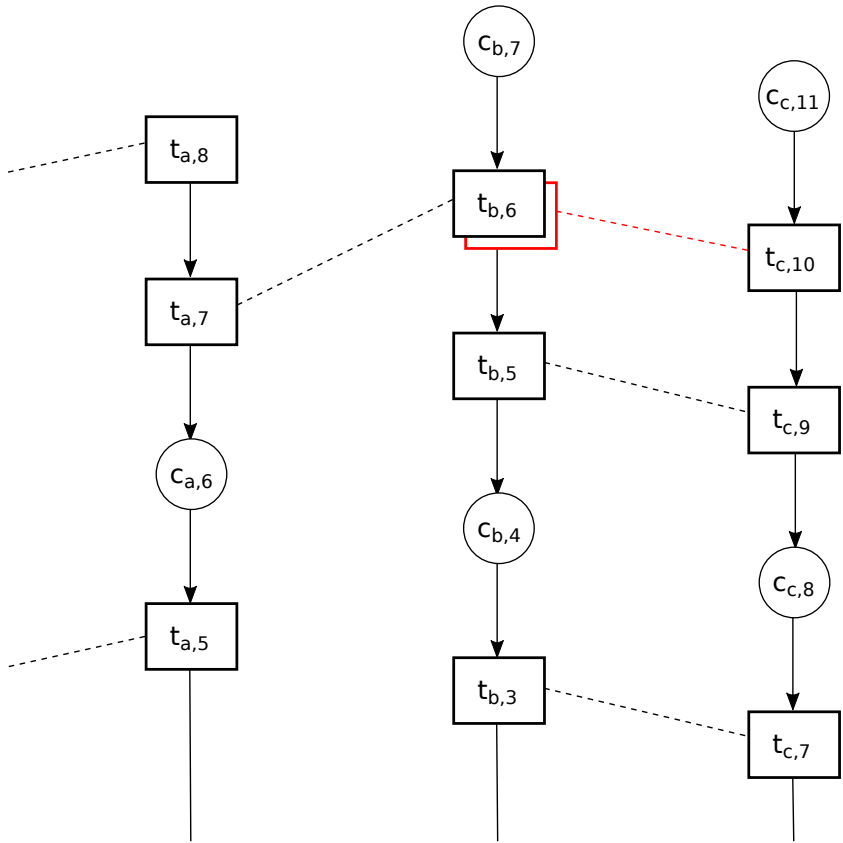
Figure A.4: Finally, when $\mathcal{F}_{r-1}$ decides on $\mathcal{C}_r$ (which should contain $c_{a,6}, c_{b,4}$ and $c_{c,8}$) and disseminates it. Nodes compute new facilitators $\mathcal{F}_r$ and create new CP blocks. For example, $c_{c,11} = \langle \mathsf{H}(t_{c,10}), 11, \mathsf{H}(\mathcal{C}_r), r, sig_c \rangle$.