A Blockchain Consensus Protocol With Horizontal Scalability

Kelong Cong
Delft University of Technology
Email: k.cong@student.tudelft.nl

Zhijie Ren
Delft University of Technology
Email: z.ren@tudelft.nl

Johan Pouwelse
Delft University of Technology
Email: peer2peer@gmail.com

***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 show a number of highly desirable correctness properties such as implicit consensus on the validity of transactions. Further, we give a free and open-source implementation of CHECO and evaluate it experimentally. Our results show a strong indication of horizontal scalability.**

I. INTRODUCTION

Bitcoin is almost ten years old, and its market capitalisation is over $60 billion USD and growing [1]. We can be reasonably sure that cryptocurrencies, even if their application is still somewhat limited, are here to stay in the foreseeable future. Driven by the success of Bitcoin, we see a renaissance of consensus research [2]–[4], where the primary focus is to improve the scalability of blockchain systems, which is due to the inefficiencies of the consensus mechanism—proof-of-work (PoW). For example, Bitcoin can only do 7 transactions per second (TPS) at most [5]. While adjusting the block size (which Bitcoin has recently done via SegWit [6]) and/or the block interval may increase TPS, it also leads to centralisation as larger blocks take longer to propagate through the network, putting miners that do not have a fast network at a disadvantage [7]. Furthermore, due to the bandwidth and latency of today's network, it is not possible to achieve more than 27 TPS from simply adjusting the block size or block interval [7].

Related work. Many approaches exist for improving the scalability of early blockchain systems. Off-chain transactions make use of the fact that if nodes make frequent transactions, then it is not necessary to store every transaction on the blockchain, only the net settlement is needed. The best examples are Lightning Network [8] and Duplex Micropayment Channels [9]. It promises significant scalability improvements, but complicates user experience and leads to centralisation. Each node must deposit a suitable amount of Bitcoins into a multi-signature account. A low deposit would not allow large transactions. A high deposit locks the user from using much of her Bitcoins outside 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. Moreover, 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, leading to centralisation.

Another way to improve transaction rate is to use traditional Byzantine consensus algorithms such as PBFT [10] in a permissioned ledger such as Hyperledger Fabric [11]. In essence, such systems contain a fixed set of nodes (sometimes called validating peer) that run a Byzantine consensus algorithm to decide on new blocks. Such systems can achieve much higher transaction rates, e.g., 10,000 TPS if the number of validating peer is under 16 for PBFT [2, Section 5.2]. However, these systems do not scale, e.g., the transaction rate drops to under 5000 TPS when the number of validating peer is 64 [2, Section 5.2]. Moreover, the validating peers are predetermined which makes the system unsuitable for the open internet.

Recent research has developed a class of hybrid systems which uses PoW for committee election, and Byzantine consensus algorithms to agree on transactions, e.g., ByzCoin [3] and Solidus [12]. This design is primarily for permissionless systems because the PoW leader election aspect prevents the Sybil attack [13]. It overcomes the early blockchain scalability issue by delegating the transaction validation to a Byzantine consensus protocol. A tradeoff of such systems is that they cannot guarantee a high level of fault tolerance when there is a large number of malicious nodes (but less than a majority). ByzCoin and Solidus all have some probability of electing 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 [14]). Again, because these systems must reach consensus on all transactions, none of them achieves horizontal scalability.

Finally, a technique that does achieve horizontal scalability is sharding, e.g., Elastico [15] and OmniLedger [4]. It involves grouping nodes into multiple committees of constant size, also known as shards, and nodes within a single shard run a Byzantine consensus algorithm to agree on a set of transactions that belong to that specific shard. The number of shards grows linearly with respect to the total computational power of the network; hence the transaction rate also grows linearly. The 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 degrade the transaction throughput 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 hinders scalability.

Research question. Thus far, there are no systems that achieve horizontal scalability in the general case, which leads to the goal of this work. Hence the research question which we wish to answer is as follows.

> *How can we design a horizontally scalable blockchain consensus protocol?*

Concretely, a blockchain consensus protocol should be application neutral. For example, PoW is application neutral because transaction semantics does not affect it, i.e. it can be applied in different applications such as cryptocurrency (Bitcoin) and domain name system (Namecoin [16]). Further, we are interested in horizontal scalability in the general case as it enables ubiquitous use. That is, adding more nodes to the network should result in higher transaction throughput.

Contribution. The key insight is not to reach consensus using an existing consensus algorithm (a modified HoneyBadgerBFT [2]) on transactions themselves, but on special blocks called checkpoint blocks, such that transactions are nevertheless verifiable at a later stage by any node in the network. Our main contributions are the following.

- We formally introduce a blockchain system—CHECO[1]. It uses individual hash chains and checkpoints on every node to achieve horizontal scalability in the general case for the first time.
- We analyse CHECO to ensure correctness according to our definition.
- We provide an implementation and then experiment with up to 1200 nodes, our results show strong evidence of horizontal scalability.

Roadmap. In Section II, we give the problem description and our system model. Section III gives the formal system architecture. In Section IV, we discuss a few design variations and their tradeoffs. We argue the correctness and fault tolerance properties of our system in Section V. Then we evaluate our system experimentally in Section VI. Finally, we conclude our work in Section VII.

II. Problem Description

We introduce the problem as a modified Byzantine consensus problem. The modification is primarily derived from the need of horizontal scalability, which is not a part of a typical Byzantine consensus problem. In our model, we consider N nodes, t of which are Byzantine. Nodes in our system make transactions with each other. Transactions can be in one of three states—*valid*, *invalid* and *unknown*. We seek a protocol that satisfies the following properties.

- *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 decide *unknown*.
- *Scalability*: If every node makes transactions at the same rate, then as N increases, the global transaction rate should increase linearly w.r.t. N.

Note that the agreement property is similar, but a relaxed version of what is often seen in a Byzantine consensus problem. Namely, the property only holds if honest nodes do not output *unknown*. For example, for a transaction, it is OK if two honest nodes output *valid* and *unknown*, but it is not OK to output *valid* and *invalid*. Our problem does not have a termination property, also known as liveness. Instead, nodes are incentivised to complete the protocol execution; we describe this phenomenon in Section V-B.

The problem is purposefully made to be application neutral, i.e. there are no constraints on the semantics of transactions. This formulation is so that the protocol can act as a building block to many applications. Due to this fact, we also do not consider global fork prevention or detection, as some application may not need such strong guarantees such as the accounting of internet traffic in Tribler [17], [18]. On the other hand, we give two alternative constructions that do have fork detection in Section IV-C.

System model. We assume purely asynchronous channels with eventual delivery. Thus in no stage of the protocol are we allowed to make timing assumptions. The adversary has full control of the delivery schedule and the message ordering of all messages.

Security assumptions. The malicious nodes are Byzantine, meaning that there are no restrictions on the type of failure. We use a static, round-adaptive corruption model. That is, if a round has started, the corrupted nodes cannot change until the next round. 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 the hash function is negligible with respect to λ [19].

III. System Architecture

To describe CHECO, we first give an informal overview and then move on to the formal description.

Early blockchain systems that use a global ledger are difficult to scale because every node must reach consensus on all the transactions that ever existed. Instead, we introduce an alternative architecture where every node has their own genesis block and hash chain. The nodes only store transactions (TX) that they are involved in on their hash chain. Transactions are stored in TX blocks, and every block only contains one transaction. A transaction between two nodes should,

[1] Derived from "*Che*ckpoint *co*nsensus".

therefore, result in two TX blocks on their respective hash chains. We introduce a special block called checkpoint (CP) block, which represents the state of a hash chain in the form of a hash pointer. Then, a collection of CP blocks from all nodes would represent the state of the whole system. A visualisation can be seen in Figure 1.

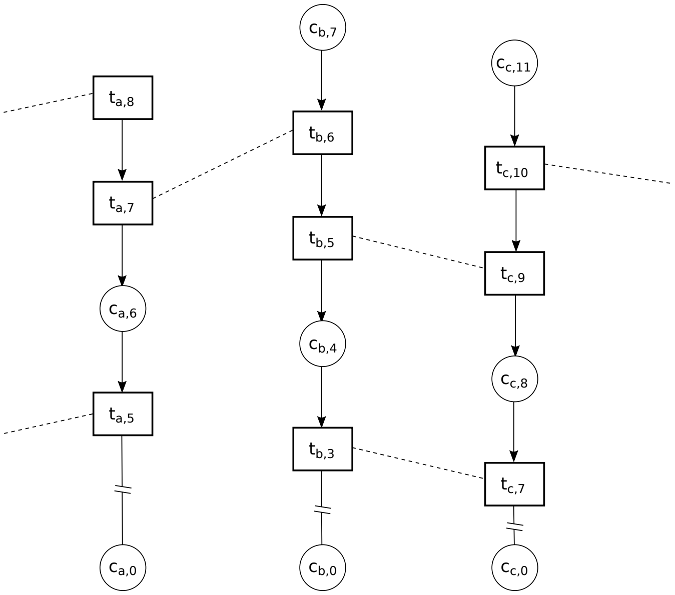


Fig. 1: Visualisation of the data structure used in CHECO. $t_{u,i}$ represents a TX block on u's chain with a sequence number i. $c_{v,j}$ represents a CP block on v's chain with a sequence number j. The blocks at the ends of the dotted lines are pairs of each other. Blocks of sequence number 0 (e.g., $t_{c,0}$) are genesis blocks.

CHECO consists of three protocols—consensus protocol, transaction protocol and validation protocol—all interacting with the distributed hash chain data structure described above. The primary protocol is the consensus protocol, which can be seen as a technique of running infinitely many times of an existing Byzantine consensus algorithm (in this work we use asynchronous common subset from HoneyBadgerBFT [2]), starting a new execution immediately after 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. 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. 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.

The transaction 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. 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 section of the counterparty's chain beginning and ending with CP blocks but contains the TX block belonging to that transaction, where the CP blocks must be in consensus. Upon the counterparty's response, the node checks whether the CP blocks are, in fact, in some consensus result and among other conditions. The transaction is valid if these conditions are satisfied. Since the transaction and validation protocols only make point-to-point communication, we achieve horizontal scalability.

For brevity, we left out issues such as selecting facilitators in a fair way and the exact validity condition. The following sections give the formal description and will answer these issues.

A. CHECO *data structure*

Each node u has a public and private key pair—pk_u and sk_u, and a hash chain B_u. The chain consist of blocks $B_u = \{b_{u,i} : i \in \{0, \dots, h-1\}\}$, 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

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

We describe each item in turn.

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

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

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

there exist one and only one pair

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

if the nodes follow the transaction protocol (described in Section III-C). 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.

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

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

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

where $\mathsf{H}(\perp)$ can be interpreted as applying the hash function on an empty string. The genesis block is unique because every node has a unique public and private key pair.

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.

Here we define a property which results from the interleaving nature of CP and TX blocks. It is used in our validation protocol (discussed in Section III-D).

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

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

B. Consensus Protocol

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

Definition 6. CHECO consensus protocol

A CHECO *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) *$|\mathcal{C}_r| \geq N - t$[2], 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 $\mathcal{F}_r$.*

1) Bootstrap Phase: 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.

2) 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 `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 `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, which is derived from HoneyBadgerBFT [2]. We do not use the full HoneyBadgerBFT due to the following. First, the transactions in HoneyBadgerBFT are first queued in a buffer

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

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 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 $\mathsf{new\_cp}(\mathcal{C}_{r+1})$ (Algorithm 1). Second, it computes the new facilitators using $\mathsf{get\_facilitator}(\mathcal{C}_{r+1}, n)$ (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 [20, 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.

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

C. Transaction Protocol

The TX protocol Π_t, shown in Algorithm 4, is run by all nodes. 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.

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

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

Algorithm 4 Π_t runs in the context of node u.

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

A key feature of Π_t 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 tx_req messages multiple times without waiting for the corresponding tx_resp messages.

D. Validation Protocol

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

1) Validity Definition: A transaction can be in one of three states in terms of validity—*valid*, *invalid* and *unknown*. Given a fragment $F_{v,j}$, the validity of the TX block $t_{u,i}$ with its corresponding fragment $F_{u,i}$ is captured by the function $\mathsf{get\_validity}(t_{u,i}, F_{u,i}, F_{v,j})$ (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 is not difficult to achieve 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.

We stress that the *unknown* state means that the verifier does not have enough information to make progress in Π_v. 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, i.e. two TX blocks that form a pair. 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 identified by its $txid$ 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 always correct and contains $t_{u,i}$. $F_{v,j}$ is the corresponding fragment received from v.

```
if $F_{v,j}$ is not a fragment created in the same round as $F_{u,i}$
then
    return unknown
$\langle \_, \_, txid, pk_v, m, \_ \rangle \leftarrow t_{u,i}$
if number of blocks of $txid$ in $F_{v,j} \neq 1$ then
    return invalid
$t_{v,j} \leftarrow$ the TX block with $txid$ in $F_{v,j}$
$\langle \_, \_, txid', pk_u, m', \_ \rangle \leftarrow t_{v,j}$
if $m \neq m' \vee txid \neq txid'$ then
    return invalid
if $t_{u,i}$ is not signed by $pk_u \vee t_{v,j}$ is not signed by $pk_v$ then
    return invalid
return valid
```

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 $\langle \texttt{vd\_req}, txid \rangle$ to v. The desired properties are as follows.

Definition 7. CHECO validation protocol

A CHECO *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 decide* 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 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 $txid$
    $F_{u,i} \leftarrow$ 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 get_validity($t_{u,i}, F_{u,j}, F_{v,j}$)
```

We make two remarks. First, just like Π_t, 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.

IV. Design Variations and Tradeoffs

In this section, we explore a few design variations, some of them require a relaxed version of our original model. They enable better performance and allow us to apply our design in the fully permissionless setting.

A. Using Timing Assumption in the Permissionless Setting

At the start of our consensus phase (Section III-B2), facilitators must wait for $N - f$ $\texttt{cp\_msg}$ messages. The use of N makes our system unsuitable for the permissionless setting. To introduce timing, instead of waiting for $N - f$ messages, we wait for some time D, such that D is sufficiently long for honest nodes to send their CP blocks to the facilitators. Consequently, this removes the need for a PKI because the collected CP blocks may be from nodes that nobody has seen in the past.

The new protocol handles churn as follows. Suppose a new node wish to join the network and the facilitators are known (this can be done with a public registry). It simply sends its latest CP block to the facilitators. Then, in the next round, the node will have a chance to become a facilitator just like any existing node. To leave the network, nodes simply stop submitting CP blocks. There is a subtlety here which happens when the node is elected as a facilitator in the following round. In this case, the node must 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.

B. Optimising Validation Protocol Using Cached Agreed Fragments

One more way to improve the efficiency of Π_v 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.

C. 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 [21]. 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.

V. Correctness and Fault Tolerance Analysis

We evaluate our system analytically to ensure the desired properties (Definition 6 and Definition 7) hold. An informal argument is given in this section. We refer to [22, Chapter 4] for an in-depth analysis.

A. Correctness of the Consensus Protocol

Π_c correctly implements the CHECO consensus protocol (Definition 6) due to the following. The agreement, validity and termination properties hold because:

- The CP blocks sent to the facilitators are eventually delivered, and then ACS eventually starts.
- Agreement, validity and termination hold for ACS as they are the properties of ACS and are proven to hold in [2].
- The consensus result and signatures are eventually disseminated to all the nodes, so honest nodes must hold the same result as the honest facilitators.

Fairness holds because we model $\mathsf{H}(\cdot)$ as a query to the random oracle. Sorting nodes by the luck value $\mathsf{H}(\mathcal{C}||pk)$ can be seen as a random permutation of a list of nodes. Therefore every node with a CP block in the consensus result has the same probability of becoming a facilitator.

B. Correctness of the Validation Protocol

Using the previous result, we show that Π_v implements the agreement and validity properties of a CHECO validation protocol (Definition 7).

The validity property holds because we use $\mathsf{get\_validity}(\cdot)$ in the validation protocol. The agreement property holds because we model $\mathsf{H}(\cdot)$ as a query to a random oracle. That is, suppose two honest nodes decided on two different states, *valid* and *invalid* for the same transaction. For that to happen, two agreed fragments must exist for the same transaction, but these fragments must also have the same agreed enclosure. Recall that blocks form a hash chain. So this is not possible unless the adversary can compute the inverse of $\mathsf{H}(\cdot)$ with high probability.

Liveness, unfortunately, does not hold in our model. A malicious node can act honestly when running the transaction protocol, but then never respond to any validation requests. Therefore some transactions can never be validated. Nevertheless, the malicious node will be at an economic loss if it is not responsive because honest nodes are less likely to make contact with nodes that do not respond to validation requests. If the probabilistic fork detection proposal (Section IV-C) is used, the uncooperative nodes will have more incentive to participate in the protocol.

C. Fault Tolerance

Finally, we consider the effect when the number of adversaries is more than t, which 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 is modelled by a hypergeometric distribution, where the random variable X is the number of malicious nodes. For our system to fail, we are interested in the following probability.

$$\Pr[X \geq \lfloor \frac{n-1}{3} \rfloor + 1]$$

Let $E[X] = n\alpha$ where $0 \leq \alpha \leq \lfloor \frac{n-1}{3} \rfloor / n$. Using the tail inequality found in [23]

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

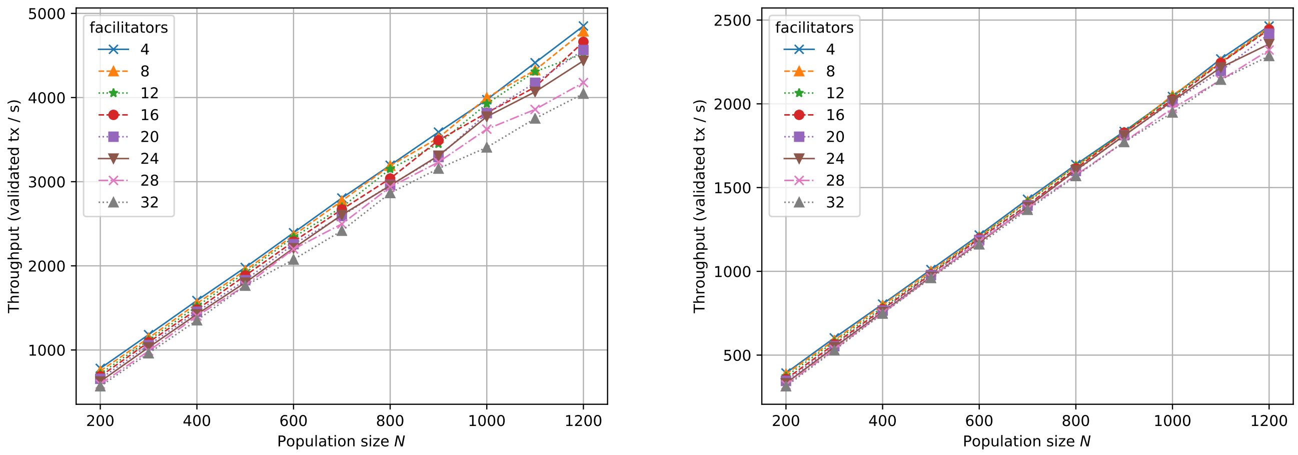
and setting

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

we arrive at 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 the desired level of fault tolerance. If n is known, then we can pick a α such that the probability becomes small. On the other hand, if α is fixed, we may increase n to achieve the same. Furthermore, hypergeometric distributions have light tails with "faster-than-exponential fall-off" [23]. So the probability of picking more than $\lfloor \frac{n-1}{3} \rfloor$ malicious nodes drops off exponentially as the expected value moves away from $\lfloor \frac{n-1}{3} \rfloor$. To put this into perspective, suppose $n = 100$ and $\alpha = 1/10$, i.e. 10% of the nodes are malicious, then the probability to elect more malicious nodes than the threshold is 9.9×10^{-6}.



(a) Every node make transactions with a fixed node.

(b) Every node make transactions with a random node.

Fig. 2: Global throughput increases as the population increases when every node transact at the same rate. Making transactions with fixed nodes results in a higher throughput because of the caching mechanism.

VI. Implementation and Evaluation

A free and open source implementation can be found on GitHub: https://github.com/kc1212/checo. It implements the three protocols and the Extended TrustChain. We also implement the caching optimisation discussed in Section IV-B. It is written in the Python programming language. The cryptography primitives we use are SHA256 for hash functions and Ed25519 for digital signatures.

We run the experiment on the DAS-5[3] with up to 1200 nodes. Every node makes transactions at 2 per second. Since Bitcoin transactions are approximately 500 bytes [24], we use a uniformly random transaction size sampled between 400 and 600 bytes.

The global throughput results are shown in Figure 2. We consider Figure 2a as the ideal case, where nodes only make transactions with a fixed node. Figure 2b is the worst case, where nodes make transactions with random nodes and the caching mechanism is unlikely to be used. Observe that the transaction rate is much lower in Figure 2b, which is because the communication of an agreed fragment is necessary to verify every transaction (no caching), putting a strain on our network infrastructure.

For Figure 2a, the magnitude of our throughput may not be self-evident at first glance. Recall that we fixed the transaction rate to 2 TPS, but how is it possible to have around 4800 transactions per second for 1200 nodes (which is 4 TPS)? It 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. Overall, 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.

The downside of our design is that the communication complexity of the consensus protocol grows polynomially as the number of facilitators grows linearly. Hence, the consensus protocol will take longer to complete, and larger fragments must be sent for transaction verification. On the other hand, it does not significantly impact the throughput; only the transaction verification delay is affected. We refer the reader to [22, Chapter 5] for additional analysis of the effect of the number of facilitators as well as other experimental results.

VII. Conclusion

In this work, we described Checo, an application neutral blockchain system with horizontal scalability. Our novel data structure allows nodes to efficiently store transactions and record state using CP blocks. The round based consensus protocol uses ACS as a building block to reach consensus on CP blocks. The consensus result lets nodes elect new facilitators and create new checkpoint blocks. To make transactions, nodes use the simple and non-blocking transaction protocol. Finally, we introduce a validation protocol which ensures that if an agreed fragment for some transaction exists, then nodes reach agreement on the validity of that transaction. The novelty of Checo is that it decouples consensus and transaction validation, which enables the desirable horizontal scalability property.

We achieve the properties described in Section II. Namely, our protocol achieves agreement on transactions as we argued in Section V-B. Further, the horizontal scalability is demonstrated in Section VI, in the ideal case as well as the worst case. In the future, we hope to apply our system to a concrete application and evaluate its performance.

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

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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 06/25/2017).

[7] 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.

[8] J. Poon and T. Dryja, "The bitcoin lightning network," Jan. 2016. [Online]. Available: https : / / lightning . network/lightning-network-paper.pdf.

[9] 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.

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

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

[12] 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.

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

[14] 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.

[15] 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.

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

[17] P. Otte, "Sybil-resistant trust mechanisms in distributed systems," Master's thesis, Delft University of Technology, Dec. 2016. [Online]. Available: http://resolver.tudelft . nl / uuid : 17adc7bd - 5c82 - 4ad5 - b1c8 - a8b85b23db1f.

[18] 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.

[19] 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.

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

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

[22] K. Cong, "A blockchain consensus protocol with horizontal scalability," Master's thesis, Delft University of Technology, Aug. 2017. [Online]. Available: http://resolver.tudelft.nl/uuid:86b2d4d8-642e-4d0f-8fc7-d7a2e331e0e9.

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

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