Bare Demo of IEEEtran.cls for IEEE Conferences

Michael Shell
School of Electrical and
Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Email: http://www.michaelshell.org/contact.html

Homer Simpson
Twentieth Century Fox
Springfield, USA
Email: homer@thesimpsons.com

James Kirk
and Montgomery Scott
Starfleet Academy
San Francisco, California 96678–2391
Telephone: (800) 555–1212
Fax: (888) 555–1212

***Abstract*—The abstract goes here.**

I. Introduction

II. Background and Related Work

A. Background on asynchronous subset consensus

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 [1]).

Definition 1. Asynchronous subset consensus

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

ACS has the nice property of censorship resilience when compared to other consensus algorithms. For instance, Hyperledger and Tendermint uses a variant of Practical Byzantine Fault Tolerance (PBFT) [2] as their consensus algorithm. In PBFT, a leader is elected, if the leader is malicious but follows the protocol, then it can selectively filter transactions. In contrast, every party in ACS are involved in the proposal phase, and it guarantees that if $n-2t$ parties propose the same transaction, then it must be in the agreed output. Thus, if some value is submitted to at least $n-2t$ nodes, it is guaranteed to be in the consensus result. For a detailed description of ACS, we refer to the HoneyBadgerBFT work [1]. To understand the remainder of this work, the knowledge of Definition 1 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.

B. 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 [3] and Duplex Micropayment Channels [4].

Off-chain transaction systems are implemented using multi-signature addresses [5] and hashed time-locked Bitcoin contracts [6]. In the simplest case, if two parties wish to make 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, this is called the opening transaction. Both parties keep the channel state which is not broadcasted to the network. The state is updated when the two parties make new transactions. The protocol disincentivises the parties from broadcasting old channel states. If this 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, this is called the closing transaction. Effectively, only two transactions need to be broadcasted to the Bitcoin network—the opening and the closing transaction, even if the two parties made thousands or millions of transactions in between. The two-party scheme can be extended to a network of channels, which allows two parties to make off-chain transactions without an open channel as long as they are connected by nodes that do.

The advantage of such systems is that they act as add-ons to Bitcoin wish already has a large number of users. Thus, if enough of the network wish for it (by setting a new block version), then a large number of users will instantly benefit from it. Some 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, off-chain transactions also suffer from the problem of Bitcoin. Namely, proof-of-work consumes an

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

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

C. Permissioned systems based on Byzantine consensus

This category of systems uses traditional Byzantine consensus algorithms such as PBFT [2]. In essence, they contain a fixed set of nodes (sometimes called validating peer) that run a Byzantine consensus algorithm to decide on new blocks. This is often used in permissioned system where the validators must be predetermined, for example, Hyperledger Fabric [7].

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 [1, Section 5.2]. Further, in contrast to proof-of-work, PBFT consensus is final. That is, transaction history cannot be re-written if it is already in the blockchain.

The major drawback of Byzantine consensus based systems is that it does not scale in terms of the number of validating peers. Going back to PBFT, its transaction rate drops to under 5000 TX/s when the number of validating peer is 64 [1, Section 5.2]. Moreover, the validating peers are predetermined which makes the system unsuitable for the open internet.

D. Combining proof-of-work 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. This design is primarily for permissionless system because the PoW leader election aspect prevents Sybils. Some examples are SCP [8], ByzCoin [9] and Solidus [10].

This technique overcomes the early blockchain scalability issue by delegating the transaction validation to the Byzantine consensus protocol (e.g. PBFT in ByzCoin [9]). A major trade-off of such systems is that they cannot guarantee correctness when there is a large number of malicious nodes (but less than a majority). For SCP, ByzCoin and Solidus, they all have some probability to elect more than t Byzantine nodes into the committee, where t is typically just under a third of the committee size (a lower bound of Byzantine consensus [11]). This problem is especially difficult to solve because the committee is always much smaller than the population size which has more than t Byzantine nodes. Classical blockchain 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.

E. Sharding

Sharding is a technique to achieve horizontal scalability by grouping nodes into multiple committees or shards. Nodes within a single shard run a Byzantine consensus algorithm to agree on a set of transactions that belong to that specific shard. An intra-shard protocol is needed for transactions that involve nodes from more than one shard. The number of 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 [12] and OmniLedger [13].

The biggest limitation of sharding is that it is only optimal is transactions stay in the same shard. In fact, Elastico cannot atomically process inter-shard transactions. OmniLedger has an inter-shard transaction protocol but choosing a good shard size is difficult. A large shard size would make the system less scalable because the Byzantine consensus algorithm must be run by a large number of nodes. A small shard size would result in a large number of inter-shard transactions which also hinder scalability. The authors of OmniLedger noted that inter-shard transactions have significantly higher latency compared to the consensus protocol [13]. Furthermore, since no shards can be compromised for the system to function correctly, the adversaries have more opportunities to compromise the system.

F. Blockchain without global consensus

Tangle [14], Corda [15] and the original TrustChain [16] do not use global consensus at all. In this scheme, every node has their own chain. Transactions between nodes are recorded on their respective chains. This effectively results in a directed acyclic graph. By avoiding global consensus, this approach achieves extremely desirable scalability properties similar to BitTorrent [17].

On the other hand, the application and security 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.

III. System Architecture

Our system consists of three protocols—the consensus protocol, the transaction protocol and the validation protocol. It is based on our prior work on TrustChain [16], where every node independently interacts with each other via their own blockchain[2].

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

A. Model and assumptions

This section and the ones following it give a technical treatment of what the content in System overview. For notational clarity, we use the following convention (adapted from [1]) 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 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 message ordering of all messages. But they are not allowed to drop messages except for their own[3].

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 [18].

In our model, we consider N nodes, which is the population size. n of them are facilitators, t out of n are malicious and the inequality $n \geq 3t+1$ must hold. This is from the work of Pease, Shostak and Lamport, where they show a network of n nodes cannot reach Byzantine agreement with $t \geq n/3$ [11]. Further, the inequality $N \geq n+t$ must also hold. This is due to our system design, which becomes clear in Section III-C2.

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

B. Extended TrustChain

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

Definition 2. Transaction block
The TX block is a six-tuple, i.e

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

We describe each item in turn.

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

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

there exist one and only one pair

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

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

Definition 3. Checkpoint block
The CP block is a five-tuple, i.e.

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

where $\mathcal{C}_r$ *is the consensus result (which we describe in Definition 4) 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 4. Consensus result
Our consensus protocol runs in rounds as discussed in **??**. *Every round is identified by a round number* r*, which is incremented on every new round. The consensus result is a tuple, i.e.* $\mathcal{C}_r = \langle r, C \rangle$*, where* C *is a set of CP blocks agreed by the facilitators of round* r*.*

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

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

Definition 5. Enclosure and agreed enclosure

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

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

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

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

Note that the "closest CP blocks" property must also apply to agreed enclosure. Suppose a chain is in the form $\{c_i, c_{i+1}, t_{i+2}, c_{i+3}\}$[4] and c_i, c_{i+1}, c_{i+3} are in $\mathcal{C}_r, \mathcal{C}_{r+1}, \mathcal{C}_{r+3}$ respectively, then the agreed enclosure of t_{i+2} is $\langle c_{i+1}, c_{i+3}\rangle$ and cannot be $\langle c_i, c_{i+3}\rangle$.

Definition 6. 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 $\{b_{u,i} : a \le i \le b\}$. Similarly,* agreed fragment *has the same definition as fragment but using agreed enclosure. For convenience, the function* agreed_fragment$(\cdot)$ *represents the agreed fragment of some TX block if it exists, otherwise $\perp$.*

C. Consensus Protocol

1) Bootstrap phase: Now we have all the necessary information to describe our consensus protocol. As with all distributed systems, there must be a bootstrap phase which sets up the system.

The reader may notice from the system overview (**??**) that the facilitators are computed from the consensus result, but the consensus result is agreed by the facilitators. Thus we have a dependency cycle. The goal of the bootstrap phase is to give us a starting point in the cycle.

To bootstrap, imagine that there is some bootstrap oracle, that initiates the code on every node. The code satisfies all the properties in **??**. Namely, every node has the same set of valid facilitators $\mathcal{F}_1$ that are randomly chosen. This concludes the bootstrap phase. For any future rounds, the consensus phase is used.

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

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

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

There are two scenarios in the consensus phase. First, if node u is not the facilitator, it sends $\langle \mathtt{cp\_msg}, c_{u,h}\rangle$ to all the facilitators. Second, if the node is a facilitator, it waits until $N-t$ messages of type `cp_msg` are received. Invalid messages are removed. 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 (after ACS) because the adversary may send different CP blocks to different facilitators, which results in invalid blocks in the ACS output, but not in any of the inputs.

The core of the consensus phase is the ACS protocol. While any ACS protocol that satisfies the standard definition will work, we use a simplification of HoneyBadgerBFT [1] as our ACS protocol 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 as it contains no transaction information.

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

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

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

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

Our protocol has some similarities with synchronizers [19, Chapter 10] because it is effectively a technique to introduce synchrony in an asynchronous environment. If we consider

Algorithm 2 Function 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.

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

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.

D. Transaction Protocol

The TX protocol, shown in Algorithm 4, is run by all nodes. It is also known as True Halves, first described by Veldhuisen [20, Chapter 3.2]. Nodes that wish to initiate a transaction calls 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 new_tx$(pk_v, m, txid)$ generates a new TX block and appends it to the caller u's chain. It is executed in the private context of u, i.e. it has access to the sk_u and B_u. The necessary arguments are the public key of the counterparty pk_v, the transaction message m and the transaction identifier $txid$.

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

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

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

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

E. Validation protocol

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

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 $t_{u,i}$ and the agreed fragment of it 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 is not difficult to achieve. 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 Algorithm 5, 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*. This is likely to be the case for new transactions because the result of agreed_fragment$(\cdot)$ would be $\perp$. The next two conditions check for tampering or missing blocks, if any of these misconducts are detected, then the TX is invalid.

We stress that the *unknown* state means that the verifier does not have enough information to continue with the validation protocol. If enough information is available at a later time, then the verifier will output either *valid* or *invalid*.

Note that the validity is on a transaction (two TX blocks with the same $txid$), rather than on one TX block owned by a single party. It is defined this way because the malicious sender may create new TX blocks in their own chain but never send `tx_req` messages. In that case, it may seem that the counterparty, who is honest, purposefully 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.

Further, the caller of get_validity$(t_{u,i}, F_{u,i}, F_{v,j})$ is not necessarily u[5] Any node w may call get_validity$(t_u u, i, F_{u,i}, F_{v,j})$ as long as w has an the necessary input parameters. $F_{u,i}$ and $t_{u,i}$ may be readily available if $w = u$. Otherwise, w could query u with the `vd_req` message and then obtain the result from `vd_resp`. This is the validation protocol which we describe next.

2) Validation protocol: Our validation protocol, 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 of the validation protocol are as follows.

Definition 7. Properties of the validation protocol

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

Algorithm 5 Function get_validity$(t_{u,i}, F_{u,i}, F_{v,j})$ 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*

1) Agreement*: If any correct node decides on the validity (except when it is* unknown*) of a transaction, then all other correct nodes are able to reach the same conclusion or* unknown*.*
2) Validity*: The validation protocol outputs the correct result according to the aforementioned validity definition.*
3) Liveness*: Any valid (invalid) transaction is marked as valid (invalid) eventually.*

Algorithm 6 Validation protocol which runs in the context of u

Upon $\langle \texttt{vd\_req}, txid \rangle$ from v
  $t_{u,i} \leftarrow$ the transaction identified by $txid$
  $F_{u,i} \leftarrow$ 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 the TX protocol, we do not block at any part of the protocol. Second, suppose some $F_{v,j}$ validates $t_{u,i}$, then that does not imply that $t_{u,i}$ only has one pair $t_{v,j}$. Our validity requirement only requires that there is only one $t_{v,j}$ in the correct consensus round. The counterparty may create any number of fake pairs in later consensus rounds. But these fake pairs only pollutes the chain of v and can never be validated because the round is incorrect.

F. Design variations and tradeoffs

Up to this point, we have discussed our protocol in the context of the model and assumptions defined in Section III-A. In this section, we explore a few design variations which we can make, some of them require a relaxed version of our original model. They enable better performance, allow us to apply our design in the fully permissionless setting and improves privacy.

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 III-C2). While this guarantees delivery (since we assumed reliable channel), it is wasteful. For example, if every facilitator is honest, a node would receive n consensus results which are identical when only one is necessary. Furthermore, realistic networks do not offer a broadcast primitive, i.e. firewalls may block incoming messages from certain nodes.

An improvement is use an epidemic protocol [21] (also known as gossiping) instead of our broadcast approach. Typical epidemic protocols works as follows. Every node buffers every message it receives up to some buffer size b. Then it forwards the messages t number of times. Every time the message is sent to f random neighbours, f is often called the fan-out. The upside of using epidemic protocol is that the communication cost is distributed more evenly between nodes. This is especially true with a lazy push approach where the node who just received a message would push the message ID (e.g. digest) to its f random neighbours, and only push the full message if the neighbour explicitly requests the message [22]. 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 uses constant time. Another downside of some epidemic protocols (e.g. eager push) is that it is difficult to guarantee delivery. It is especially true when the parameters are not 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.

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) [23]. One limitation of our system is that we use the parameter N in our algorithms, which makes it unsuitable for the permissionless environment where users can join and leave at will. In this section we show how making a timing assumption would allow us to operate in the permissionless setting.

At the start of our consensus phase (Section III-C2), facilitators must wait for $N - f$ `cp_msg` messages. This is the only place where we used N as a parameter. To introduce

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

Suppose a new node wish to join the network and the facilitators are known (this can be done with a public registry). It simply sends its latest CP block to the facilitators. Then, in the next round, the node will have a chance to become a facilitator just like any existing node. To leave the network, nodes simply stop submitting CP blocks. There is a subtlety here which happens when the node is elected as a facilitator in the following round. In this case, the node must 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) Privacy preserving validation protocol using compact blocks: Our approach already has privacy preserving features in comparison to early blockchain systems. That is, the transactions for each node are only revealed during the validation protocol. Hence if two nodes never directly or indirectly interact with each other, their transactions are never revealed.

We can take our privacy-preserving property one step further by introducing another level of hash pointer indirection. The result is shown in Figure 1.

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

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

Each compact block has a corresponding full block (either a CP block or a TX block). The relationship is uniquely identified with Seq or Digest. Recall that our original validation protocol requires the nodes to send the full agreed fragment. With compact blocks, it is only necessary to send the compact version of the agreed fragment. The validation then proceeds in a similar fashion, provided that the pair of the to-be-validated TX block is known.

The space saving of this approach 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 [24]), 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%.

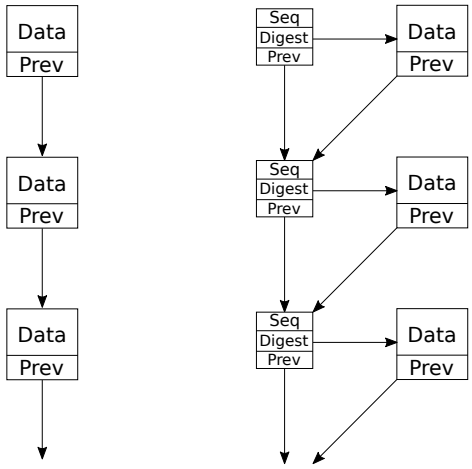

Fig. 1: The chain on the left represent direct chaining, where the digest in "Prev" is simply the digest of the previous block. The chain on the right uses compact blocks, represented by the smaller squares. Compact blocks also form a chain as before, but they each have a hash pointer to the full block, identified by "Seq" and "Digest".

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, upon receiving an agreed fragment from node A, rather than validating a single transaction, we attempt to validate all transactions with A in the unknown state in that fragment.

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

5) Global fork detection: The validation algorithm guarantees that there are no forks within a single agreed fragment. This is sufficient for some applications such as proving the 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.

There are a variety of approaches to do global fork detection. First and the easiest solution is to simply ask for the complete chain of the counterparty. The verifier can be sure that there are no forks if the following holds.

1) The hash pointers are correct.
2) All the CP blocks are in consensus.
3) The TX of interest is in the chian.

We use this approach in our prior work on Implicit Consensus [25]. It may sound inefficient at first, but nodes employ caching to minimise communication costs.

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 make sure 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 proof, then they run the transaction protocol as usual. If B provides an invalid proof or refuses to respond, then A would refuse to make the transaction. The probability that an honest node catches out a malicious node is

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

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

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

IV. Implementation

The prototype implementation can be found on GitHub.

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

It implements the three protocols and the Extended TrustChain discussed in **??**. We also implement two optimisations—privacy preserving validation protocol using compact blocks (Section III-F3) and optimised validation protocol using cached agreed fragments (Section III-F4). It is written in the event driven paradigm, using the Python programming language[6]. We use the Twisted[7] library for networking.

Every node keeps a persistent TCP connection with every other node. This creates a fully connected network for our experiment. It is certainly not ideal in real world scenarios where nodes may have limited resources (e.g. sockets). But as a prototype, it is sufficient to run a network of over a thousand nodes and experiment with it.

Finally, the cryptography primitives we use are SHA256 for hash functions and Ed25519 for digital signatures. Both of which are provided by libnacl [8].

A. Experimental setup

The goal of the experiment is to run the three protocols—consensus protocol, transaction protocol and validation protocol—simultaneously and analyse the communication cost, the consensus duration and the global throughput. We investigate these properties under the following parameters. 2 TX/s

1) The number of facilitators $n \in \{4, 8, \ldots, 32\}$. The maximum n is 32 because the limitation in liberasurecode mentioned in Section IV, but our results give a good indication of how our system may function for larger numbers of n.
2) The population size $N \in \{200, 300, \ldots, 1200\}$. N stops at 1200 is due to our physical setup, which we describe next.

The experiment is run on the DAS-5[9]. 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. This gives us the aforementioned 1200 number. With this setup, we cannot run more nodes because the every machine only has 65535 ports available (minus the reserved ones). But 40 nodes each need 1200 TCP connections which is 48000 TCP connections per machine and that is inching close to the limit. While it is possible to have 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, since Bitcoin transactions are approximately 500 bytes [24], we use a uniformly random transaction size sampled between 400 and 600 bytes.

V. Experimental Evalutation

A. Communication cost of the consensus protocol

Figure 2 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. This reinforces our analytical result in **??**. 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. This 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. This is also expected because the ACS algorithm has an n^2 term in its communication complexity.

We are also interested in how communication costs translate to time. Hence, for the same experiment, we record the duration in seconds and the result is shown in Figure 3. 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

[6]https://www.python.org/

[7]https://twistedmatrix.com/

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

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

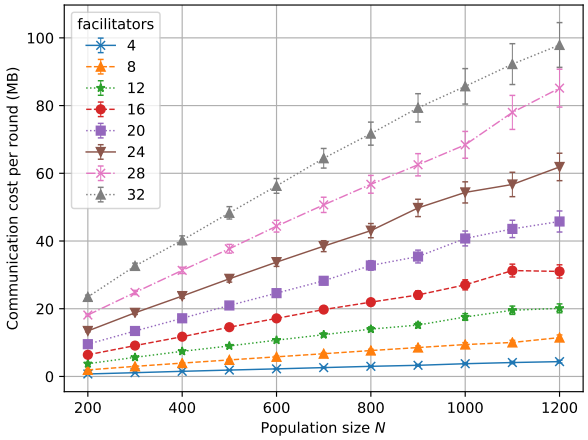

Fig. 2: Communication cost for the consensus protocol per round increases linearly with population size. The error bars are larger for higher population size or higher number of facilitators is because rounds take longer thus they are repeated fewer times.

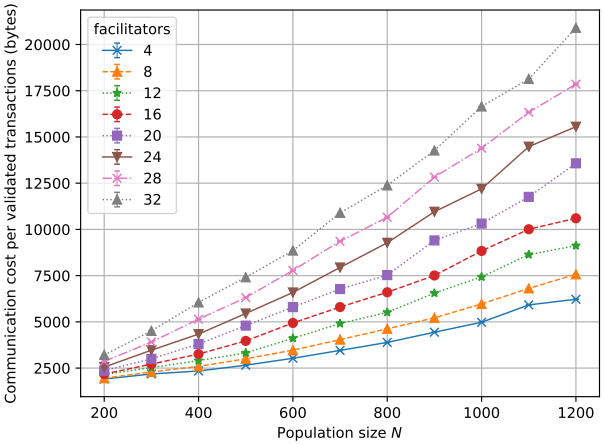

Fig. 4: Communication cost per verified transaction has similar (near linear) trend as Figure 3. Fluctuation for the fixed-neighbour mode exists because the cache mechanism is unpredictable.

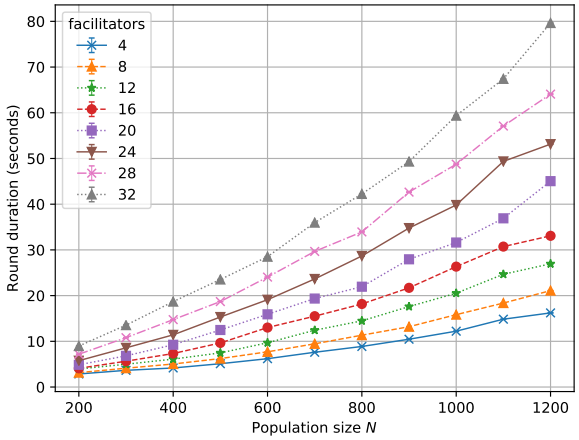

Fig. 3: Round duration does not increase linearly with the population size. This is likely due to the additional hashing operation required for larger consensus result.

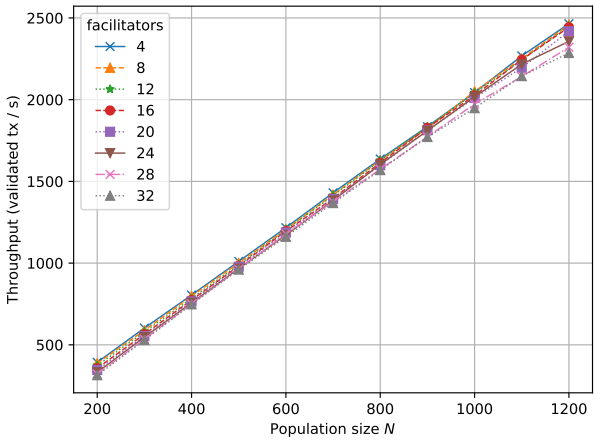

Fig. 5: Global throughput increases as the population increases when every node transact at the same rate. Using fixed neighbours results in a higher throughput because of the caching mechanism.

operations. These results do not conform to analytical result in **??**. Nevertheless, the difference is minor and there are ways to optimise the luck value computation. For example, the luck value can be computed by the facilitators and are sent with the consensus result. Then the non-facilitator nodes simply use it if there are enough signatures.

B. Communication cost of transaction and validation

C. Linear Global Throughput

Finally, the global throughput results are shown in Figure 5. 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.

VI. 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 for use in our consensus protocol. The round based consensus protocol uses ACS as a building block to reach consensus on checkpoint blocks. The consensus result, which is a set of checkpoint blocks, lets nodes elect new facilitators and create new checkpoint blocks. To make transactions, nodes use the transaction protocol,

which is adapted from an existing work called True Halves. 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.

The research question we asked in **??** is the following.

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

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

This work is the first step in building a new paradigm for blockchain consensus protocol. It has the potential to efficiently cultivate trust on the internet in the presence of faults without a central authority. We hope to improve our design by building a concrete application on top of it.

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] 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.

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

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

[4] 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.

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

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

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

[8] L. Luu, V. Narayanan, K. Baweja, C. Zheng, S. Gilbert, and P. Saxena, "Scp: A computationally-scalable byzantine consensus protocol for blockchains.," *IACR Cryptology ePrint Archive*, vol. 2015, p. 1168, 2015.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

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

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

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

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

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

[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).

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