Blockchain Consensus Protocol with Horizontal Scalability

K. Cong

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

July 27, 2017

Outline

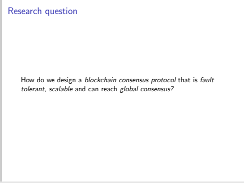
Outline

Motivation

- Blockchain systems offer an alternative to central authorities for the first time
- Market cap (40 billion for Bitcoin) and trade volume figures indicate they are here to stay
- Early blockchain systems are not scalable (7 TX/s for Bitcoin)
- Parameter tuning leads to centralisation

Research question

How do we design a *blockchain consensus protocol* that is *fault tolerant*, *scalable* and can reach *global consensus?*


Research question

How do we design a *blockchain consensus protocol* that is *fault tolerant*, *scalable* and can reach *global consensus?*

- Blockchain consensus protocol—independent of the application, like PoW (not PoS)
- Byzantine fault tolerance—if some number of nodes are malicious then the system should continue to function
- Horizontal scalability—if every node make transactions at the same rate, then the global throughput should increase
- Global consensus—all node should agree on a global state, useful for some applications, e.g. digital cash

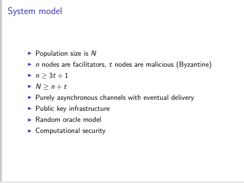
Inspiration

- A restaurant owner does not report all of its transactions with an central authority
- Ocassionally a customer may leave without paying and this event is reported to a central authority
- Our blockchain system achieves scalability using a similar idea

Outline

System model

- Population size is N
- n nodes are facilitators, t nodes are malicious (Byzantine)
- $n \geq 3t + 1$
- $N \geq n + t$
- Purely asynchronous channels with eventual delivery
- Public key infrastructure
- Random oracle model
- Computational security

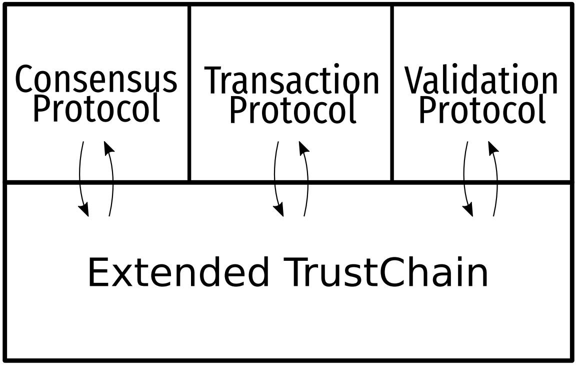


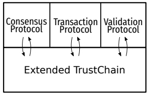
System model

- Population size is N
- n nodes are facilitators, t nodes are malicious (Byzantine)
- $n \geq 3t + 1$
- $N \geq n + t$
- Purely asynchronous channels with eventual delivery
- Public key infrastructure
- Random oracle model
- Computational security

At this point, explain facilitators are nodes that have some extra task to do, temporarily.

Architecture overview





- The primary data structure is the Extended TrustChain, extension of our prior work
- The three protocols the tasks as their name suggests
- They are independent and run concurrently
- The only synchronisation happens via the Extended TrustChain
- But in no part of those protocol do we lock the Extended TrustChain

Extended TrustChain

- Everyone has their own chain and genesis block
- Two types of blocks
  1. Transaction (TX) block
  2. Checkpoint (CP) block
- A transaction involves two parties and results in two TX blocks (a pair)
- A CP block captures the chain state
- TX and CP blocks are chained together using hash pointers

Extended TrustChain

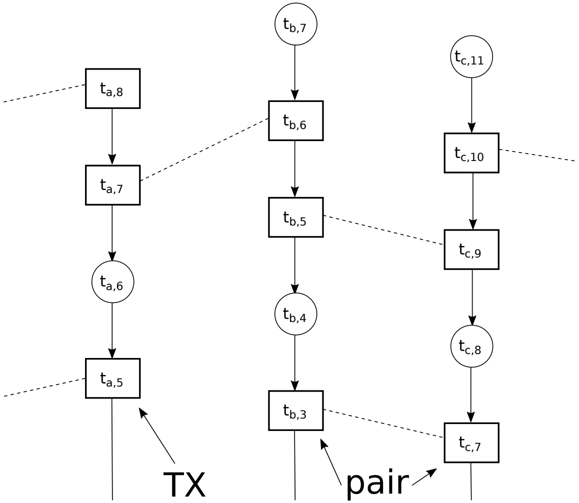


Figure: TX block is a six-tuple: $t_{u,i} = \langle \mathrm{H}(b_{u,i-1}), i, txid, pk_v, m, sig_u \rangle$.
CP block is a five-tuple: $c_{u,i} = \langle \mathrm{H}(b_{u,i-1}), i, \mathrm{H}(\mathcal{C}_r), r, sig_u \rangle$.

Extended TrustChain

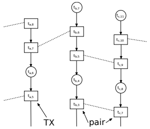


Figure: TX block is a six-tuple: $t_{u,i} = \langle \mathrm{H}(b_{u,i-1}), i, txid, pk_v, m, sig_u \rangle$.
CP block is a five-tuple: $c_{u,i} = \langle \mathrm{H}(b_{u,i-1}), i, \mathrm{H}(\mathcal{C}_r), r, sig_u \rangle$.

- In this example there are three nodes
- Squares are TX blocks and circles are CP blocks
- Explain the block content in caption
- The dotted line represent pairs of TX blocks
- Geared with the understanding of our data structure, we are ready to talk about the consensus protocol

Extended TrustChain: fragment definition

- A fragment $F_{u,i}$ is defined on a TX block $t_{u,i}$
- It is a section of the chain beginning and ending with CP blocks that contains the TX

Consensus protocol: overview

1. Runs in rounds
2. In round r, n out of N act as facilitators
3. The facilitators collect CP blocks from all nodes
4. Facilitators run a Byzantine consensus algorithm to agree on a set of CP blocks
5. Disseminate the consensus result $\mathcal{C}_r$, i.e. the CP blocks
6. From $\mathcal{C}_r$, new facilitators are computed
7. Repeat

Consensus protocol: properties

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

- *Agreement*: If one correct node outputs a set of facilitators $\mathcal{F}_r$, then every node outputs $\mathcal{F}_r$
- *Validity*: If any correct node outputs $\mathcal{F}_r$, then
  1. $|\mathcal{C}_r| \geq N - t$ must hold for the $\mathcal{C}_r$ which was used to create $\mathcal{F}_r$,
  2. $\mathcal{F}_r$ must contain at least $n - t$ honest nodes and
  3. $|\mathcal{F}_r| = n$.
- *Fairness*: Every node with a CP block in $\mathcal{C}_r$ should have an equal probability of becoming a member of $\mathcal{F}_r$.
- *Termination*: Every correct node eventually outputs some $\mathcal{F}_r$.

Consensus protocol

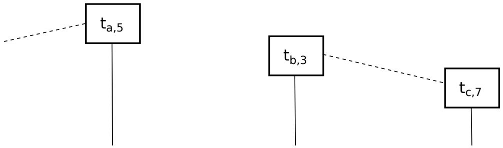


Figure: Suppose we are in a state where $\mathcal{C}_{r-1}$ has just been agreed by some facilitators but not yet propagated.

Consensus protocol

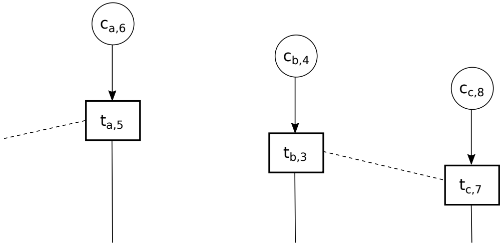


Figure: Nodes receive consensus result $\mathcal{C}_{r-1}$, first n nodes ordered by $H(\mathcal{C}_r||pk)$ become $\mathcal{F}_{r-1}$, send the new CP blocks to $\mathcal{F}_{r-1}$.

Consensus protocol

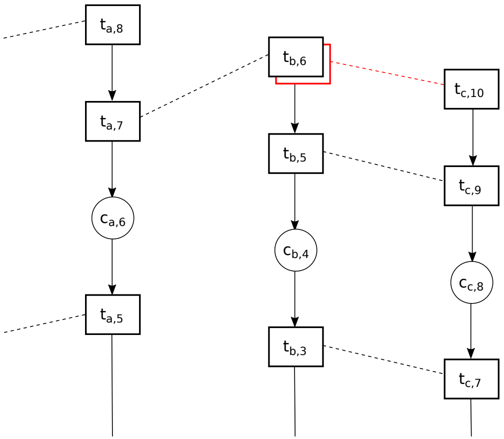


Figure: Transactions carry on as usual in round r, while facilitators are trying to reach consensus on the new CP blocks concurrently.

Consensus protocol

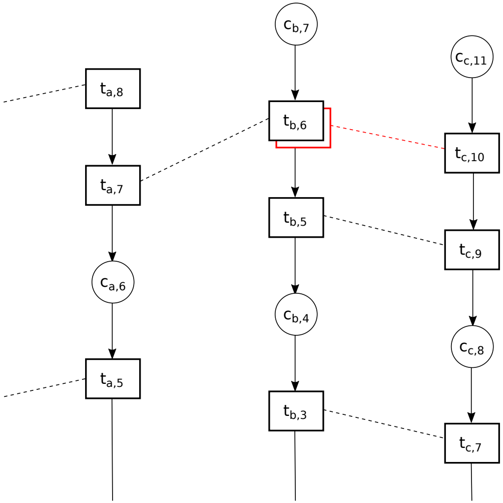


Figure: $\mathcal{F}_{r-1}$ agree and disseminate $\mathcal{C}_r$, CP blocks at round $r-1$ $(c_{a,6}, c_{b,4}, c_{c,8})$ should be in $\mathcal{C}_r$.

Transaction protocol

- Request (`tx_req`) and response (`tx_resp`) protocol
- Two TX blocks containing the same *txid* are generated
- Non-blocking

Transaction protocol

Create $t_{u,h}$ and send $\langle \texttt{tx\_req}, t_{u,h} \rangle$ to start a transaction.

```
**Upon** $\langle \texttt{tx\_req}, t_{v,j} \rangle$ from $v$
    $\langle \_, \_, txid, pk_v, m, \_ \rangle \leftarrow t_{v,j}$                ▷ unpack $t_{v,j}$
    new_tx($pk_u, m, txid$)                                   ▷ create and store $t_{u,i}$
    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 $v$
    $\langle \_, \_, txid, pk_v, m, \_ \rangle \leftarrow t_{v,j}$                ▷ unpack $t_{v,j}$
    store $t_{v,j}$ as the pair of the TX with identifier $txid$
```

Validation protocol: overview

- Request (`vd_req`) and response (`vd_resp`) protocol
- Transactions are in three states—*valid*, *invalid* and *unknown*—defined by get_validity($\cdot$)
- The goal is to identify which state a given transaction is in satisfying the validation protocol properties

Validity definition

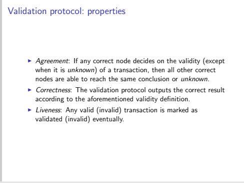
Function get_validity($t_{u,i}, F$) validates the transaction $t_{u,i}$

```
Check that $v$ sent the correct $F$, otherwise return *unknown*.

$\langle \_, \_, txid, pk_v, m, \_ \rangle \leftarrow t_{u,i}$
**if** number of blocks of $txid$ in $F \neq 1$ **then**
    **return** *invalid*
                                                        ▷ TX exists
$\langle \_, \_, txid', pk'_u, m', \_ \rangle \leftarrow t_{v,j}$
**if** $m \neq m' \vee pk_u \neq pk'_u$ **then**
    **return** *invalid*
                                                        ▷ no tampering
**return** *valid*
```

Validation protocol: properties

- *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*.
- *Correctness*: The validation protocol outputs the correct result according to the aforementioned validity definition.
- *Liveness*: Any valid (invalid) transaction is marked as validated (invalid) eventually.



Validation protocol: properties

- *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*.
- *Correctness*: The validation protocol outputs the correct result according to the aforementioned validity definition.
- *Liveness*: Any valid (invalid) transaction is marked as validated (invalid) eventually.

Liveness is not possible in our case because malicious nodes can simply go offline or refuse to respond, but the other two properties are possible.

Validation protocol

Send $\langle \texttt{vd\_req}, \mathit{txid} \rangle$ to v to begin.

```
**Upon** $\langle \texttt{vd\_req}, \mathit{txid} \rangle$ from $v$
    $t_{u,i} \leftarrow$ the transaction identified by $\mathit{txid}$
    $F_{u,i} \leftarrow$ agreed_fragment$(t_{u,i})$
    send $\langle \texttt{vd\_resp}, \mathit{txid}, F_{u,i} \rangle$ to $v$
**Upon** $\langle \texttt{vd\_resp}, \mathit{txid}, F_{v,j} \rangle$ from $v$
    $t_{u,i} \leftarrow$ the transaction identified by $\mathit{txid}$
    set the validity of $t_{u,i}$ to get_validity$(t_{u,i}, F_{v,j})$
```

Outline

Correctness of the consensus protocol

Theorem

For all rounds, the consensus protocol satisfies agreement, validity, fairness and termination.

Proof.

(sketch) Because consensus result are eventually delivered and the properties of Byzantine consensus, we get agreement, validity and termination. Fairness is from the fact that we model $\mathrm{H}(\cdot)$ as a random oracle (RO) and the input to the RO is different for every node, thus the list of nodes ordered by $\mathrm{H}(\mathcal{C}_r||pk)$ is a random permutation of those nodes. $\square$

Correctness of the validation protocol

Theorem

If any correct node decides on the validity (except when it is unknown) of a transaction according to the validity definition, then all other correct nodes are able to reach the same conclusion or unknown.

Proof.

(sketch) Proof by contradiction. For this attack to work, the adversary must be able to create two different fragments but with the same checkpoint enclosure. We model H(·) as a RO, so the adversary need to query the RO a exponential[1] number of times. But the adversary can only query the RO a polynomial number of times. □

[1] In terms of the security parameter.

Linear global throughput argument

- Throughput has a notion of time but our model is asynchronous
- Additional assumptions needed to make the argument—every unit of communication takes a non-negligible amount of time to process
- Bandwidth relation—$NC \geq r_{\text{tx}} I$, where I is $O(N)$
- If r_{tx} satisfies the inequality, then LHS and RHS grows at the same rate, thus we have linear global throughput

Outline

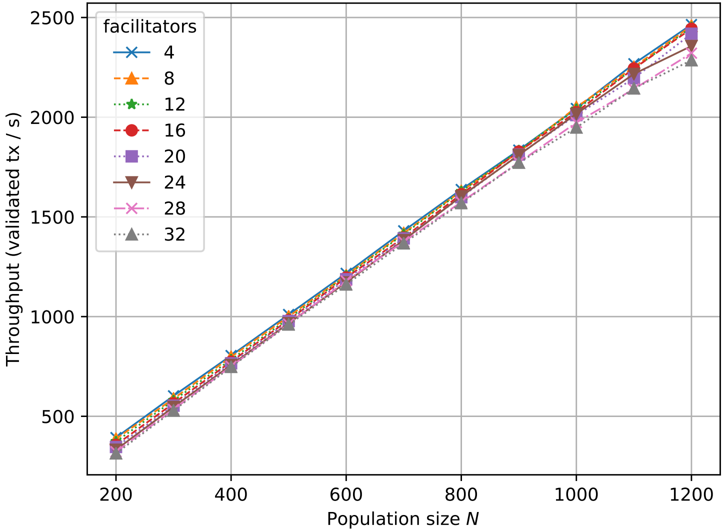
Implementation and experiment setup

- Prototype implementation on Github[2]
- SHA256 for hash functions and Ed25519 for digital signature
- Experiment on the DAS-5[3]
    - Up to 30 machines
    - Each running 40 nodes

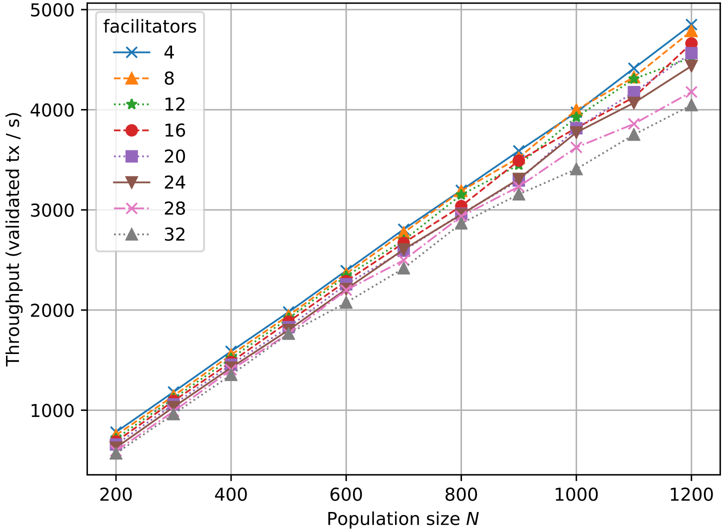
[2] `https://github.com/kc1212/consensus-thesis-code`

[3] `http://www.cs.vu.nl/das5/`

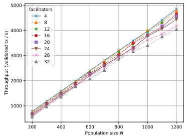
Throughput vs population size (random neighbour)



Throughput vs population size (fixed neighbour)



Throughput vs population size (fixed neighbour)



No need to request for fragment every time a TX needs to be validated.

Upon receiving a fragment, validate as many TX as possible.

Outline

Conclusion

How do we design a *blockchain consensus protocol* that is *fault tolerant*, *scalable* and can reach *global consensus?*

- ✓ Fault tolerant up to t nodes
- ✓ Horizontal scalability
- ✓ Global consensus

Future work

- Improve fault tolerance
- Improve fork detection
- Analyse the system in the permissionless environment
- Concrete application