# Blockchain Consensus Protocol with Horizontal Scalability

K. Cong

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

July 26, 2017

# Outline

# Outline

# Motivation

- Blockchain systems offer an alternative to central authorities for the first time
- Market cap 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?*

# Intuition

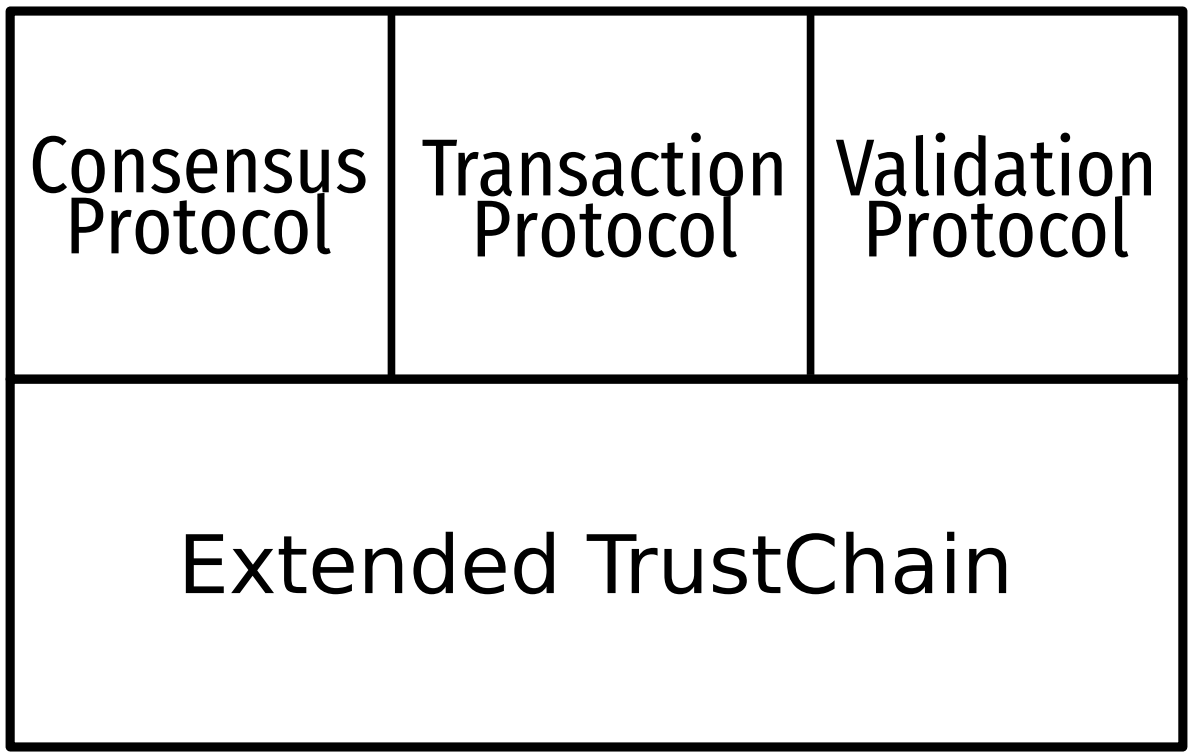
- A restaurant owner does not report all of its transactions with an central authority
- Ocassionally a customer may leave without paying and this event is reported to a central authority
- Our blockchain system achieves scalability using the same idea

# 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
- Application neutural

# Architecture overview



# Extended TrustChain

- Everyone has their own chain
- Two types of blocks, transaction (TX) blocks and checkpoint (CP) blocks
- A transaction involves two parties and results in two TX blocks
- A CP block captures the chain state
- TX and CP blocks are chained together using hash pointers

# Extended TrustChain

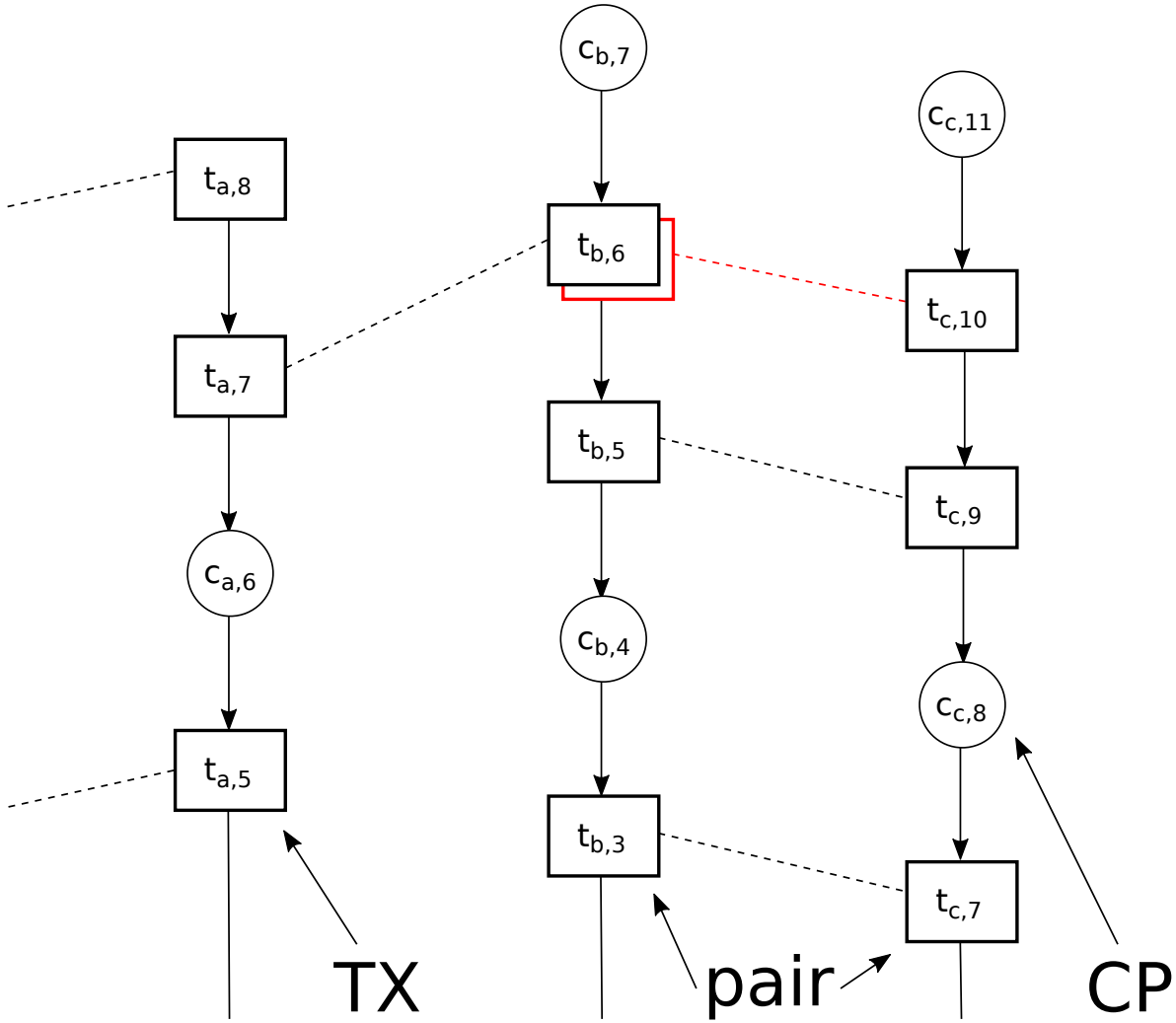



Figure: TX block is a six-tuple: $t_{u,i} = \langle \mathrm{H}(b_{u,i-1}), i, txid, pk_v, m, sig_u \rangle$.
CP block is a five-tuple: $c_{u,i} = \langle \mathrm{H}(b_{u,i-1}), i, \mathrm{H}(\mathcal{C}_r), r, sig_u \rangle$.

# Consensus protocoloverview

1. In round $r$, $n$ out of $N$ lucky nodes are selected at random to act as facilitators
2. Facilitators run a BFT (Byzantine Fault Tolerant) consensus algorithm to agree on a set of CP blocks
3. Disseminate the consensus result $\mathcal{C}_r$, i.e. the CP blocks
4. Repeat

# Consensus protocolproperties

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

- *Agreement*: If one correct node outputs a list 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

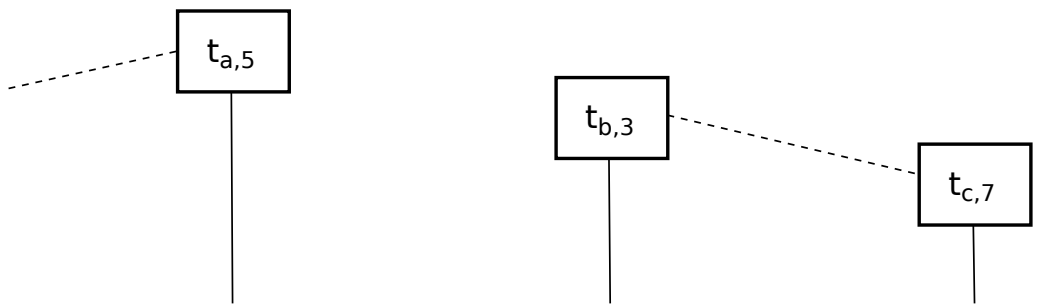



Figure: Suppose we are in a state where $\mathcal{C}_{r-1}$ has just been agreed by some facilitators but not yet propagated.

# Consensus protocol

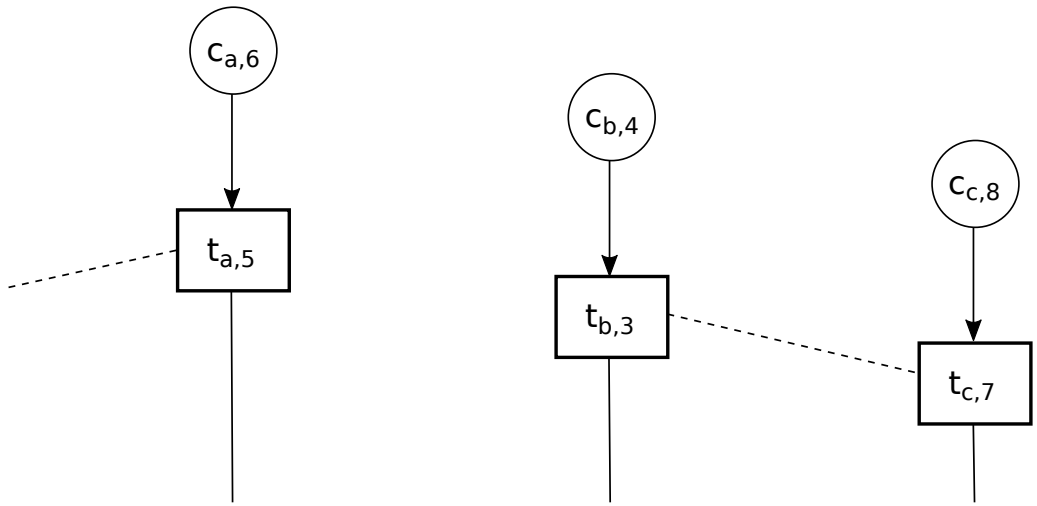



Figure: Nodes receive consensus result $\mathcal{C}_{r-1}$, compute the new facilitators $\mathcal{F}_{r-1}$, and send out the new CP blocks.

# Consensus protocol

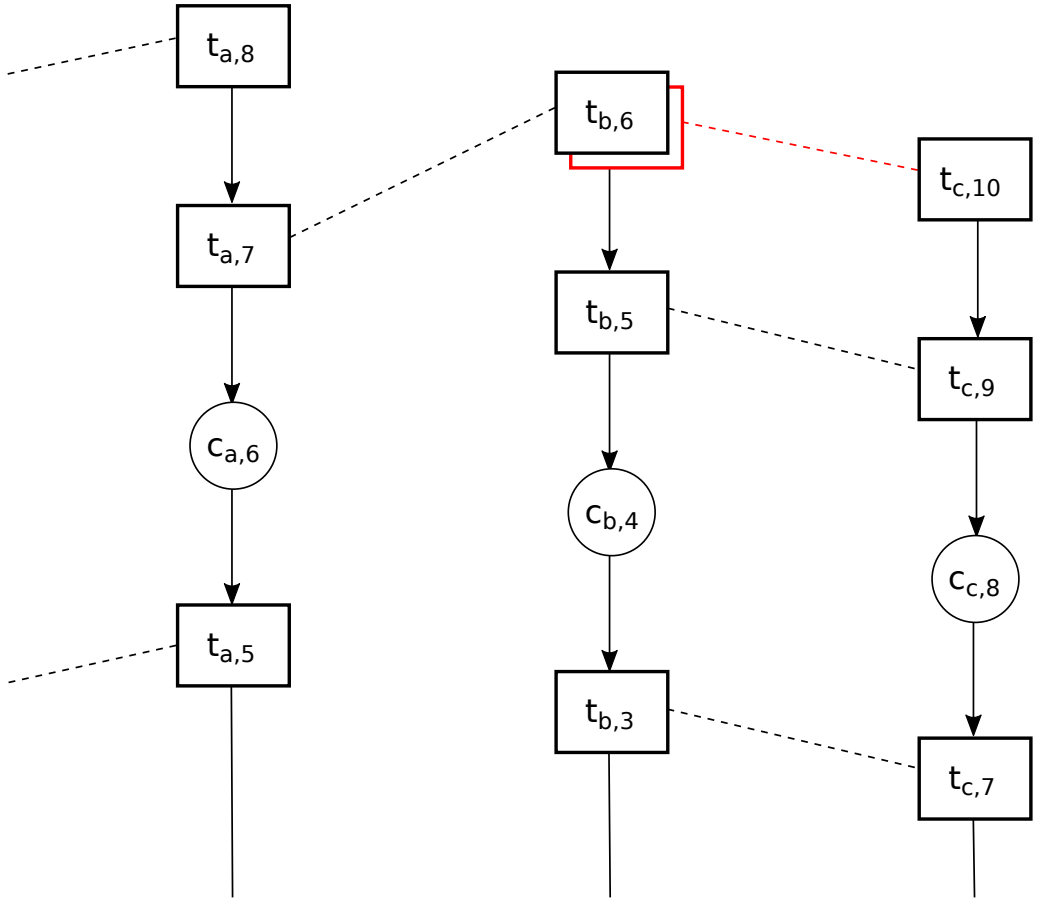



Figure: Transactions carry on as usual in round $r$, while facilitators are trying to reach consensus on the new CP blocks concurrently.

# Consensus protocol

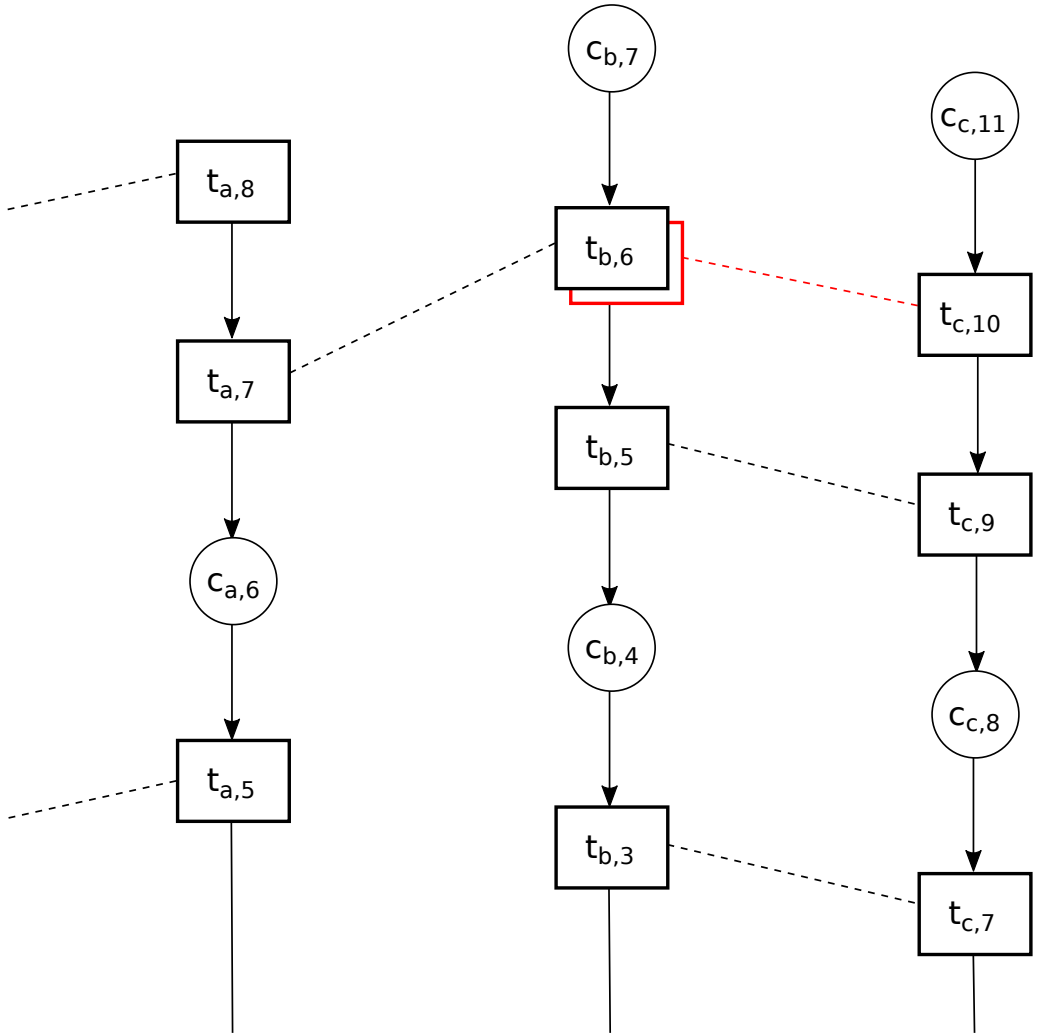



Figure: CP blocks at round $r-1$ should be in $\mathcal{C}_r$. The first $n$ nodes ordered by $\mathrm{H}(\mathcal{C}_r||pk)$ become $\mathcal{F}_r$.

# 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 protocoloverview

- Request (`vd_req`) and response (`vd_resp`) protocol
- Transactions are in three states—*valid*, *invalid* and *unknown*
- The goal is to identify which state a given transaction is in

# Validation protocolproperties

- *Correctness*: The validation protocol outputs the correct result according to the aforementioned validity definition.
- *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*.
- *Liveness*: Any valid transactions can be validated eventually.

# Validity definition

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

Check that $v$ sent the correct agreed fragment, otherwise 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*

▷ 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

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

**Upon** $\langle \texttt{vd\_req}, txid \rangle$ from $v$
  $t_{u,i} \leftarrow$ the transaction identified by $txid$
  $F_{u,i} \leftarrow \text{agreed\_fragment}(t_{u,i})$
  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 $txid$
  set the validity of $t_{u,i}$ to $\text{get\_validity}(t_{u,i}, F_{v,j})$

# 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 ACS, 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. □

# Correctness of the validation protocol

## Theorem

*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.*

## 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 $H(\cdot)$ 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—$NC \geq r_{\text{tx}} l$, where $l$ is $O(N)$
- If $r_{\text{tx}}$ satisfies the inequality, then LHS and RHS grows at the same rate, thus we have linear global throughput

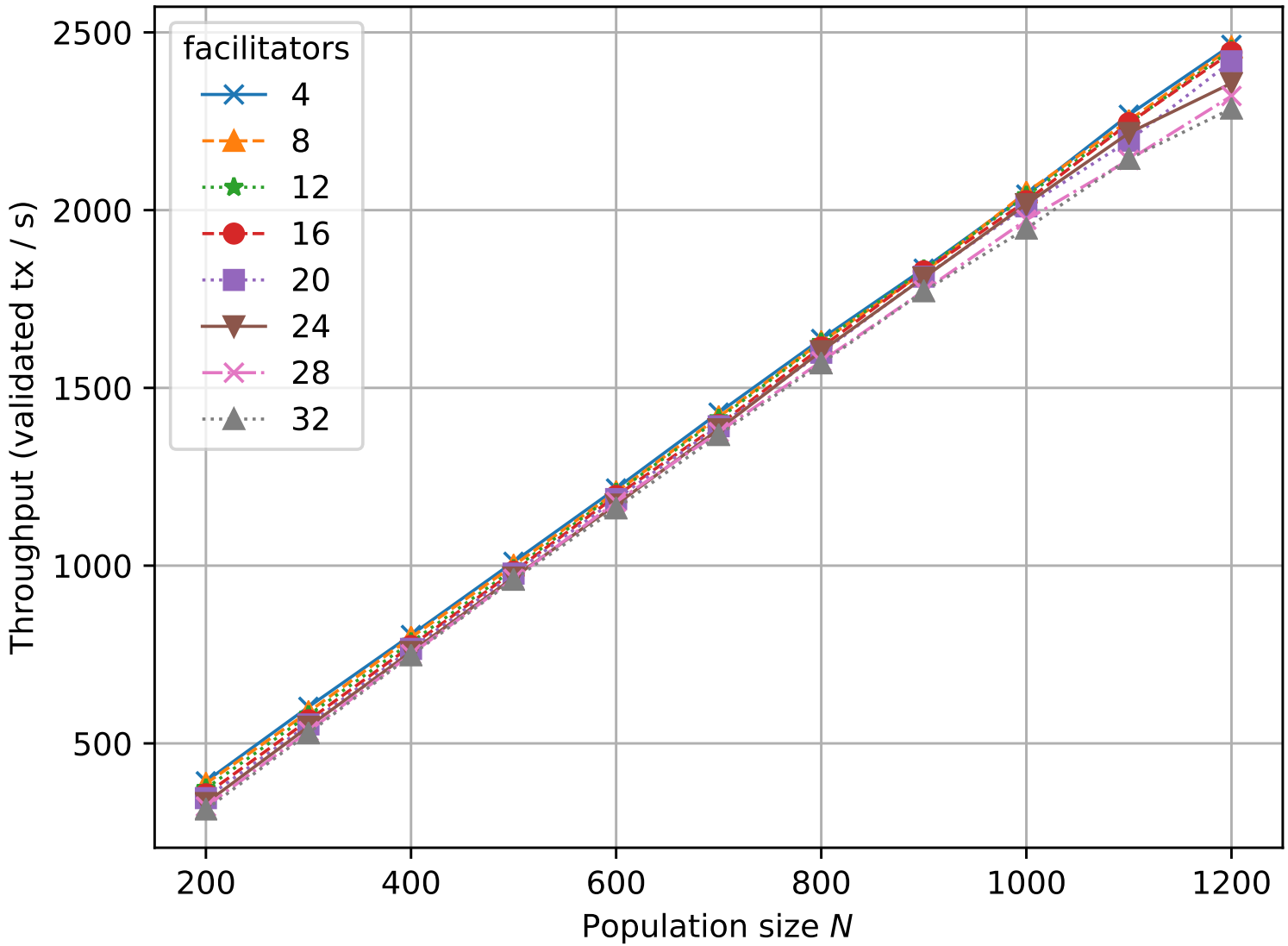
# Outline

# Implementation and experiment setup

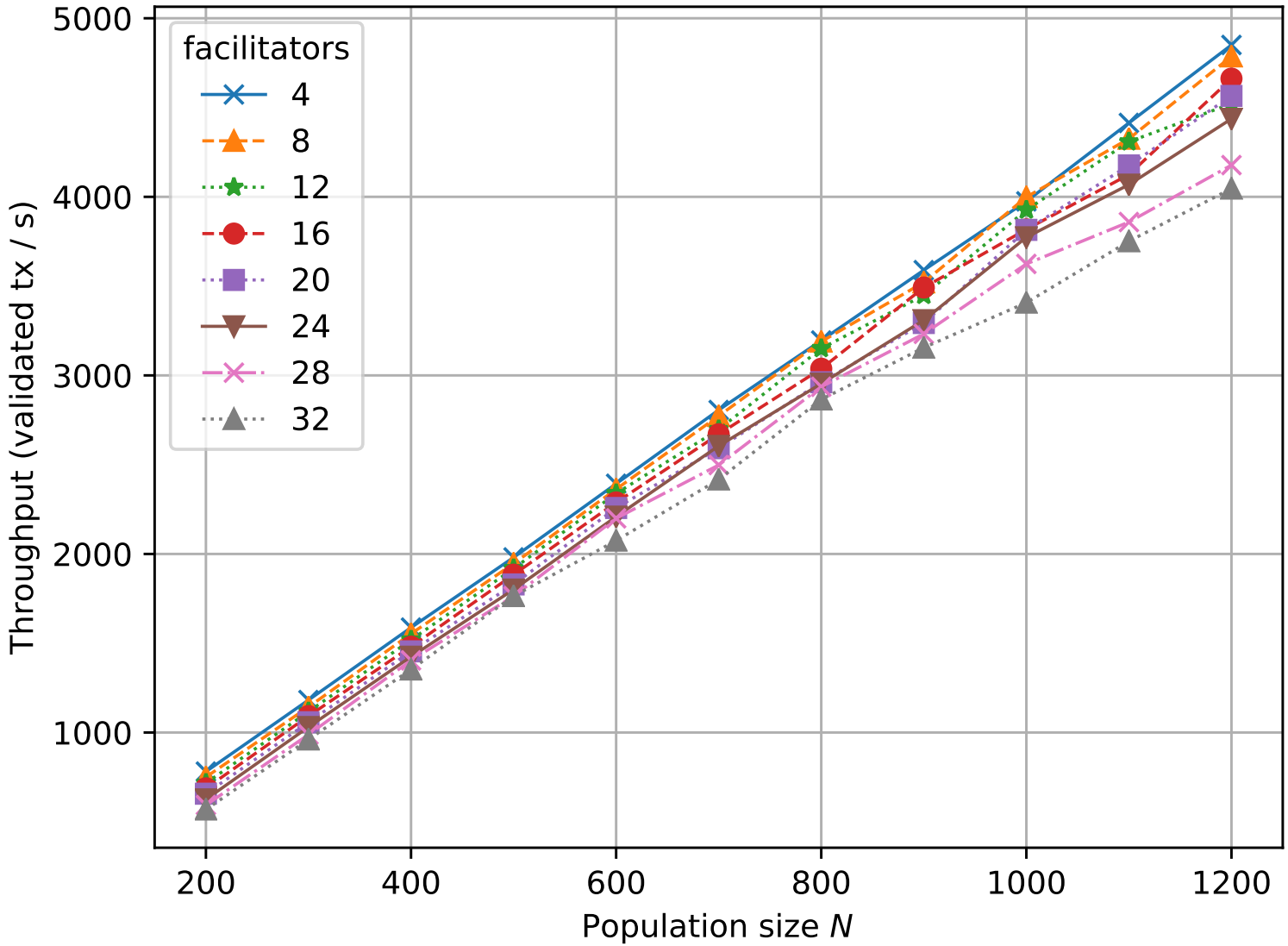
- Prototype implementation on Github[2]
- SHA256 for hash functions and Ed25519 for digital signature
- Experiment on the DAS-5[3]

[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)



# 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