A Blockchain Consensus Protocol With Horizontal Scalability

Kelong Cong
kelong.cong@epfl.ch
EPFL

Zhijie Ren
z.ren@tudelft.nl
TU Delft

Johan Pouwelse
peer2peer@gmail.com
TU Delft

IFIP Networking, Zurich 2018

Outline

Outline

The dangers of centralisation

- Technological advancements give us convenience
- But it puts central authorities in control
- Many are motivated by profit or the goals of the local government
- The interest of the authorities do not align with the users

Blockchain: a new hope?

- Blockchains are distributed (replicated) ledgers with no central control
- They enable internet-scale consensus for the first time
- Some initial applications include:
  - Digital cash (e.g., Bitcoin, Litecoin)
  - Domain name system (e.g., Namecoin)
  - Storage rental (e.g., Filecoin)
  - General purpose (e.g., Ethereum)

Blockchain: not there yet

- All blockchain systems have a consensus algorithm
- Early consensus algorithms (PoW) do not scale
- Bitcoin is limited to 7 transactions per second
- 100,000 transaction backlog in May 2017
- We require horizontal scalability for ubiquitous use
- More users → more transactions per second globally

Related work

- Off-chain solution
  - Lightning Network[1]
  - Perun[2]
- On-chain solution
  - Parameter tuning
  - BFT consensus (e.g. Tendermint[3], ByzCoin[4])
  - Sharding (e.g. Elastico[5], OmniLedger[6])

[1] https://lightning.network/
[2] https://perun.network
[3] https://www.tendermint.com/
[4] KJGKGF, USENIXSecurity16
[5] LNZBGS, CCS16
[6] KJGGSF, S&P18

Related work

State-of-the-art—Sharding:

- Split state into multiple shards
- Shards run consensus algorithm in parallel

Challenges:

- Choosing and evolving the shard size
- Perform atomic inter-shard transactions
- Parameter choice highly depends on the application

Research question

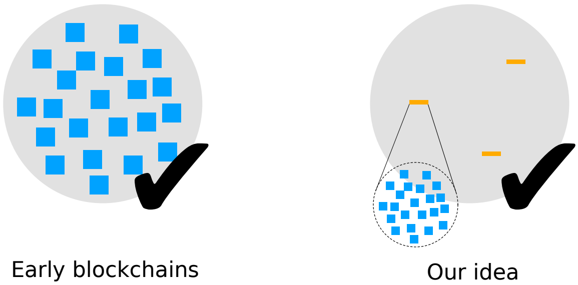
How can we design a *blockchain consensus protocol* that is *fault-tolerant*, *horizontally scalable*, and able to reach *global consensus?*

- Blockchain consensus protocol—application neutral, e.g., PoW
- Fault-tolerant—tolerate a number of malicious nodes
- Horizontal scalability—more nodes in the network leads to higher transaction throughput
- Global consensus—all node should agree on a global state

Outline

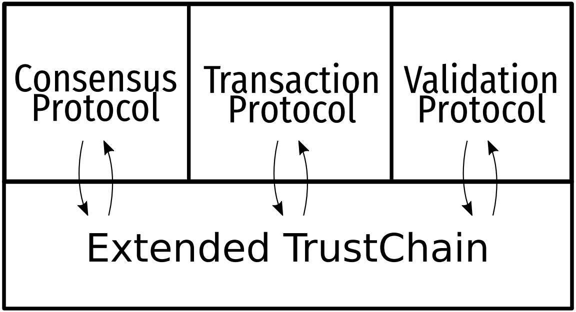
Intuition and idea explored in this thesis

- It is expensive to verify and reach consensus on all transactions
- Our idea: we decouple consensus and validation
- A single digest represents an arbitrarily large number of transactions
- Reach consensus on the small digest
- Nodes then independently check the validity of the transactions of interest

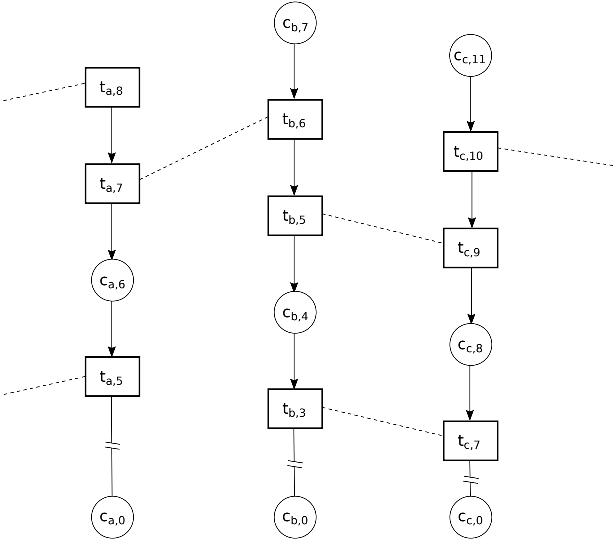


Architecture overview

The four components of CHECO

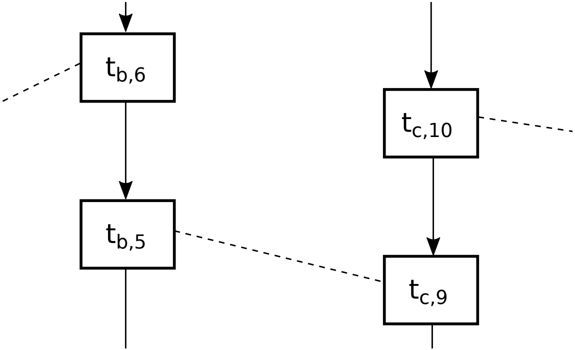


Extended TrustChain



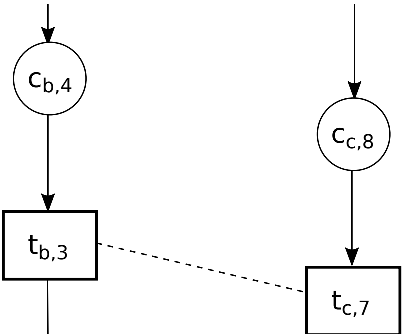
Extended TrustChain: Transaction (TX) block

- Goal: record transactions
- A transaction is represented by a pair of TX blocks, i.e. a contract signed by both parties

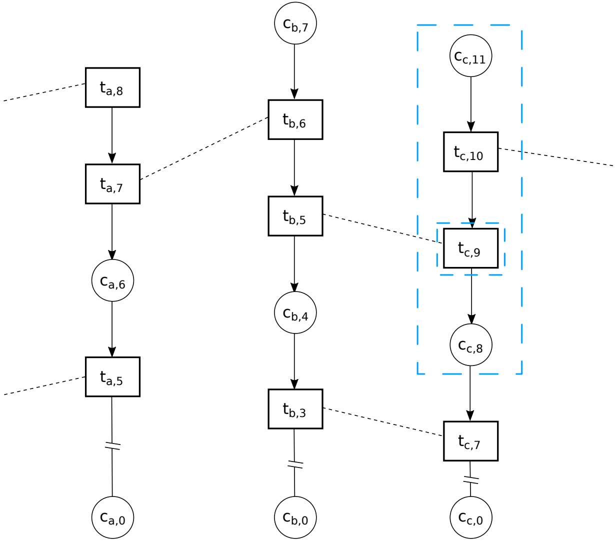


Extended TrustChain: Checkpoint (CP) block

- Goal: represent the state of the chain using a single digest
- A collection of CP blocks from all the nodes represent the state of the system
- Nodes become aware of the system state by running our consensus protocol



Extended TrustChain: Fragment of a TX block

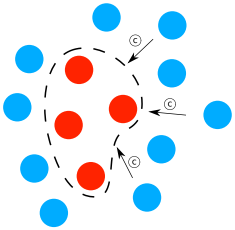


Consensus protocol

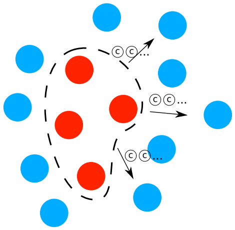
- Goal 1: reach consensus on a collection of CP blocks amongst all the nodes
- Goal 2: create new CP blocks at the end of the protocol
- Uses an existing fault-tolerant consensus algorithm (HoneyBadgerBFT[7]) as the building block
- But it cannot be used in a large network due to high communication complexity
- We overcome this limitation by selecting a small number of *facilitators* from the network to run HoneyBadgerBFT

[7] MXCSS, CCS16

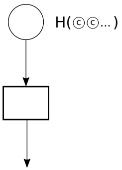
Consensus protocol



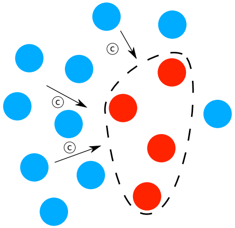
collect CP blocks

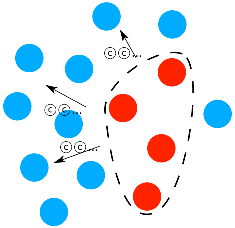


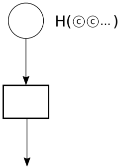
disseminate result



create new CP block
elect new facilitators



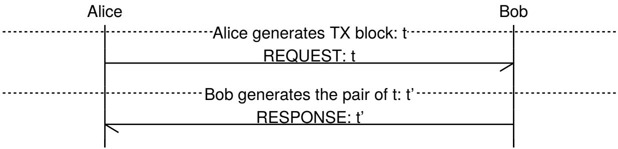




Consensus protocol: properties

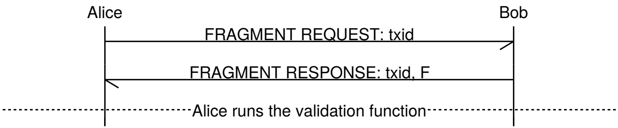
- *Agreement*: Every correct outputs the same set of facilitators.
- *Validity*: The consensus result is valid such that a new set of facilitators can be computed from it.
- *Termination*: Every correct node eventually outputs a set of facilitators.

Transaction protocol



- Two TX blocks are generated on the chains of Alice and Bob
- No guarantee that nodes follow this protocol

Validation protocol



- To check that the transaction protocol is correctly followed
- Alice needs the fragment of the TX on Bob's hash chain
- Validation function checks whether the fragment is OK and contain the transaction
- Can be generalised—any node may run the validation protocol on any transaction (does not need to be their own)

Validation protocol: properties

Consensus on CP blocks → consensus on transactions

- CP blocks of the fragments are "anchored" due to the consensus protocol
- It is difficult to modify the fragment once "anchored"
- Since the transaction protocol and the validation protocol only use point-to-point communication, we achieve horizontal scalability.

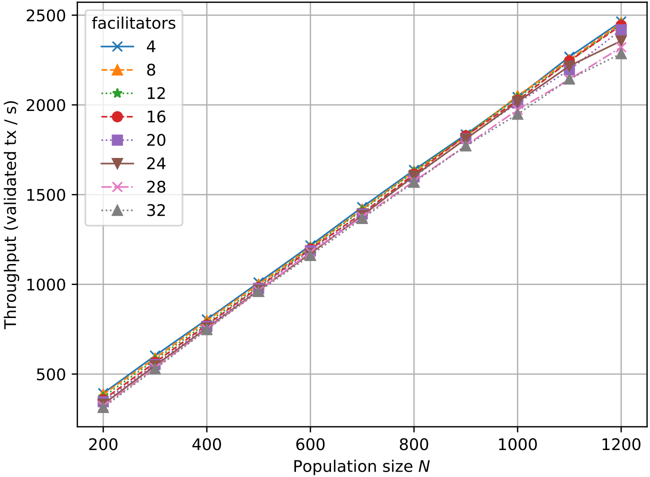
Outline

Implementation and experiment setup

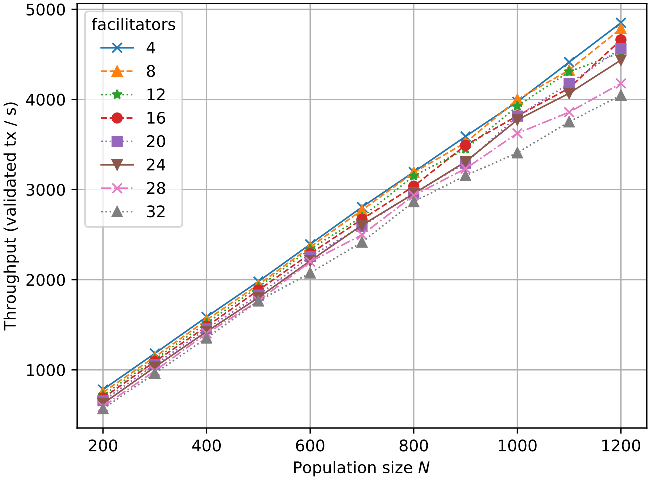
- Free and open source implementation on Github: `https://github.com/kc1212/checo`
- SHA256 for hash functions and Ed25519 for digital signature
- Experiment on the DAS-5[8]
- Up to 1200 nodes

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

Validated transaction throughput (random node)



Validated transaction throughput (fixed neighbour)



Outline

Conclusion

Our work answers the research question.

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

- Fault-tolerance is achieved using HoneyBadgerBFT
- Horizontal-scalability is achieved by separating consensus and validation, demonstrated experimentally
- Global-consensus on transactions is achieved via consensus on CP blocks

Thank you

Any questions?

TX block

1. Hash pointer to the previous block
2. Sequence number
3. Transaction ID
4. Public key of the counterparty
5. Transaction message m
6. Signature the five items above

A transaction is represented by a *pair* of TX blocks

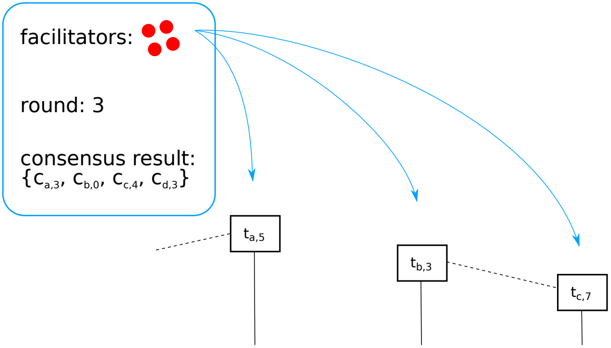
CP block

1. Hash pointer to the previous block
2. Sequence number
3. Digest of consensus result, i.e. a set of CP blocks
4. Round number r
5. Signature on the four items above

Background on ACS

- Asynchronous common subset
- A simplification of HoneyBadgerBFT
- n nodes
- t nodes may be malicious
- Input: every node proposes a set of values, e.g., $\{A, B\}, \{B, C\}, \ldots$
- Output: set union of the majority, e.g., $\{A, B, C, \ldots\}$

Consensus protocol: part 1

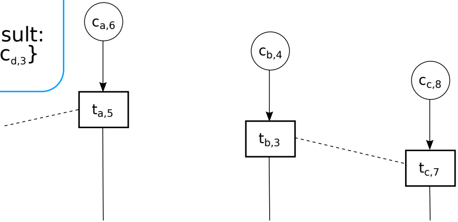


Consensus protocol: part 2

facilitators:

round: 3

consensus result:
$\{c_{a,3}, c_{b,0}, c_{c,4}, c_{d,3}\}$

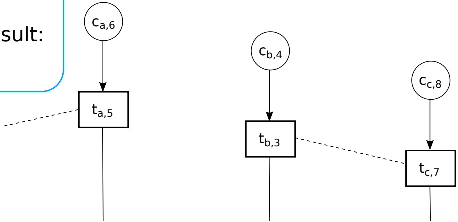


Consensus protocol: part 3

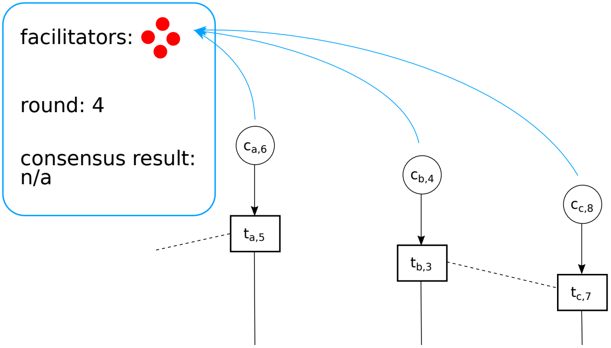
facilitators: lottery
$\{c_{a,3}, c_{b,0}, c_{c,4}, c_{d,3}\}$

round: 4

consensus result:
n/a



Consensus protocol: part 4

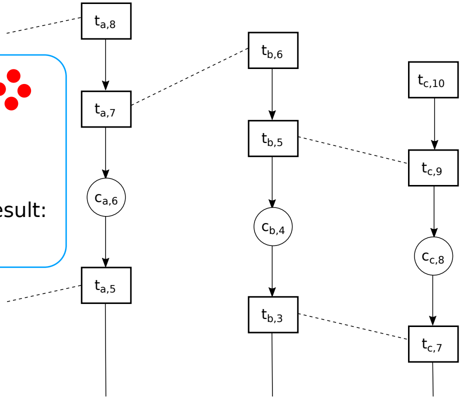


Consensus protocol: part 5

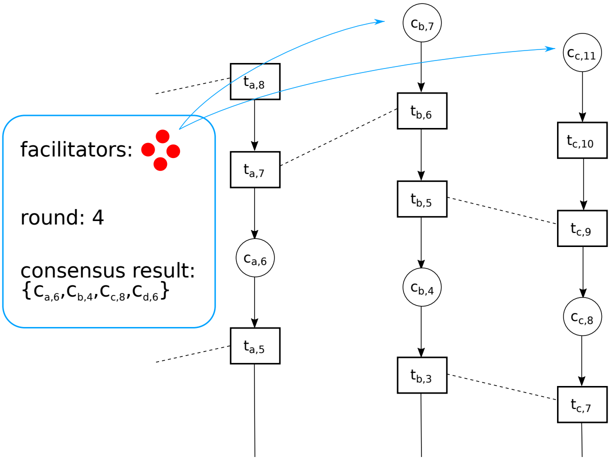
facilitators:

round: 4

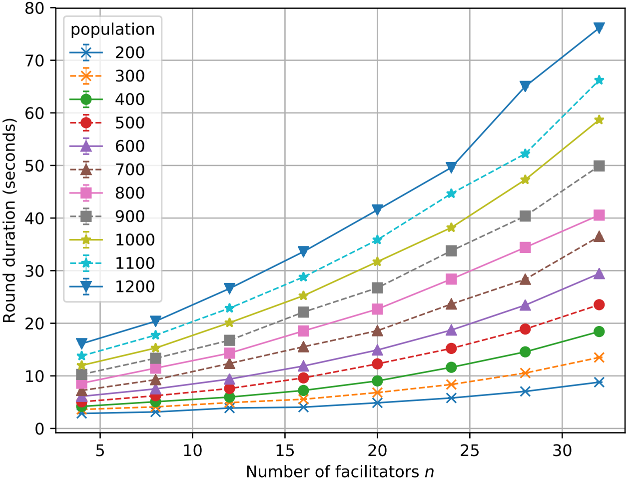
consensus result: n/a



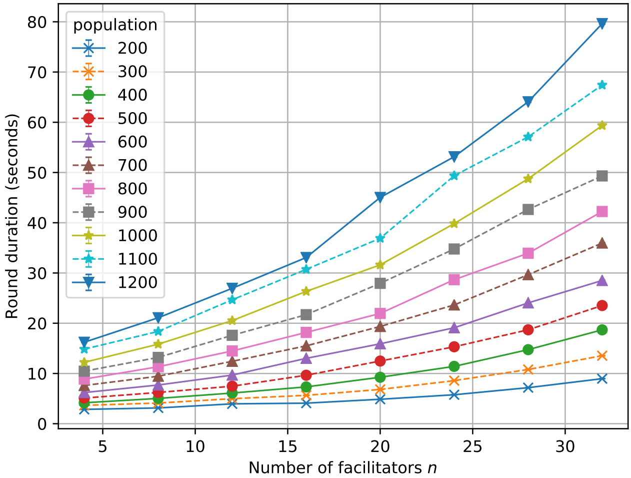
Consensus protocol: part 6



Effect of the number of facilitators (fixed neighbours)



Effect of the number of facilitators (random neighbours)



Future work

- Implement and experiment with a concrete application
- Analyse the system in the permissionless environment
- Improve fault tolerance

Stress test (fixed neighbour)