
Faculty Electrical Engineering, Mathematics and Computer Science

# TODO title

*TODO subtitle*

Kelong Cong

Supervisors:
Dr. J. Pouwelse
Dr. Unknown
Prof. Unknown

Delft, Month 2017

# Chapter 1

# Checkpoint Consensus

## 1.1 Preliminaries

### 1.1.1 Requirements

- Permissionless
- Byzantine fault tolerant
- No PoW
- Works under churn
- Underlying data structure is TrustChain
- Detects forks or double-spends
- No step in the protocol blocks transactions
- Application independent

### 1.1.2 Assumptions

- Asynchronous network
- Private and authenticated channel
- We elect $N$ consensus promoters in every round, we assume the number of faulty promoters is $f$ and $N = 3f + 1$.
- Promoters have the complete history of the previously agreed set of transactions (TX).

### 1.1.3 Notation and definition

- $y = \mathtt{h}(x)$ is a cryptographically secure hash function (random oracle), the domain $x$ is infinite and the range is $y \in \{0, 2^{256} - 1\}$.
- We model our system in the permissionless setting, where each participating party has a unique identity $i$, and a chain $B_i$.
- A chain is a collection of blocks $B_i = \{b_{i,j} : j \in \{1 \dots h\}\}$, the blocks are linked together using hash pointers, similar to bitcoin. All blocks contain a reference to the previous block, the very first block with no references is the genesis block.

  The sequence number of the block begins at 0 on the genesis block and is incremented for every new block. The height of the chain is $h = |B_i|$.

- There are two sets of blocks $T_i$ and $C_i$, where $T_i \cup C_i = B_i$ and $T_i \cap C_i = \emptyset$. This can be seen as the block type, where $t_{i,j}$ and $c_{i,j}$ to represent a *transaction block (TX block)* and *checkpoint block (CP block)* respectively.

- $\texttt{typeof} : B_i \to \{\tau, \gamma\}$ returns the corresponding type of the block.

- A block of type $\tau$ is a six-tuple, i.e.

$$t_{i,j} = (\texttt{h}(b_{i,j-1}), h_s, h_r, s_s, s_r, m).$$

  $h_s$ and $h_r$ denote the height (the sequence number for when the TX is made) of the sender and receiver respectively. $s_s$ and $s_r$ are the signatures of the sender and the receiver respectively. $i = \{s, r\}$ and $j = \{h_s, h_r\}$ depending on whether $i$ is the sender or the receiver.

- Given two TX blocks $t_{i,j}$ and $t_{i',j'}$, but $i \neq i'$. If $h_s = h'_s$, $h_r = h'_r$, $m = m'$ and the signatures are valid, then we call them a *pair*.

- If there exist two TX blocks $t_{i,j}$ and $t_{i',j'}$, where $s_s$ and $s'_s$ is created by the same public key, $h_s = h'_s$, but $i \neq i'$, then we call this a *fork*.

- A block of type $\gamma$ is a three-tuple, i.e.

$$c_{i,j} = (\texttt{h}(b_{i,j}), H(\mathcal{C}_r), p)$$

  where $\mathcal{C}_r$ is the consensus result in round $r$ and $p \in 0, 1$ which indicates whether $i$ wish to become a promoter in the following consensus round.

- We define

$$\texttt{newtx} : B_i \times B_j \times M \to B_i \times B_j$$

  as a function that creates new TX blocks. Its functionality is to extend the given chains using the following rule. If the input is $(B_s, B_r, m)$ then the output is $(B'_s, B'_r)$ where $B'_s = \{(\texttt{h}(b_{s,h_s}), h_s + 1, h_r + 1, s_s, s_r, m)\} \cup B_s$, $B'_r = \{(\texttt{h}(b_{r,h_r}), h_s + 1, h_r + 1, s_s, s_r, m)\} \cup B_r$, $h_s = |B_s|$ and $h_r = |B_r|$.

- We define

$$\texttt{newcp} : B_i \times \mathbb{R}_{\geq 1} \times \{0, 1\} \to B_i$$

  as a function that creates new CP blocks. Concretely, given $(B_i, r, p)$, it results in $\{(\texttt{h}(b_{i,h}), H(\mathcal{C}_r), p)\} \cup B_i$ where $h = |B_i|$, and $\mathcal{C}_r$ is the latest consensus result at round $r$.

- Note that in the actual system, `newtx` and `newcp` perform a state transition.

- We define

$$\texttt{getr} : C_i \to \mathbb{R}_{\geq 1}$$

  as a function that gets the consensus round number used to create the given CP block.

- The CP blocks that follows a pair of TX blocks must be created using the same $\mathcal{C}_r$, otherwise the transaction is invalid. Concretely, given a pair $t_{i,j}$ and $t_{i',j'}$, then there exist $c_{i,k}$ and $c_{i',k'}$ where $j < k$, $j' < k'$, $\{\texttt{typeof}(b_{i,x}) : x \in \{j, \dots k-1\}\}$ are all $\tau$, $\{\texttt{typeof}(b_{i',x}) : x \in \{j', \dots k'-1\}\}$ are all $\tau$ and $\texttt{getr}(c_{i,k}) = \texttt{getr}(c_{i',k'})$. This constraint is not the result of the consensus protocol, but the validation protocol.

- The result of a consensus in round $r$ is a set of two-tuple of CP. Namely, $\mathcal{C}_r = \{(c_{i,j}, c_{i,k}) : j < k, \text{agreed by the promoters}\}$.

### 1.1.4 Properties

- A TX block has two *ancestor* blocks. Let a pair be $t_{i,j}$ and $t_{i',j'}$, then $(b_{i,j-1}, b_{i',j'-1})$ is the input block of $t_{i,j}$.
- A CP block has one *ancestor* block, that is simply the block with the previous sequence number, i.e. the ancestor of $c_{i,j}$ is $b_{i,j-1}$.
- Every TX block $t_{i,j}$ is enclosed by two CP blocks $(c_{i,a}, c_{i,b})$, where $a = \arg\min_{k,k<j,\texttt{typeof}(b_{i,k})=\gamma}(j-k)$ and $b = \arg\min_{k,k>j,\texttt{typeof}(b_{i,k})=\gamma}(k-j)$.
- The blocks between two $\gamma$ blocks $(c_{i,a}, c_{i,b})$ is $\{b_{i,j} : b_{i,j} \in B_i, a \leq j \leq b\}$.

## 1.2 Checkpoint consensus

### 1.2.1 Luck value

First we define the luck value $l_{i,j} = \texttt{h}(k_i||c_{i,j})$, where $k$ is the public key of $i$. A lower luck value equates to higher luck.

### 1.2.2 Promoter registration

Node $i$ can register as a promoter when the latest consensus result is announced (suppose after the completion of round $r-1$), then it generates a new block $b_{i,j} = \texttt{newcp}(B_i, r-1, 1)$. The current promoters (in round $r$) may decide to include $b_{i,j}$ in the new consensus result. If $b$ is in it, then $i$ becomes one of the promoter of round $r+1$.

We can fix the number of promoters to $N$ by sorting the promoters by their "luck value"' and taking the first $N$.

### 1.2.3 Promoter invitation

The output of promoter registration is a random set of $N$ promoters. If 1/3 of the population is malicious, then we cannot guarantee that the chosen promoters satisfy the $< n/3$ requirement.

Promoter invitation is an attempt to involve human in the protocol. A naive method method is to use $N$ tickets, and then distribute them to trusted nodes. Nodes with the ticket can forward it to others. We have to rely on the humans to always forward the tickets to other honest humans. Finally, the nodes that hold a ticket are promoters. The result is that we have a permissioned system.

### 1.2.4 Setup phase

The setup phase should satisfy the BFT conditions regarding the promoter selection, that is:

1. *Agreement*: If any correct node outputs a promoter $p$, then every correct node outputs $p$.
2. *Total Order*: If one correct node outputs the sequence of promoters $\{p_1, p_i, \ldots, p_n\}$ and another has output $\{p'_0, p'_1, \ldots, p'_{n'}\}$, then $p_i = p'_i$ for $i \leq \min(n, n')$.
3. *Liveness*: All $N - f$ correct nodes terminate eventually.

We begin in the state where $\mathcal{C}_{r-1}$ has just been agreed but has not been disseminated yet. The exact technique to disseminate $\mathcal{C}_{r-1}$ is irrelevant, broadcasting or gossiping are both sufficient. In fact, dissemination is not necessary, nodes interested in the result can simply query the promoters that created $\mathcal{C}_{r-1}$.

**Lemma 1.** *If a node sees a valid $\mathcal{C}_{r-1}$ and another node sees a valid $\mathcal{C}'_{r-1}$, then $\mathcal{C}_{r-1} = \mathcal{C}'_{r-1}$.*

*Proof.* (sketch) $\mathcal{C}_{r-1}$ is signed by at least $N - f$ promoters from round $r-1$. □

The potential promoters now need to first discover whether they are the first $N$ lucky promoters.

**Lemma 2.** *The new set of promoter for the next consensus round is consistent with respect to all the nodes in the network.*

*Proof.* (sketch) All nodes use the same deterministic function to compute the luck value. □

Nodes should wait for some time to collect the CP blocks, so they wait for some time $\Delta$ before moving on to the next phase.

**Lemma 3.** *Promoters waiting for a some time $\Delta$ to collect transactions does not violate the asynchronous assumption.*

*Proof.* (sketch) This can be seen as a long delay in the asynchronous system. □

**Corollary 1.** *The setup phase satisfies the validity, agreement and liveness properties.*

*Proof.* (sketch) Validity and agreement is satisfied because the promoters are computed using a deterministic algorithm using the latest consensus result. Liveness is satisfied because $\mathcal{C}_{r-1}$ is eventually propagated to all node by gossiping. □

### 1.2.5 Consensus phase

The consensus phase should satisfy the BFT conditions regarding the CP blocks, that is:

1. *Agreement*: If any correct node outputs a CP block $c$, then every correct node outputs $c$.
2. *Total Order*: If one correct node outputs the sequence of CP blocks $\{c_1, c_i, \ldots, c_n\}$ and another has output $\{c'_0, c'_1, \ldots, c'_{n'}\}$, then $c_i = c'_i$ for $i \leq \min(n, n')$.
3. *Liveness*: All $N - f$ correct nodes terminate eventually.

We need an atomic broadcast algorithm for the consensus phase. We use a similar but simplified construction as [3], where atomic broadcast is constructed from the reliable broadcast[1] [1] and asynchronous common subset (ACS). The ACS protocol requires a binary Byzantine agreement protocol, and for that it needs a trusted dealer to distributed the secret shares. Promoters can check whether the secret shares are valid, but they cannot prevent the dealer from disclosing the secrets.

There techniques that uses no dealers. First is to use PBFT [2], but we must change our asynchronous assumption into the weak synchrony assumption. Most likely this is not possible because of the "When to start?" problem. It's difficult to give a bounded delay for propagating the consensus result.

Second is to use an inefficient binary Byzantine agreement protocol where its message complexity is $O(N^3)$ rather than $O(N^2)$ and becomes a bottleneck. Or a suboptimal one, e.g. $n/5$ instead of $n/3$.

Suppose we use a dealer, what is the effect to the algorithm if the dealer is malicious?

## 1.3 Fraud detection

Fraud detection is application specific. In the context of electronic cash, a fraudulent transaction will have an effect in subsequent transactions. In the context of P2P file sharing, the transactions are reports of the number of bytes uploaded and/or downloaded, thus they are independent of each other.

We focus in the latter case, where trasactions are independent.

[1] Reliable broadcast solves the Byzantine generals problem.

### 1.3.1 Threat model

- What is the influence of a fork for future transactions?
- What if both parties (sender and receiver) are malicious?

### 1.3.2 Existence testing

Suppose node $i$ created a TX $t_{i,j}$ and its pair is $t_{i',j'}$. Our protocol states that there exist two CP block $c_{i,k}$ and $c_{i',k'}$ such that $k > j$ and $k' > j'$, but all blocks between $(k, j)$ and $(k', j')$ are of type $\tau$. To verify whether $t_{i',j'}$ exists, $i$ performs the following tasks:

1. Waits for $c_{i',k'}$ to be in consensus.
2. Finds $c_{i',k''}$ such that $k'' < j' < k'$ and $k''$ is also in consensus (in some previous consensus round).
3. Asks $i'$ for $\{b_{i',x} : x \in \{k'', \dots k'\}\}$, and verified that all the blocks follow the rules of TrustChain and contains $t_{i',j'}$.

Node $i$ may wish to perform existence testing for all its unverified TX blocks.

### 1.3.3 Random sampling

Existence testing detects fraud when a single party of the TX is malicious. To detect fraud when both parties are malicious, every node performs existence testing on all TX blocks between CP blocks as if they are some other node. The assignment is determined using CP block digest where every node is assigned to the nearest node that has a higher digest, cycle back if there is no higher digest.

If the number of malicious parties is low, then it's more probable that a fork is detected. If where there are a large number of malicious parties, we repeat the random sampling process.

# Bibliography

[1] Gabriel Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162. ACM, 1984. v

[2] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999. v

[3] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42. ACM, 2016. v