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

Introduction

This is a story about a scalable blockchain framework and Byzantine generals...

Chapter 2

Checkpoint Consensus

2.1 Preliminaries

2.1.1 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

2.1.2 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).

2.1.3 Notation, definition and properties

- $y = \mathtt{h}(x)$ 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 τ 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 $t_{i',j'}$, if $i \neq i'$, $i = s$, $i' = r$, $h_s = h'_s$, $h_r = h'_r$, $m = m'$ and the signatures are valid, then we call them a *pair*. Note that given one TX block, the pair can be determined directly.
- 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 γ 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

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

  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 τ, $\{\texttt{typeof}(b_{i',x}) : x \in \{j', \dots k'-1\}\}$ are all τ and $\texttt{round}(c_{i,k}) = \texttt{round}(c_{i',k'})$. 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 CP blocks. Namely,

  $$\mathcal{C}_r = \{c_{i,j} : \text{agreed by the promoters}\}.$$

- A TX block can be valid, invalid or unknown. All TX blocks begin as unknown, they can be validated using our validation protocol.

- 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

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

$$b = \underset{k,k>j,\texttt{typeof}(b_{i,k})=\gamma}{\arg\min} (k-j).$$

  We call this the *enclosure* of $t_{i,j}$.
- The *piece* of $t_{i,j}$ defined by the enclosure $(c_{i,a}, c_{i,b})$ is $\{b_{i,j} : a \leq j \leq b\}$. We define

$$\texttt{pieces} : B_i \to P(B_i)$$

  as the function that returns the pieces (P denotes power set).
- Every TX block $t_{i,j}$ is enclosed by two *agreed* CP blocks $(c_{i,a}, c_{i,b})$ (CP blocks that are in some consensus result), where

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

$$b = \underset{k,k>j,\texttt{typeof}(b_{i,k})=\gamma}{\arg\min} (k-j).$$

  We call this the *agreed enclosure* of $t_{i,j}$.
- The *agreed piece* of $t_{i,j}$ defined by the agreed enclosure $(c_{i,a}, c_{i,b})$ is $\{b_{i,j} : a \leq j \leq b\}$. Note that *piece* $\subseteq$ *agreed piece* for some TX block. We define

$$\texttt{a-pieces} : B_i \to P(B_i)$$

  as the function that returns the agreed pieces.

2.2 Checkpoint consensus

2.2.1 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. We assume an application agnostic system and do not attempt to defend against the sybil attack.

An alternative is to use *proof of work*. This defends the sybil attack. But an incentive is needed for the nodes that expend their CPU resources.

2.2.2 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.

2.2.3 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.

2.2.4 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 $r - 1$. □

Lemma 2. *Eventually all node sees a valid $\mathcal{C}_{r-1}$.*

Proof. (sketch) Liveness is satisfied because $\mathcal{C}_{r-1}$ is eventually propagated to all node by gossiping. □

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

Lemma 3. *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 Δ before moving on to the next phase. Note that promoters waiting for a some time Δ to collect transactions does not violate the asynchronous assumption because this behaviour can be seen as a long delay in the asynchronous system.

Corollary 1. *Setup phase satisfies agreement and total order because the protocol is run deterministicly on the same input (lemma 1, lemma 3). It also satisfies liveness due to lemma 2.*

2.2.5 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 are 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?

Lemma 4. *Consensus phase satisfied agreement, total order and liveness.*

Proof. Defer proof? Refer to the papers. □

Theorem 1. *Checkpoint consensus satisfied agreement, total order and liveness.*

Proof. (sketch) Both phases are asynchronous so we do not need to make assumptions on when begin phase begins. Both phases also satisfy the agreement, total order and liveness. □

2.3 Validation

Since we reach consensus on checkpoints rather than all the transactions, we need to detect fraud, such as forks. Further, we need to ensure the system is secure in a sense that valid transactions cannot be forged into invalid transactions once it has reached consensus, and vice versa.

First, we define the requirements of a valid transaction $t_{i,j}$ as follows, much of it is derived from the TrustChain model.

1. There exist a pair $t_{i',j'}$ that satisfies the pair definition.
2. $t_{i,j}$ and $t_{i',j'}$ is created using `newtx`, i.e. valid signatures and hash pointers, etc..
3. There exist an agreed piece that contains $t_{i,j}$ and another agreed piece that contains $t_{i',j'}$. All the blocks in the agreed pieces have valid hash pointers, the blocks in the pieces do not need to be valid.
4. The CP blocks $c_{i,k}, c_{i',k'}$ that immediately follow $t_{i,j}$ and $t_{i',j'}$ are created using `newcp` using the same consensus result $\mathcal{C}_r$ as input and are in the agreed pieces.

Blocks that do not satisfy the definition above are not necessarily invalid. It may be the case that the validity cannot be determined due to incomplete information. For such cases we say its validity is unknown. Now we define an invalid transaction.

1. TODO

Now we define the properties that are desired by the validation protocol.

1. *Correctness*: The validation protocol outputs the correct result according to the aforementioned requirements.

[1] Reliable broadcast solves the Byzantine generals problem.

2. *Agreement*: If any correct node decides on the validity of a transaction, then all other correct nodes are able to reach the same conclusion.

3. *Liveness*: Any valid transactions can be validated eventually.

4. *Unforgeability*: If some transaction is determined to be valid, then it cannot be changed to an invalid transaction, the opposite also applies.

Note that the input of the validation protocol is a TX block, so these properties hold with respect to TX blocks. In practice, if a node has a set of TX blocks that are in the unknown state, then it must run the validation protocol to determine whether they are valid. Further, these conditions do not imply all invalid transactions (forks) can be found globally, only the validity of the TX blocks that the honest nodes are interested in can be determined. The advantage of this scheme is that it saves a lot of computational and bandwidth costs because nodes only run the validation protocol on the TX blocks of their own interest.

2.3.1 Validation protocol

Assume node u is aware of all the past consensus results $\mathcal{C}_r$. Suppose u wish to validate $t_{i,j}$. It performs the following.

1. Determine the pair $t_{i',j'}$.

2. Find the agreed enclosure for $t_{i,j}$ and $t_{i',j'}$ from $\mathcal{C}_r$, otherwise return "unknown".

3. Query i and i' for the agreed pieces and ensure hash pointers are correct. Otherwise return "unknown".

4. Check that $t_{i,j}$ and $t_{i',j'}$ are in the agreed pieces and are created correctly using `newtx`. Otherwise return "invalid".

5. Check the checkpoints $c_{i,k}$ and $c_{i',k'}$ that immediately follow $t_{i,j}$ and $t_{i',j'}$ are in the agreed pieces and are created in the same round, i.e. $\texttt{round}(c_{i,k}) = \texttt{round}(c_{i',k'})$. Otherwise return "invalid".

6. Return "valid".

Most likely $u = i$ or $u = i'$, because they are incentivised to check the validity of their TX blocks that are of unknown validity.

2.3.2 Analysis

In this section we analyse the validation protocol with respect to the four properties in section 2.3.

Lemma 5. *The validation protocol is correct.*

Proof. Correctness follows directly from the protocol specification, namely it directly implements the validation requirements. □

Lemma 6. *The validation protocol satisfies the agreement property.*

Proof. We proof by contradiction. Suppose two nodes i and j where $i \neq j$ runs the protocol on the same TX block $t_{k,l}$, i outputs "valid" and j outputs "invalid". i and j would have both picked the same enclosure $c_{k,a}$ and $c_{k,b}$, as these are determined from the consensus result. Then k must produce two agreed pieces that start with $c_{k,a}$ and end with $c_{k,b}$ with valid hash pointers, but one satisfies the validation protocol and the other one doesn't (e.g. missing $t_{k,l}$). Producing these two pieces is equivalent to producing a collision. Due to the properties of cryptographic hash functions this is not possible, thus a contradiction. □

Lemma 7. *The validation protocol does not satisfy the liveness property.*

Proof. (sketch) Nodes may be offline. □

There are many ways around the liveness. As long as the TX is validated once, then the agreed pieces can be gossiped.

Lemma 8. *A valid TX block cannot be made into an invalid TX block and vice versa.*

Proof. We proof by contradiction. Suppose $t_{k,l}$ is determined to be valid using the enclosure $c_{k,a}$ and $c_{k,b}$ and the agreed piece p. Then $t_{k,l}$ is forged into an invalid transaction. To forge it into an invalid transaction the attacker must either tamper with the data within the block or remove $t_{k,l}$ from p. In other words create p' where $p \neq p'$, enclosed by $c_{k,a}$ and $c_{k,b}$, and has valid hash pointers. Producing p' is equivalent to finding a collision, thus a contradiction.

TODO proof "vice versa". □

2.3.3 Time and message complexity

For a single transaction, the message complexity is linear with respect to the size of the agreed pieces for both part of the TX pair. Time complexity is constant since we only perfrom two queries (to i and i').

2.4 Fraud detection

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. 7

[2] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999. 7

[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. 7