A Formal Specification of OBFT Checkpointing for Blockchain Systems

April 16, 2021

1 Introduction

This document is a formal specification of a checkpointing system of nodes running the Ouroboros-BFT (OBFT) consensus for guaranteeing the integrity of an external blockchain. The specification of the OBFT protocol is described in [OBFT-18], which also contains a section on its application for a checkpointing system. More in-depth information on checkpointing, along with an analysis of the security guarantees it offers, is provided by [CHKP-19].

This specification is influenced by [SL-D5], and it aims to use the same conventions and notation.

2 Definitions

We define several types and corresponding accessor functions that are used further in the specification. In particular BlockRef, Vote and Checkpoint are all references to a blockchain block (by its hash and number) without a signature, with a single signature, and with multisignatures, respectively (fig. 1).

We also define the cryptographic functions sign and recoverPubKey (fig. 2) which are not explained in more detail. They are assumed to use the ECDSA signature scheme.

Finally, in fig. 3 we define the configuration parameters for the system, which are: *checkpointing interval* as the distance between two consecutive checkpointed blocks, the minimum number of signatures (votes) required to form a checkpoint, and the collection of federation nodes' public keys used for validating the votes.

Abstract types

$hash$	$\in$	BlockHash	block hash
sig	$\in$	Sig	signature
$pubKey$	$\in$	PubKey	public key

Derived types

$(hash, number)$	$\in$	BlockRef	$=$ BlockHash $\times \mathbb{N}$	block reference
$(hash, number, sig)$	$\in$	Vote	$=$ BlockHash $\times \mathbb{N} \times$ Sig	vote for a block
$msig$	$\in$	MSig	$=$ [Sig]	multisignature
$(hash, number, msig)$	$\in$	Checkpoint	$=$ BlockHash $\times \mathbb{N} \times$ Sig	checkpoint

Accessor functions

hash	$\in$	BlockRef $\to$ BlockHash	block hash
hash	$\in$	Vote $\to$ BlockHash	block hash
hash	$\in$	Checkpoint $\to$ BlockHash	block hash
number	$\in$	BlockRef $\to \mathbb{N}$	block number
number	$\in$	Vote $\to \mathbb{N}$	block number
number	$\in$	Checkpoint $\to \mathbb{N}$	block number
sig	$\in$	Vote $\to$ *Sig*	voter's signature

Figure 1: Type definitions

Cryptographic functions

sign	$\in$	BlockRef $\to$ Vote	sign block reference to form a vote
recoverPubKey	$\in$	Sig $\to$ PubKey$^?$	recover public key from the signature if valid

Figure 2: Cryptographic functions

Configuration parameters

k	$\in$	$\mathbb{N}$	checkpointing interval
m	$\in$	$\mathbb{N}$	minimum number of signatures
$fedPubKeys$	$\in$	[PubKey]	federation public keys

Figure 3: Cofniguration parameters

3 Casting a Vote

Checkpointing nodes form a federation running the OBFT consensus protocol. Each node holds the state formed by $lastChkp \in \mathsf{Checkpoint}$, which initially points to the blockchain genesis block and has an empty multisignature, and $votes \in [\mathsf{Vote}]$ which is a sequence of votes gathered from all the federation nodes.

Each node observes the blockchain using an associated node's JSON RPC. Upon seeing a new block, expressed by the NEWBLOCK signal carrying a $blockRef \in \mathsf{BlockRef}$, the node casts a vote if the difference between the new block's number and last checkpoint's number is equal to the k parameter. The checkpointing node does so regardless of any forks occurring on the blockchain. Thus multiple votes from the same node can be cast on the same block number. Only a single candidate is later selected to form a Checkpoint.

These rules are expressed in fig. 4. Casting a vote results in sending a transaction that carries the vote to the OBFT chain.

$$\text{No-Vote}\ \frac{\mathsf{number}\ blockRef \neq \mathsf{number}\ lastChkp + k}{k \vdash \begin{pmatrix} lastChkp \\ votes \end{pmatrix} \xrightarrow[\text{NEWBLOCK}]{blockRef} \begin{pmatrix} lastChkp \\ votes \end{pmatrix}} \tag{1}$$

$$\text{Register-Vote}\ \frac{\begin{array}{c}\mathsf{number}\ blockRef = \mathsf{number}\ lastChkp + k \\ vote := \mathsf{sign}\ blockRef \\ votes' := votes; vote\end{array}}{k \vdash \begin{pmatrix} lastChkp \\ votes \end{pmatrix} \xrightarrow[\text{NEWBLOCK}]{blockRef} \begin{pmatrix} lastChkp \\ votes' \end{pmatrix}} \tag{2}$$

Figure 4: Casting Vote inference rules

4 Reaching Consensus on a Checkpoint

First, we define the helper functions (fig. 5) used in reaching consensus on which block to checkpoint. Function findWinner analyses the *votes* sequence and finds a candidate with most votes (TODO: tie-breaker to be defined). The winner is then converted to a Checkpointusing the function makeCheckpoint.

Upon receiving the NEWVOTE signal, which is a transaction carrying a *vote*, the consensus rules (fig. 6) determine how the node state changes as a result. If the *vote* has a correct signature, then it is appended to the *votes* sequence, and we attempt to find a winner. If the winner is found, then a new Checkpointis formed and assigned to *lastChkp* variable. The *votes* sequence is cleared.

$$\mathsf{findWinner} \in [\mathsf{Vote}] \to \mathbb{N} \to \mathsf{BlockRef}^?$$

$$\mathsf{findWinner}\ votes\ m = \begin{cases} b & \exists b = \arg\max_b \mathsf{g}\ b,\ \mathsf{g}\ b \geq m \\ \diamond & \text{otherwise} \end{cases}$$

where

$$\mathsf{g}\ b \in \mathsf{BlockRef} \to \mathbb{N}$$

$$\mathsf{g}\ b = \left|[\ \diamond \mid\ \mathsf{hash}\ vote = \mathsf{hash}\ b,\ \mathsf{number}\ vote = \mathsf{number}\ b,\ vote \in votes]\right|$$

$$\mathsf{makeCheckpoint} \in [\mathsf{Vote}] \to \mathsf{BlockRef} \to \mathsf{Checkpoint}$$

$$\mathsf{makeCheckpoint}\ votes\ b = (\mathsf{hash}\ b, \mathsf{number}\ b, msig)$$

where $msig = [\mathsf{sig}\ vote \mid\ vote \in votes,\ \mathsf{hash}\ vote = \mathsf{hash}\ b]$

Figure 5: Checkpoint Consensus helper functions

$$\text{Invalid-Signature}\ \frac{\mathsf{recoverPubKey}\ (\mathsf{sig}\ \mathit{vote}) \notin \mathit{fedPubKeys}}{\begin{matrix}\mathit{m} \\ \mathit{fedPubKeys}\end{matrix} \vdash \begin{pmatrix}\mathit{lastChkp} \\ \mathit{votes}\end{pmatrix} \xrightarrow[\text{NEWVOTE}]{\mathit{vote}} \begin{pmatrix}\mathit{lastChkp} \\ \mathit{votes}\end{pmatrix}} \tag{3}$$

$$\text{No-Winner}\ \frac{\begin{matrix}\mathsf{recoverPubKey}\ (\mathsf{sig}\ \mathit{vote}) \in \mathit{fedPubKeys} \\ \mathit{votes}' := \mathit{votes}; \mathit{vote} \\ \mathsf{findWinner}\ \mathit{votes}'\ \mathit{m} = \diamond\end{matrix}}{\begin{matrix}\mathit{m} \\ \mathit{fedPubKeys}\end{matrix} \vdash \begin{pmatrix}\mathit{lastChkp} \\ \mathit{votes}\end{pmatrix} \xrightarrow[\text{NEWVOTE}]{\mathit{vote}} \begin{pmatrix}\mathit{lastChkp} \\ \mathit{votes}'\end{pmatrix}} \tag{4}$$

$$\text{New-Checkpoint}\ \frac{\begin{matrix}\mathsf{recoverPubKey}\ (\mathsf{sig}\ \mathit{vote}) \in \mathit{fedPubKeys} \\ \mathit{votes}' := \mathit{votes}; \mathit{vote} \\ \mathsf{findWinner}\ \mathit{votes}'\ \mathit{m} = \mathit{b} \\ \mathit{lastChkp}' := \mathsf{makeCheckpoint}\ \mathit{votes}'\ \mathit{b} \\ \mathit{votes}'' := [\,]\end{matrix}}{\begin{matrix}\mathit{m} \\ \mathit{fedPubKeys}\end{matrix} \vdash \begin{pmatrix}\mathit{lastChkp} \\ \mathit{votes}\end{pmatrix} \xrightarrow[\text{NEWVOTE}]{\mathit{vote}} \begin{pmatrix}\mathit{lastChkp}' \\ \mathit{votes}''\end{pmatrix}} \tag{5}$$

Figure 6: Checkpoint Consensus inference rules

5 Relaying Checkpoints into the Blockchain

This process is not fully defined yet, as OBFT imposes a latency of $3t + 1$ slots ([OBFT-18]) on when a checkpoint becomes stable, which could result in delays or decreased security in the blockchain. The following approaches are under consideration:

1. Federation posts checkpoint blocks (fixed checkpoint) – checkpoint blocks are special blocks that contain checkpoint information along with parent hash and number. The checkpoint is therefore placed at a fixed position in the blockchain. This approach conforms to the liveness guarantees presented in [CHKP-19], but depending on OBFT slot duration and blockchain block interval, it may result in periods of inactivity (or wasteful mining) in the blockchain.

2. Federation posts checkpoint transaction (floating checkpoint) – checkpoints are passed onto the blockchain as transactions. Once such a transaction is included in the chain, the chain can be replaced with a different chain that contains exactly the same checkpoints. This means that mining in the blockchain can continue without waiting for a checkpoint, but it also results in a varying position of the checkpoint in the chain, which lessens liveness guarantees.

3. Federation is queried for checkpoint information (oracle) – this approach is presented in [OBFT-18]. It removes the problem of latency, but it provides no liveness guarantees. It also makes the federation nodes susceptible to DoS attacks, as the federation nodes need to be available to every node on the blockchain network.

References

[CHKP-19] D. Karakostas and A. Kiayias. Securing proof-of-work ledgers via checkpointing, 2019. URL https://dimkarakostas.com/checkpointed_blockchains.pdf.

[OBFT-18] A. Kiayias and A. Russell. Ouroboros-bft: A simple byzantine fault tolerant consensus protocol, 2018. URL https://eprint.iacr.org/2018/1049.pdf.

[SL-D5] I. F. M. Team. A formal specification of the cardano ledger, iohk deliverable sl-d5, 2019. URL https://github.com/input-output-hk/cardano-ledger-specs/tree/master/shelley/chain-and-ledger/formal-spec.