

# Consensus Protocol Detail

In this document, we use the term **nodes** to refer to the Ethereum clients participating the consensus protocol. There are two types of nodes, proposers and validators. Consensus is divided into heights. For each height  $H$ , the system will commit a block  $B^H$ . To commit block  $B^H$ , we may process multiple rounds at height  $H$ . Note that in each round, we have one proposer (chosen in a round-robin fashion in each round) and the rest of nodes are validators. In the following description, we use  $+\frac{2}{3}$  to denote a number greater than  $\frac{2}{3} \times n$ , where  $n$  is the number of nodes. In addition, we use  $(H, R)$  to denote round  $R$  at height  $H$ .

## GENERAL MESSAGE STRUCTURE

During the consensus process, nodes broadcast messages. Messages have the form  $M_{X,H,R,u}(B)$ . A message contains the sender of the message  $u$ , the height  $H$  and round  $R$  in which  $u$  broadcasts the message, and the message type  $M_X$ . There are four messages types,  $Proposal_{VotingInstruction}$ ,  $Proposal_{NewBlock}$ ,  $Vote_{Proposal}$ , and  $Vote_{PreCommit}$ . Note that  $B$  refers to a block and may be  $nil$ .

## VOTES AND LOCKSETS

During the consensus process, nodes broadcast votes. A vote is nothing but a message. Specifically, a vote has the form  $Vote_{X,H,R,u}(B)$ , and there are two types of votes,  $Vote_{Proposal}$  and  $Vote_{PreCommit}$ . Apart from the information stored in a message,  $Vote_{X,H,R,u}(B)$  also contains a hash of block  $B$ . Nodes will save votes in **locksets** for future reference. A lockset  $Lockset_{X,H,R,u}$  must store  $+\frac{2}{3}$  votes  $Vote_{X,H,R,u}(B)$  such that  $H' = H$ ,  $R' = R$ , and  $X' = X$ . Note that we may have  $Vote_{X,H,R,u_1}(B_1)$  and  $Vote_{X,H,R,u_2}(B_2)$  in  $Lockset_{X,H,R,u}$  such that  $u_1 \neq u_2$  and  $B_1 \neq B_2$ . Moreover,  $B_1$  may be  $nil$ .

## PROPOSALS

In the beginning of each round, a proposer  $u$  will broadcast a proposal. A proposal is a message and has the form  $Proposal_{X,H,R,u}(B)$ . There are two types of proposals,  $Proposal_{VotingInstruction}$  and  $Proposal_{NewBlock}$ . Other information contained in a proposal is listed below.

- $Proposal_{VotingInstruction,H,R,u}(B)$ :
  - 1) Block  $B$ ,
  - 2)  $Lockset_{Proposal,H,R-1,u}$ .
- $Proposal_{NewBlock,H,R,u}(B)$ :
  - 1) Block  $B$ ,
  - 2) If  $R = 0$ , the message also contains  $Lockset_{PreCommit,H-1,R',u}$  for some round  $R'$  in height  $H - 1$ .
  - 3) If  $R > 0$ , the message also contains  $Lockset_{Proposal,H,R-1,u}$ .

## QUORUM OF A LOCKSET $S$ , $Quorum(S)$

If there are  $+\frac{2}{3}$  votes for the same block  $B$  in a lockset  $S$  and  $B \neq nil$ , then  $Quorum(S) = B$ . Otherwise, we say  $Quorum(S)$  does not exist.

## VALIDITY OF MESSAGES AND LOCKSETS

Consider message  $M_{X,H',R',u(B)}$ . For the message to be valid in  $(H, R)$ ,  $H'$  must be equal to  $H$ ,  $R'$  must be equal to  $R$ , and the signature must be consistent with  $u$ . Note that a vote is a message, and a vote is valid in  $(H, R)$ , if it is a valid message in  $(H, R)$ . Moreover, we may save votes in locksets for future reference. Recall that a lockset  $Lockset_{X,H,R,u}$  must store  $+\frac{2}{3}$  votes  $Vote_{X',H',R',u'}(B')$  such that  $H' = H$ ,  $R' = R$ , and  $X' = X$ . We say that  $Lockset_{X,H,R,u}$  is valid if all the votes in it are valid in  $(H, R)$  and are from different nodes. Other constraints for *Proposal* messages to be valid are listed below.

- $Proposal_{VotingInstruction,H,R,u}(B)$ :

The message must contain  $Lockset_{Proposal,H,R-1,u}$  such that:

- 1)  $Lockset_{Proposal,H,R-1,u}$  is valid.
- 2)  $Quorum(Lockset_{Proposal,H,R-1,u}) = B$ .
- 3)  $u$  must be the proposer in  $(H, R)$ .

- $Proposal_{NewBlock,H,R,u}(B)$ :

If  $R = 0$ , then the message must contain  $Lockset_{PreCommit,H-1,R',u}$  such that:

- 1)  $Lockset_{PreCommit,H-1,R',u}$  is valid.
- 2)  $Quorum(Lockset_{PreCommit,H-1,R',u}) = B^{H-1}$ , where  $B^{H-1}$  is the block committed in height  $H - 1$ .
- 3)  $u$  must be the proposer in  $(H, R)$ .

If  $R > 0$ , then the message must contain  $Lockset_{Proposal,H,R-1,u}$  such that:

- 1)  $Lockset_{Proposal,H,R-1,u}$  is valid.
- 2)  $Quorum(Lockset_{Proposal,H,R-1,u})$  does not exist.
- 3)  $u$  must be the proposer in  $(H, R)$ .

## SIGNATURE AND HASH FUNCTION

When a node broadcasts a message, it needs to sign the message. In our implementation, we use the built-in ECDSA signature. The signer can sign the data with private key. The produced signature is in the  $(V, R, S)$  format. The signer's address can be derived from the signature  $(V, R, S)$  using secp256k1 elliptic curve. On the other hand, a message may contain a hash of Block. In our implementation, we use the built-in keccak256 hash function.

### CONSENSUS STEPS

Next, we consider the steps for a node  $u$  in  $(H, R)$ ,  $R = 0, 1, 2, \dots$ .

#### Step 1 (Propose):

If  $u$  is a proposer:

- 1) If  $Quorum(Lockset_{Proposal,H,R-1,u}) = B$  exists,  $Prop = Proposal_{VotingInstruction,H,R,u}(B)$ .  
Otherwise, create a new block  $B$  and set  $Prop = Proposal_{NewBlock,H,R,u}(B)$ .
- 2) Broadcast  $Prop$ .
- 3) Go to step 2.

Otherwise, go to Step 2.

#### Step 2 (Prevote):

Case 1:  $u$  has broadcast  $Vote_{PreCommit,H,R-1,u}(B)$  for some block  $B$  in  $(H, R - 1)$ :

- 1) Broadcast  $Vote_{Proposal,H,R,u}(B)$ .
- 2) Go to Step 3.

Case 2:  $u$  receives a valid  $Proposal_{VotingInstruction,H,R,u}(B)$  or a valid  $Proposal_{NewBlock,H,R,u}(B)$  within  $TimeoutProposal$  after height  $H$  starts:

- 1) Broadcast  $Vote_{Proposal,H,R,u}(B)$ .
- 2) Go to Step 3.

Case 3: Otherwise:

- 1) Broadcast  $Vote_{Proposal,H,R,u}(nil)$ .
- 2) Go to Step 3.

#### Step 3 (Precommit):

- 1) Collect a valid lockset  $Lockset_{Proposal,H,R,u}$  of  $Vote_{Proposal}$ .
- 2) Wait for  $TimeoutProposalVote$  to store more votes in  $Lockset_{Proposal,H,R,u}$ .
- 3) If  $Quorum(Lockset_{Proposal,H,R,u}) = B$  exists, broadcast  $Vote_{PreCommit,H,R,u}(B)$ . Otherwise, broadcast  $Vote_{PreCommit,H,R,u}(nil)$ . Note that if a quorum exists before  $TimeoutProposalVote$ ,  $u$  can stop collecting votes.
- 4) Go to Step 4.

#### Step 4 (Commit):

- 1) Collect a valid lockset  $Lockset_{PreCommit,H,R,u}$  of  $Vote_{PreCommit}$ .
- 2) Wait for  $TimeoutPrecommitVote$  to store more votes in  $Lockset_{PreCommit,H,R,u}$ .
- 3) If  $Quorum(Lockset_{PreCommit,H,R,u}) = B$  exists, commit  $B$  and go to Step 1 in  $(H + 1, 0)$ .
- 4) Otherwise, go to Step 1 in  $(H, R + 1)$ .

### VALIDITY OF A BLOCK

Once a node  $u$  commits a block  $B$  in  $(H, R)$ , it stores to database a key-value pair  $(Hash(B), Lockset_{PreCommit,H,R,u})$ . Nodes could load  $Lockset_{PreCommit}$  to verify blocks or to sync with others. More specifically, a block  $B$  is valid if the key-value pair  $(Hash(B), S)$  satisfying the following constraints:

- 1)  $S$  is a valid locksets storing  $Vote_{PreCommit}$ .
- 2)  $Quorum(S) = B$ .

### CONSTANT SETTING

- $TimeoutProposal$ : The Maximum time for waiting a proposal. This is set to 3 seconds initially.
- $TimeoutProposalVote$ : The maximum time for waiting more  $Vote_{Proposal}$ . This is set to 1 second initially.

- *TimeoutPrecommitVote*: The maximum time for waiting more  $Vote_{PreCommit}$ . This is set to 1 seconds initially.
- *TimeoutFactor*: This is the factor used to extend the above timeouts after each round. More specifically,  $TimeoutX$  in round  $R = TimeoutX \times TimeoutFactor^R$ . *TimeoutFactor* is set to 1.5.