# EverTrust consensus

A secure, fair, scalable and high performance blockchain consensus

Version 0.20

ChainRD Technologies (Beijing) Inc.

## Contents

## 1 Introduction

The design goal of ET consensus algorithm is simple, which is, secure, fair, majority-consent, massively scalable with low latency and high throughput.

The ET consensus algorithm gives equal opportunity to all capable, active and willing-to-participate nodes to participate and accepts pluggable consensus node qualification mechanism to mitigate prominent risks like Sybil attack. To mitigate against cross-block and cross-master collusion, ET consensus introduces per-block *block master*s, an ordered list of consensus nodes unpredictable beforehand but deterministic just-in-time for block generation and population.

The ET consensus algorithm leverages the following properties a blockchain network exposes:

1) By accepting block x, a consensus node implicitly accepts all cryptographically linked blocks before it. There's no need to explicitly confirm each block, instead asynchronous nonblocking cumulative confirmation is much more efficient.

2) Most consensus nodes are honest, so trustworthy message proxying and multicast are possible, which enables scalable, low latency, high throughput and load-balanced deterministic population of blocks and transactions.

Based on the above properties, ET consensus employs a state-of-the-art dual-ledger asynchronous nonblocking cumulative consensus architecture with the regular ledger for regular blocks composed of transactions and state changes and the commit ledger for commit blocks composed of cumulative assertion evidences of the regular blocks & updates on the consensus committee.

With this dual-ledger asynchronous nonblocking cumulative consensus architecture, the amortized messaging complexity for block consensus is drastically reduced from O($n^2$) to the low order of O(n), hence the massive scale and high performance it's capable of achieving.

## 2 Terms & definitions

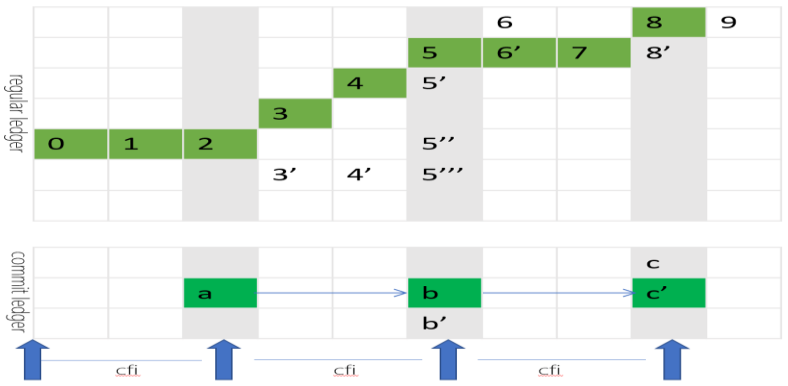
| Term | Definition |
|---|---|

| | |
|---|---|
| blockchain node | A computing device joining the blockchain network, also called “node” for simplicity |
| Block assertion | Assertion evidence on the latest block “path” since last confirmed (regular) block. |
| Block masters | An ordered list of consensus nodes responsible for forging a specific block or blocks; they are unpredictable beforehand but deterministic just-in-time. The number of block masters for a block is denoted as *m* |
| Block path | A chain of blocks cryptographically linked |
| Block wait time | The time it takes to receive block x after block x-1, detectable and configurable |
| Commit block | A blockchain block that is composed of a confirmed block branch since last confirmation interval, together with block assertions as evidence from consensus nodes. |
| Commit ledger | The blockchain ledger for commit blocks |
| Confirmation interval | The interval in number of regular blocks that a consensus confirmation is performed, denoted as *cfi* |
| Consensus node | A blockchain node participating in the blockchain consensus process. Also called a miner for historical reasons. |
| Consensus committee | An auto-adjusted committee composed of all consensus nodes responsible for the consensus process. The size of the committee is denoted as *n* |
| Master rank | The rank of a node as a master responsible to forge a block, ranged from 0 (highest) to n (lowest). |
| Observer committee | Composed of all blockchain nodes that are not qualified as consensus nodes. |
| Regular block | A blockchain block that is composed of transactions and state updates and connected with adjacent blocks cryptographically. |
| Regular ledger | The blockchain ledger for regular blockchain blocks |

## 3 Nonblocking dual ledger

The dual-ledger architecture is for pipelined asynchronous and nonblocking consensus. The regular ledger, composed of regular blocks, moves ahead in a nonblocking fashion as new regular blocks are generated. The commit ledger, independently moves ahead every *cfi* regular blocks.

The regular ledger may branch from the last confirmed regular block, only be corrected on next *cfi* boundary by a new commit block which, after collecting block assertions from all consensus nodes, decides the latest confirmed regular block hence eliminating the branches since the last *cfi* boundary.



The commit ledger may also tentatively branch but the longest valid commit chain endorsed by the majority of the consensus nodes prevails.

A regular block, is mainly composed of new transactions executed and the state changes caused as well as other block data. A commit block, is mainly composed of the “path” of regular blocks confirmed, condensed block assertions from each consensus node in the consensus committee as evidence, as well as committee updates.

On receiving a regular block, a node verifies it and if valid append it to a branch starting from the latest confirmed regular block. If it’s a master for the block or for next block, it assumes mastership if/when needed.

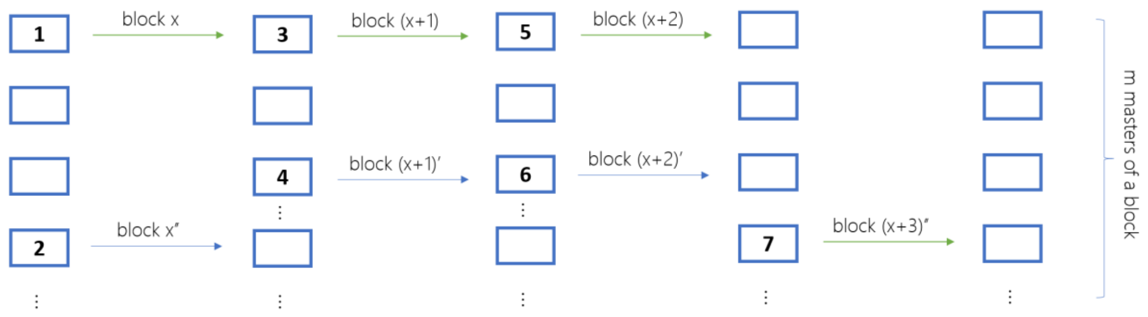
On receiving a commit block, a consensus node removes the invalid branch(es) since last confirmed regular block and commit the confirmed blocks to the regular ledger. All consensus logic restarts from this latest committed regular block.

## 4 Three confirmation states

To improve performance and handle possible split-brain and merge-brain cases, the ET consensus defines three confirmation states for a block or blockchain branch: local, island-local and global. Local confirmation is after local verification of the block or block branch but is by no means final. Island-local confirmation is an intermediate state for an island that is not connecting majority of the consensus committee. Global confirmation refers to the confirmation state from majority of the consensus committee.

## 5 Per-block block master(s)

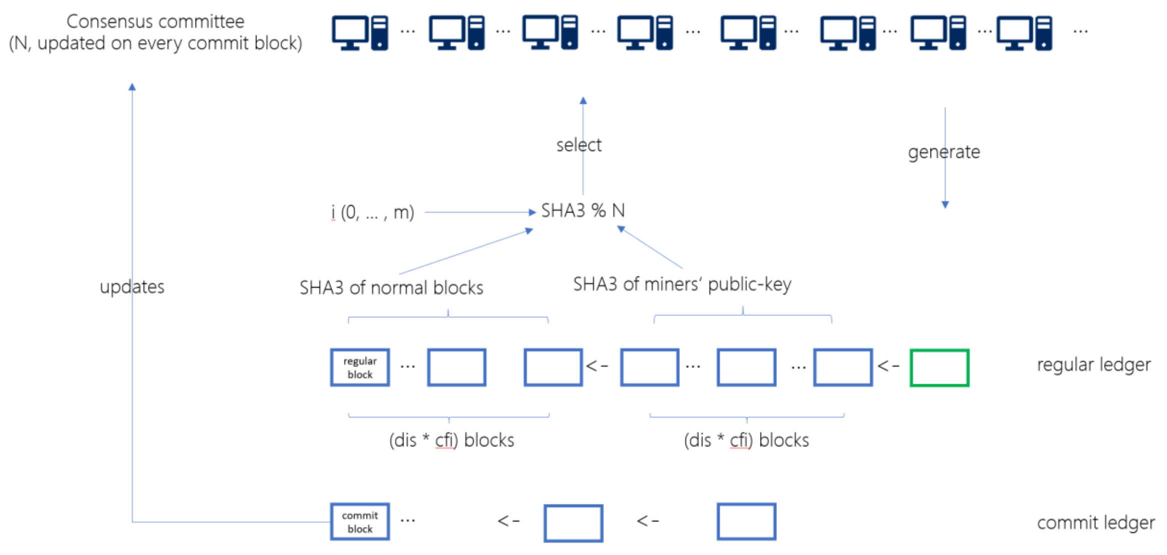
For each block to be generated, the ET consensus decides, unpredictably beforehand but deterministically just-in-time the list of *m* masters (master list) responsible for the generation and population of that block. With enough variants introduced on master selection, cross-block and cross-master collusion can be effectively mitigated.



As shown in the diagram, node 2), 4), 6) ,7) assume mastership if received an incorrect block from its immediate higher-ranking peer master, or waited master_rank*block_wait_time but still not received a valid block.

### 5.1 Master selection

To mitigate against collusion by consensus nodes of adjacent blocks, the per-block masters must be, unpredictably deterministic, i.e. unpredictable beforehand but deterministic just-in-time, so that the influence of a specific miner and block must be ignorable and unpredictable. To achieve this, there could be unlimited number of ways, but the ET consensus uses the following formula to decide the m masters for a block x.



For block x, the index (in the ordered consensus committee) of the master for rank r, i.e. the r$^{th}$ master, is:

```
first 32 bits of SHA3 (
    SHA3 (
        block-hash of x – 2*dis*cfi, // dis: a distance factor, by default set to 100
        … …
        block-hash of x – dis*cfi - 1,
        miner-coinbase of x – (dis*cfi),
        … …
        miner-coinbase of x – 2,
        miner- coinbase of x - 1 ),
    island_id, //0 if mainland, otherwise SHA3 of consensus nodes in island
 committee
    SHA3_of_second_latest_commit_block,
    view_id, // the number of attempts to do mastership calculation for block x
    r, // master rank, 0 < = r < m, where m is the number of masters configured.
    x, //next block number
) % number-of-consensus-nodes
```

If the number returned is the index of the current node in the consensus committee, then the node is the r$^{th}$ master for block x. In case, the index for rank r returned collides with one with higher rank r' ( 0 <= r' < r), the next index is selected and so on.

Note that if x < 2 * dis * cfi blocks, zero is used in the calculation.

On receiving a candidate for block x - 1, a blockchain node would, in a non-blocking mode, figure out all m masters for block x as well as its own ranking in the list of masters for block x, just in case higher-ranking masters' candidates for block x don't go through.

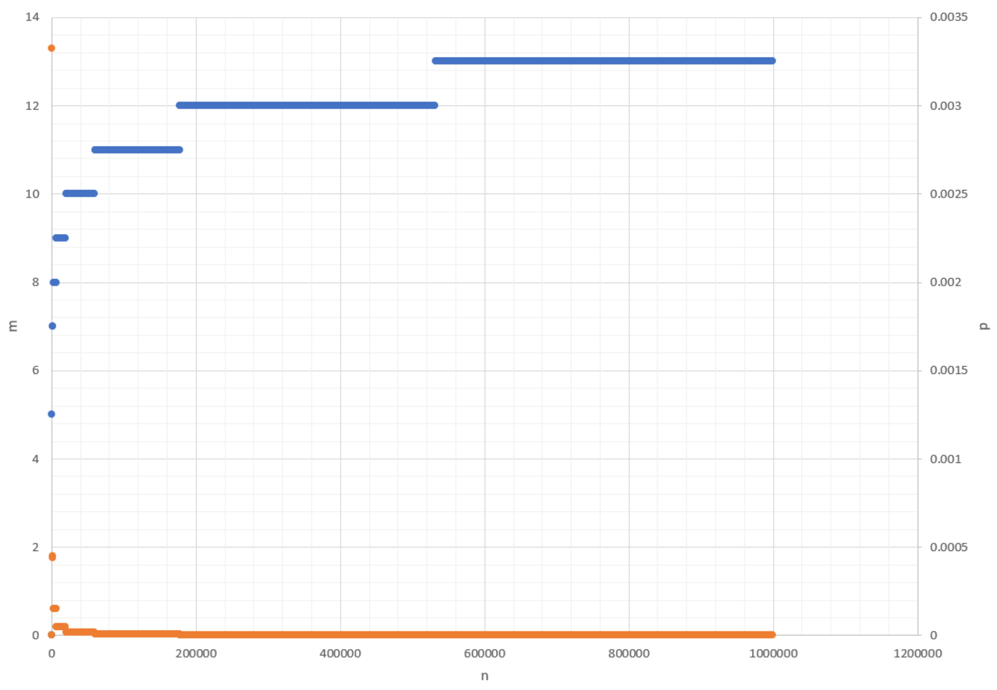
The formula to select masters introduces enough variables and span enough block distances so that it makes it effectively impossible for adjacent miners to collude. Also, a miner cannot effectively collude with downstream miners via adding or removing TXs in the block.

If all m masters of a block fail to propose an acceptable block, the next batch of m masters are automatically selected until all consensus nodes are enumerated. Given the fact that majority of the nodes are honest, the blockchain will auto-recover timely.

Nonetheless, for performance and availability reasons, it's desirable to keep the probability of all m masters being bad actors low, which means the right *m* per *n* (total number of consensus nodes) must be selected. As the selection of masters is unpredictable beforehand, assuming 2/3 of the nodes in the consensus committee are honest, the probability that all *m* masters for a block in a consensus committee of n nodes is dishonest, is calculated as follows:

$$P = [(n/3) /n] * [(n/3 -1)/(n-1)] * \dots *[(n/3 - m+1) /(n-m+1)]$$

This probability is negatively corelated to *n* and *m*. The following diagram shows the correlation of *n*, *m* for P = 1/*n*.



For a given *n*, *m* must be selected to satisfy P << 1/*n*. For a blockchain network with 10,000 nodes, by setting *m* to 11, we can keep the probability P at less than 1/100,000. Remarkable!

## 5.2 Master switching

Each master in the ordered list of *m* masters for a block, serves as a watchdog to its higher-ranking peer masters. If after master_rank * block_wait_time, a master at rank master_rank, has not yet received a valid block x, it would assume mastership automatically. This design ensures that there's always a node generating block x and threats from dishonest nodes, either by trying to do harm without generating the block or generated an incorrect one, is mitigated timely and resiliently.

When the first set of *m* masters are enumerated, another set of *m* masters are automatically considered until all *n* nodes in the consensus committee are enumerated.

When the consensus committee is sufficiently large, each block most likely is associated with a significantly different ordered set of *m* masters. Even if it's not large, the ordering of masters for each block should be significantly different.

Each master list can be configured to generate multiple blocks before master-list switching, or have the block size sufficiently large if one-block-per-master-list is preferred.

# 6 Consensus pipelining

The consensus process is composed of two independent pipelines parallel to each other: regular block pipeline and commit block pipeline.

The regular block pipeline moves ahead from one regular block to the next in sequence. At each regular block height *x*, multiple candidates from multiple masters may exist, but optionally delayed *master_rank* * *block_wait_time* for each master ranked at *master_rank* for block *x*. Here, *block_wait_time* is the configurable or detectable delay in milliseconds to receive a block after the previous one.

The commit block pipeline moves ahead from one commit block to the next in sequence. Each commit block cumulatively confirms the latest regular block since last confirmed regular block by the previous commit block. Regular blocks not confirmed in the commit block will be discarded.

The assumption is that, given a sufficiently large number of masters for block *x*, *m*, one of them will indeed propose a valid block (regular or commit) with very high probability. If not, the next set of *m* masters will take over until all consensus nodes are enumerated. Since majority of consensus nodes are deemed honest, consensus will be reached eventually.

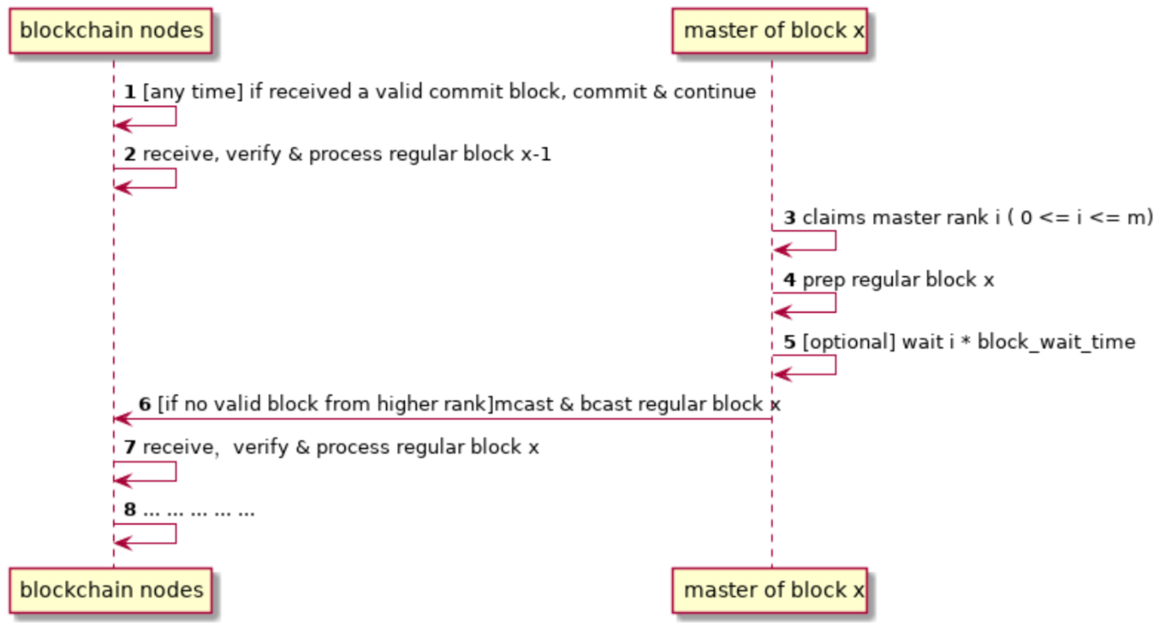
## 6.1 Regular block pipelining

1) At any time, if a node receives a valid commit block, it will commit the regular blocks confirmed and update the consensus and observer committees as directed.

   Because the commit block pipeline is parallel to the normal block pipeline, newer normal blocks, if not linked to the latest confirmed one, will have to be rolled back.

2) On receiving block *x-1*, a blockchain node checks the miner's rank in the master list for the block. If it's from a higher ranking master than the one at hand and received within allowed block wait time (*master_rank* * *block_wait_time*), or if it's the first candidate for the block, a blockchain node verifies it. If it verifies, the blockchain node rolls back the effect of the previous candidate block (if present) and attaches this new block to the local regular ledger.

3) Calculate all *m* masters for block *x*, together with its own ranking in the ordered master list


blockchain nodes
master of block x
1 [any time] if received a valid commit block, commit & continue
2 receive, verify & process regular block x-1
3 claims master rank i ( 0 <= i <= m)
4 prep regular block x
5 [optional] wait i * block_wait_time
6 [if no valid block from higher rank]mcast & bcast regular block x
7 receive, verify & process regular block x
8 … … … … …
blockchain nodes
master of block x

4) If local node is one of the *m* masters for block x, it would prepare regular block *x* by selecting the TXs to be included and executing them in parallel and time-boxed.

5) [optional] Wait for *master_rank* * *block_wait_time*, unless a candidate from the immediate higher-ranking master is invalid

   If *master_rank* is less than m, implementation may choose not to wait or wait less for better latency (trading bandwidth for latency), e.g. 1/2 * *master_rank** *block_wait_time*.

   Note that there can be multiple candidates for block *x* on the wire in parallel and they could be all valid. This is to improve latency if and when a higher-ranking master for block *x* is not "fast" enough or not reachable by a majority of the nodes on the blockchain network. Branch trimming is done via commit block pipeline that happens every *cfi* normal blocks.

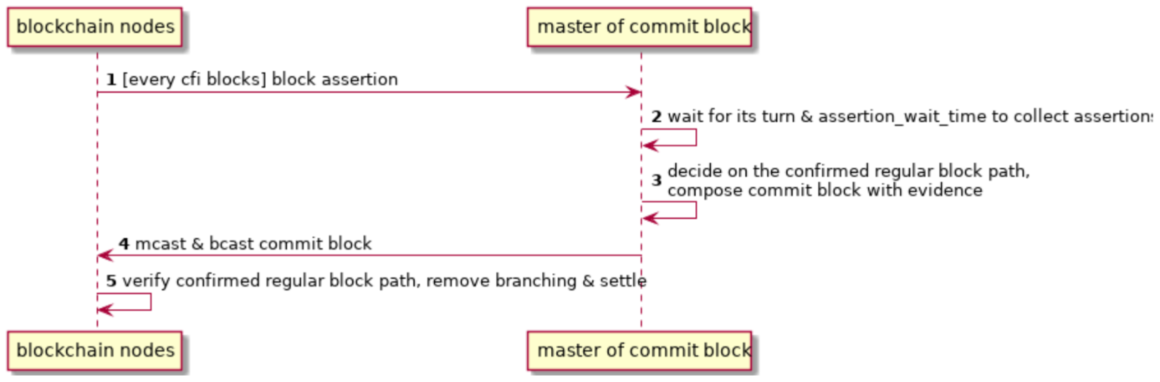
6) Multi-cast block *x* to all other *m-1* peer masters of block *x* and all *m* masters of block x+1, then broadcasts (gossips) it to all other nodes.

7) Receive, verify and process block *x* as it would block *x-1*

8) Then consensus process for another regular block starts….

## 6.2 Commit block pipelining

Commit block pipelining is to commit regular blocks via a commit block after the cfith regular block since the last confirmed regular block. A commit block removes possible branching in the regular ledger since the last block confirmation. The commit block pipelining progresses as follows:


blockchain nodes
master of commit block
1 [every cfi blocks] block assertion
2 wait for its turn & assertion_wait_time to collect assertions
3 decide on the confirmed regular block path, compose commit block with evidence
4 mcast & bcast commit block
5 verify confirmed regular block path, remove branching & settle
blockchain nodes
master of commit block

1) After *cfi*th regular block since the last confirmed regular block, a consensus node analyzes its local regular block branch(es) since last confirmed regular block, decides its assertion of the block path to confirm, and sends the block assertion to all of the *m* masters for the next commit block.

   The block assertion includes the block path, instead of only the latest block asserted, so that the longest common block path can be identified and confirmed.

   Because all *m* masters for the next commit block, receive all block assertions from all blockchain nodes, if the commit block from a master is not correct, e.g. with discounted block assertions due to master foul play, denial-of-service attack or network reachability problem etc., lower ranking peer masters will generate and send out its confirmation block(s) to all blockchain nodes.

   If a higher ranking master issues a commit block with less assertions which causing split-brain, a lower ranking master must correct it by issuing its commit block if split-brain is not true.

Note that advanced networking optimization mechanisms can be employed to avoid all nodes flood all *m* masters if needed.

2) The *m* masters of the commit block, wait at least *block_wait_time* to receive as many as possible block assertions from all nodes.

   The wait here is to ensure all block assertions from still-connected blockchain nodes arrive at all *m* masters for the commit block.

3) The *m* masters of the commit block, analyze the block assertions received and then choose the majority-confirmed block path based on the commit criteria.

   The commit criteria can be configurable, e.g. choosing the longest block path with endorsement from 2/3 of the consensus committee (default), or simply the full path with most assertions, etc.

   The commit block, includes the selected block path, condensed signature evidences from all consensus nodes, as well as additions and deletions of the consensus committee and observer committee.

   The algorithm guarantees a commit block generation, since the worst case is a confirmation of the last confirmed regular block.

4) The master(s) of the commit block multicasts its commit block to all of the *m*-1 peer masters of this block, then broadcasts (gossips) it to all other nodes.

   Lower-ranking masters of the commit block, wait *master_rank* * *block_wait_time* to receive candidate commit blocks from its higher-ranking peers. If no valid commit block is received, it sends out its commit block. To trade bandwidth for better latency, this wait can be enforced by the recipients.

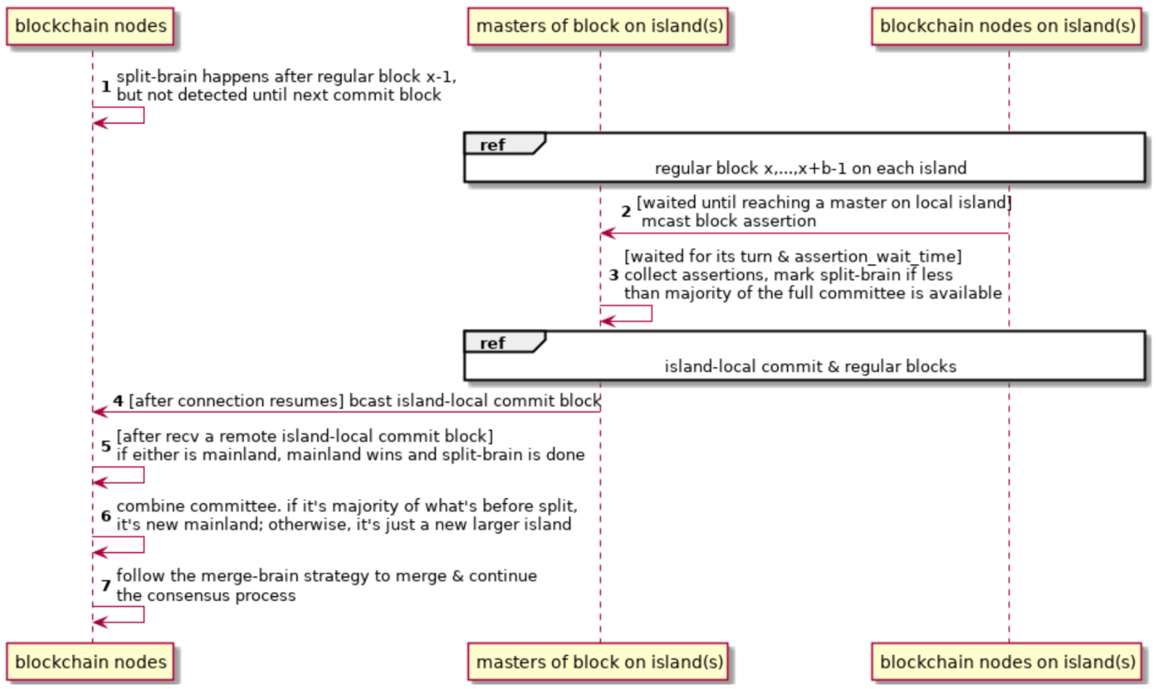
5) Each node verifies the commit block, rollback abandoned regular blocks, commit confirmed regular block path and continue.

   Each node also validates and records the updates to the consensus committee and observer committee. The new consensus committee decided in this commit block will take effect after the next commit block.

If after (2 + m) * *block_wait_time*, a blockchain node does not receive a valid commit block, it would calculate the next set of *m* masters for the commit block, and multicast its block assertion to this new set of *m* masters to restart commit block pipelining. This continues until a valid commit block arrives or all consensus nodes are enumerated. Since majority of consensus nodes are honest, a commit block will arrive. However, this commit block could signal a split-brain.

## 6.3 Split-brain & merge-brain

Split-brain happens when there's availability problem due to outage or denial-of-service attack. Split-brain divides the blockchain network into multiple islands. If there's one island with at least 2/3 of the consensus committee, it's the mainland and the mainland always wins. There may be no mainland at all.


blockchain nodes
masters of block on island(s)
blockchain nodes on island(s)
1 split-brain happens after regular block x-1, but not detected until next commit block
ref
regular block x,...,x+b-1 on each island
2 [waited until reaching a master on local island] mcast block assertion
3 [waited for its turn & assertion_wait_time] collect assertions, mark split-brain if less than majority of the full committee is available
ref
island-local commit & regular blocks
4 [after connection resumes] bcast island-local commit block
5 [after recv a remote island-local commit block] if either is mainland, mainland wins and split-brain is done
6 combine committee. if it's majority of what's before split, it's new mainland; otherwise, it's just a new larger island
7 follow the merge-brain strategy to merge & continue the consensus process
blockchain nodes
masters of block on island(s)
blockchain nodes on island(s)

Split-brain may happen any time. If it happens, it'd get detected during block confirmation. As show in the diagram,

1) Split-brain happens after regular block *x-1* due to outages or denial-of-service etc. but not detected yet.

As shown in the diagram, split-brain divides the blockchain network into islands, each of which is generating its own blocks starting from block *x* without knowing the existence of the others.

If and when a master for a block is on the other island, there will be delays but all islands would function independently as expected.

On split-brain, no new consensus nodes can be added on islands. However, mainland functions as usual.

2) When it's time for block confirmation, all blockchain nodes on an island enumerate and possibly expand the *m* masters for the commit block, and finally a reachable master (on local island) gets the block assertions.

   A consensus node starts a timer after sending its block assertion to the *m* masters, and waits for (2 + *m*) * *block_wait_time*. If no valid commit block is received, picks up another set of *m* masters and multicasts its assertion to them.

3) A master for the commit block, wait (1 + *master_rank*) * *block_wait_time* for its turn to receive all block assertions and issue its commit block.

   If less than 2/3 (configurable) of the committee is available, mark split-brain and sets island-local in the commit block.

   On split-brain, island-local consensus continues. All regular and commit blocks are marked as such.

4) When marked split-brain, a master will try to reconnect "lost" consensus nodes. After connection resumes, a master on an island broadcasts (gossips) its island-local commit block to all consensus nodes reachable.

5) After receiving a "remote" commit block(s) from a remote island, if there's mainland, mainland wins and split-brain is done with all islands sync from the mainland.

   A mainland is an island with block assertions in its commit block from 2/3 (configurable) or more of the original consensus committee.

6) All blockchain nodes combine committee. If the new committee is 2/3 of the committee before split-brain, a new mainland is formed; otherwise, it's just a new larger island.

   If after a pre-configured time period (in number of commit blocks) no mainland is formed, a blockchain network votes to turn an island to mainland.

7) Follow the merge-brain strategy (e.g. roll back to where it splits, choose the longest within simply majority of the consensus, etc.) to decide the latest commit block together with the regular blocks confirmed, and continue the consensus process.

# 7 O(n) complexity

With regard to the consensus efficiency, for a small *m* with sufficient confidence, *m* << *n* as the probability graph in section **Error! Reference source not found.** shows, the a mortized number of messages per regular block is in the low order of O(n). Here's the simple proof:

## Best-case scenario

In the best case scenario where the first master is always honest and capable, we have:

1) Regular blocks: cfi normal blocks, n * cfi messages
2) Block assertion, n*m messages
3) Block confirmation: n messages
4) 1) +2) +3) divided by cfi: (n*cfi + n*m + n) /cfi = [(cfi+m+1) /cfi]*n ≈ m*n

## Worst-case scenario

In the statistically worst-case scenario where the last master is honest and capable, we have:

1) Regular blocks (all masters propose): cfi regular blocks, n * cfi *m messages
2) Block assertion, n*m messages
3) Block confirmation (all masters confirm): n*m messages (all m masters confirm but only the last is honest)
4) 1) +2) +3) divided by cfi: (n*cfi*m + n*m + n*m) /cfi = [(cfi+2)*m /cfi*]n ≈ m*n

## 8 References

1) https://bitcoin.org/bitcoin.pdf
2) https://github.com/ethereum/wiki/wiki/White-Paper
3) https://github.com/ethereum/research/blob/master/papers/casper-basics/casper_basics.pdf
4) https://eprint.iacr.org/2016/871.pdf