

# DiemBFT Documentation and Test Report

Team Name: RVP

Raj Patel, 114363611  
Prince Kumar Maurya, 114354075  
Vishal Singh, 114708875

## Setup

[Test case Template](#)  
[Generating Test cases](#)  
[Logging files and Ledger Info](#)

## Implementation

[Pending Block Tree](#)  
[MemPool](#)  
[Persistent Ledger](#)  
[Speculative Ledger](#)  
[Initiation of Diem Process and chain processing](#)  
[Genesis Block](#)  
[Client Request](#)  
[Replica Info](#)  
[Chain Termination](#)

## Test Case Report - Our Use Cases

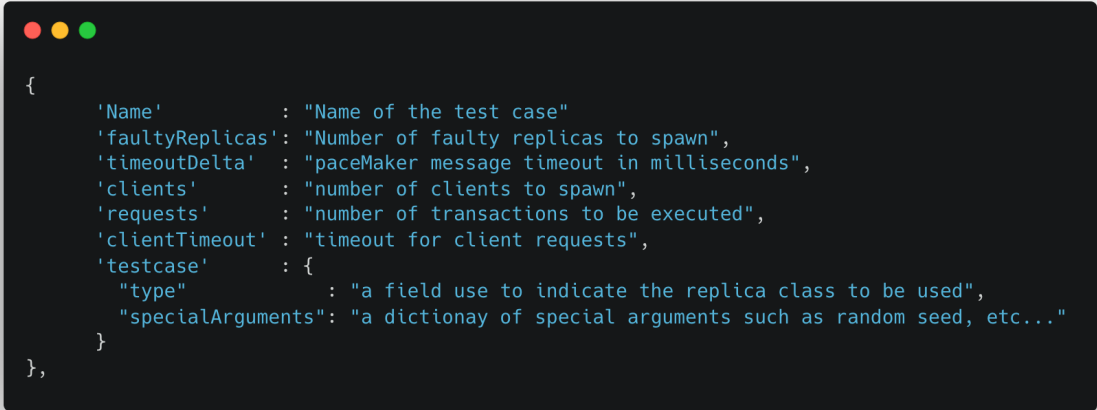
[Normal Execution Flow](#)  
[Normal Replicas with multiple clients](#)  
[Client requests timeout, re-submission, and handling request de-duplication](#)  
[Omission Failure](#)  
[Forge Signature](#)  
[Delay Failures](#)

## Test Case Report - Phase 2 Test Cases

[Proposal Drop](#)  
[In the third round, proposal messages to validators 1 and 2 are dropped from the leader](#)  
[Proposal Delay](#)  
[Proposal Drop + Timeout Delay](#)  
[SetAttr Failure](#)  
[Failures in multiple rounds](#)  
[Proposal Drop Twice](#)  
[Vote Delay Twice](#)  
[Probabilistic Failures](#)

# Setup

## Test case Template



```
{
  'Name'      : "Name of the test case"
  'faultyReplicas': "Number of faulty replicas to spawn",
  'timeoutDelta' : "paceMaker message timeout in milliseconds",
  'clients'     : "number of clients to spawn",
  'requests'    : "number of transactions to be executed",
  'clientTimeout' : "timeout for client requests",
  'testcase'    : {
    "type"      : "a field use to indicate the replica class to be used",
    "specialArguments": "a dictionary of special arguments such as random seed, etc..."
  }
},
```

The following is the test case template used for all the test cases. In the figure shown above, descriptions of all fields are provided.

## Generating Test cases

The test cases are generated using the testdiem.da file. The following code fetches the appropriate replica object.

```

def getReplicafromConfiguration(scenario):
    if scenario == "omission":
        return new(diem_replica_omission.Replica_Omission), self.specialArguments
    elif scenario == "normal":
        return new(diem_replica.Replica), {}
    elif scenario == "forge_signature":
        return new(diem_replica_forge.Replica), self.specialArguments
    elif scenario == "delay":
        return new(diem_replica_delay.Replica_Delay), self.specialArguments

def run:
    .....
    # spawning faulty replicas
    while i < self.faulty_replicas:
        replica, special = self.getReplicafromConfiguration(self.scenario)
        specialArgs[i] = special
        replicas.append(replica)
        i += 1

    # spawning normal replicas
    while i < replicas_required:
        replica, special = self.getReplicafromConfiguration("normal")
        specialArgs[i] = special
        replicas.append(replica)
        i += 1

    .....

```

The test cases are generated using the testdiem.da file. The following code fetches the appropriate replica object. The run method uses getReplicaConfiguration to use the appropriate replica based on the scenario specified. It can run multiple test cases one after another if the at-line position 382 in the code return value is an array instead of an element of array.

The non-faulty replicas which are equivalent to  $2*f + 1$  are invoked using the normal scenario.

## Logging files and Ledger Info

The logging file is generated based on the file specified during the command. using the command specified in the readme will generate a log in out.log file.

The ledger is stored on a flat file in the /tmp directory corresponding to diemLedger\_\*. (Here \* corresponds to the replica/validator ID).

To obtain the ledger after the test run. You can run the `testdb.py` file.

The following is a sample output :

```
(py36) vscode → /workspaces/diemBFT/DiemBFT/src (release x) $ python testdb.py
```

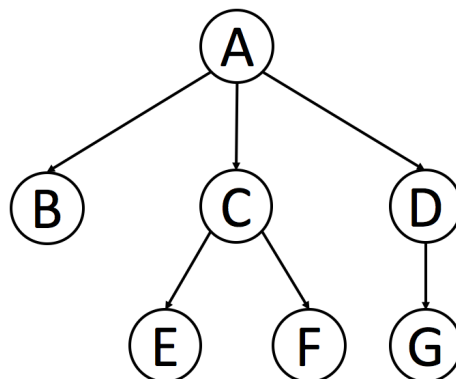
Commits for replica 0	Block transaction	Block ID
Parent Block ID		
0	0	0
0	1-0	65686032487e62dbd357b9e6f973523aca3e4b0736f677bfb3dc485401e8bd6d
012c26c83f31d17f4e8d1d147027813d785bab9c6eaf746a22cf5508a9f96da5	10-0	4d8ce626ce0683c34c39af47def4a8d5bf63ea2e697856fbc8e7e84e0d1ae695
65686032487e62dbd357b9e6f973523aca3e4b0736f677bfb3dc485401e8bd6d	2-0	a53c82a10849a921022acf56dad2baab36b39819c455b217c3c5a9dc5f2bf6b0
a53c82a10849a921022acf56dad2baab36b39819c455b217c3c5a9dc5f2bf6b0	3-0	48ebb55cfb53cec080c8bfcf23e4ed44147618a6bf69b17640fbf2a3b983322a
48ebb55cfb53cec080c8bfcf23e4ed44147618a6bf69b17640fbf2a3b983322a	4-0	e7c8d70689f93badad1bd820a405cb0c0a3be662f3c90d29cd617ce5b285a9d4
e7c8d70689f93badad1bd820a405cb0c0a3be662f3c90d29cd617ce5b285a9d4	5-0	246b1098c695e5357eefd5ea2fb0644696cc60341bb99701d34bc868490da165
246b1098c695e5357eefd5ea2fb0644696cc60341bb99701d34bc868490da165	6-0	959f5f47343c11c45efb8f80ba2ea04de24f4e5bab4c203d6b8541cfeef05826
959f5f47343c11c45efb8f80ba2ea04de24f4e5bab4c203d6b8541cfeef05826	7-0	35c8ed996a0116350cb91fe415a033a01f00655e1b97910cb381f5f353f99bbe
35c8ed996a0116350cb91fe415a033a01f00655e1b97910cb381f5f353f99bbe	8-0	0e2c3e9a4bd87e9c6373203079b88b1e3aacbd6eb69f75b22fe1e0947ad557b1
0e2c3e9a4bd87e9c6373203079b88b1e3aacbd6eb69f75b22fe1e0947ad557b1	9-0	012c26c83f31d17f4e8d1d147027813d785bab9c6eaf746a22cf5508a9f96da5

## Implementation

(This section is just to provide a detailed explanation about our design decisions and reasoning. For actual pseudo-codes refer to the pseudo-code.pdf).

### Pending Block Tree

The pending block tree is essentially a [trie](#)-like / n-ary tree data structure with the ability to perform pruning.



In the figure shown above A, B, C, etc are essentially block-ids. Each node contains a dictionary/map for its children. For example, A will contain B, C, D as its children, likewise C will contain E, F as its children.

For making the lookup / adding / pruning transactions quick we have used a cache on top of this. The cache has a pointer to the reference in the tree.

The following is the pseudo-code for adding a transaction into the tree.

```
def add(prev_node_id, block):
    node = self.get_node(prev_node_id)
    if node == null:
        node = root

    node.childNodes[block.id]=Node(prev_node_id,block)
    cache[block.id]=node.childNodes[block.id]
```

The following is the pseudo-code for pruning a node

```
def prune(self,id):
    curr_node = self.get_node(id)

    if curr_node == null:
        return
    self.root = curr_node
    self.cache_cleanup(id)

def cache_cleanup(id):
    cache = {}
    prune_helper(root)

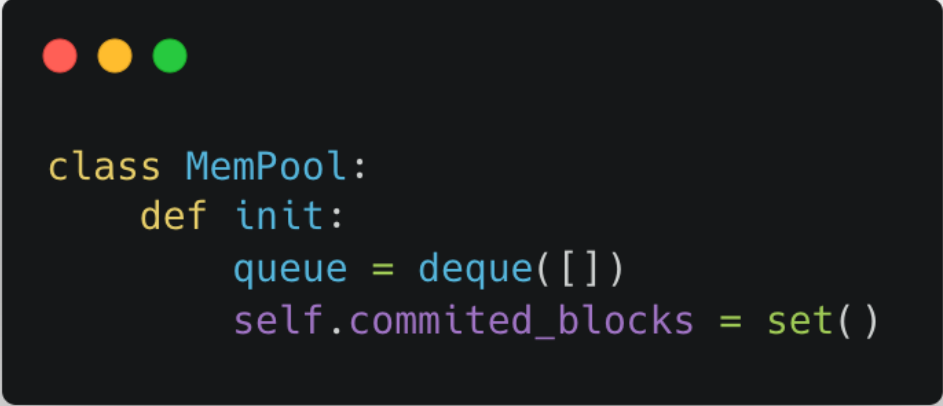
def prune_helper(node):
    if node is None:
        return

    cache[node.block.id]=node

    for block_id in node.childNodes.keys():
        cache[block_id] = node.childNodes[block_id]
        prune_helper(node.childNodes[block_id])
```


## MemPool

The mempool uses a queue for holding the incoming transactions. It also uses a set as a cache for committed blocks. This ensures in case a command gets re-transmitted for a committed block




```
class MemPool:
    def init:
        queue = deque([])
        self.committed_blocks = set()
```

Pseudo-code for getting a transaction



```
def get_transactions():  
    # currently only sends one transaction  
    if queue has transaction:  
        command = queue.pop()  
        return command  
    else:  
        return EmptyRequest()
```

Pseudo-code for inserting a command



```
def insert_command(self, command):  
    if command not in committed_blocks and command not in queue:  
        queue.append(command)
```

## Persistent Ledger

For ledger persistent, each replica is storing its ledger in a flat-file inside the temp directory. The API used for storing and retrieving blocks is using [LevelDB](#). For storage, we are using block\_id as the key and block as the value.

While writing we make use of the sync flag which ensures flush happens before execution is returned back to the validator.

Only the committed blocks are stored inside the persistent ledger.

## Speculative Ledger

The speculative ledger is similar to the persistent ledger. The only difference is that it stores the pending blocks as well. Thus in some way, it becomes storage of block to the block ids in the pending block tree.

Instead of keeping the speculative ledger in memory, we are using a flat file similar to a persistent ledger. This ensures the process does not run out of memory.

## Initiation of Diem Process and chain processing

The initiation happens with round -1. Leader election chooses replica 0 as the first leader just for this round and advances\_round\_qc. After which block 1 is proposed and a new QC gets generated after receiving required vote messages. After generating a new QC, replica 0 again makes a proposal, and the next leader, replica 1 waits for vote messages.

The reason why we use current\_round as -1 as opposed to round 0 as mentioned in the paper is to start with an even length cycle. If we start with 0, the vote messages for the proposal will be sent to replica 1 which is the next leader. (round 0 -> advance round to 1 -> broadcast proposal message -> send vote to next leader (current round 1 + 1) / 2 which will be replica 1).

## Genesis Block

The genesis block is the very first block in the ledger. It contains 0 as its block ID and references parent ID as itself. It is portrayed as being formed at round -1, likewise, its parent round is portrayed as -1 as well.

Similar to the genesis block, genesis QC is formed as well. Which contains no votes as signatures but has been authored as 0.

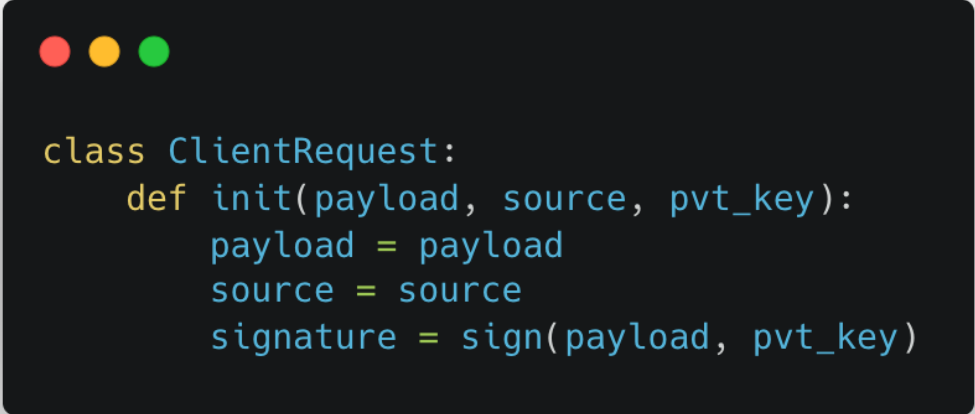
During leader election, there are conditions in place such that the genesis block or genesis QC is not used for electing reputation leaders.



## Client Request

The client request object holds the transaction as payload as well as the signature of the payload.

When the transaction is processed the client can request a replica to provide the committed block from the ledger and then use the payload to verify the signature with the original payload.



```
class ClientRequest:
    def init(payload, source, pvt_key):
        payload = payload
        source = source
        signature = sign(payload, pvt_key)
```

## Replica Info

The replica info object is used to provide the metadata about a replica to other replicas/clients. It contains the public key of the replica, the process ID used to send messages, and its replica ID.



```
class ReplicaInfo:
    def init(process, public_key, replicaID):
        process = process
        public_key = public_key
        replicaID = replicaID
```

## Chain Termination

Diem follows a 2-chain protocol where the Block proposed at round  $N$  will be committed at round  $N + 2$  at all replicas. In order to ensure the termination of all transactions sent by the client, we make use of 2 additional empty blocks. These dummy (empty) blocks serve as placeholders so that whatever client transaction has been sent gets committed. The only flaw with this approach is that the first dummy block will get committed at any one of the replicas. As QC for dummy block, 2 gets formed but not propagated as there are no further transactions in mempool.

As these dummy blocks are empty their presence has no impact on the state of the blockchain. For example: In a monetary blockchain, these blocks can be replaced with 0 value denominations. So a transaction like that will have no effect on the final state.

Another way of providing this functionality is to generate blocks whenever `get_transactions` is called. (This is the way the real-world Blockchain system maintains liveness). Another way is to generate the blocks and see if the last two generated blocks are dummy or empty blocks and terminate based on that behaviour.

In case no transactions are available for processing, the replicas go into an await where they wait for a transaction to appear to continue processing again. Round numbers will get incremented. This can be further optimized by not setting a timer for a round when the mempool is empty.

# Test Case Report - Our Use Cases

All configuration files are in testdiem.da. Logging is common as specified by the run command. Check the test setup section above for more details.

## Normal Execution Flow

```
{
  'Name'      : "Normal Replicas",
  'faultyReplicas': 1,           # number of replicas which can go faulty
  'timeoutDelta': 2500,         # milliseconds
  'clients'   : 1,              # number of clients to spawn
  'requests'  : 10,             # number of requests
  'clientTimeout': 5,
  'testcase'  : {
    "type"      : "normal",
    "specialArguments": {}
  }
}
```

The format is as defined in the test template above. This case does not have any byzantine behavior. The log file is as defined in the test setup above. The expected output is all 10 transactions to be executed and committed into the ledger. The resulting ledger state should look similar to the image below.

Commits for replica 0	Block transaction	Block ID
Parent Block ID		
0	0	0
6de1bc2975e6e83df62f97d8b852f05057834b0a484a0d7561d8d8ffb362864a	1-0	276997196f320c6a46bb4ddb9a418e31c0ab394f4cccd94d4e6f71a138da3ccb
276997196f320c6a46bb4ddb9a418e31c0ab394f4cccd94d4e6f71a138da3ccb	10-0	44df4292d52c6829a7b4b00240a17a33ca1483eab1145d16ccc0856f4e742df9
954049e08765d61c41271456f427dded547d6187a8d58dafcff1309b46b28feb	2-0	954049e08765d61c41271456f427dded547d6187a8d58dafcff1309b46b28feb
ace6e662ae8cddf77f3fc0a124256da670fcd809bf654dd4ca7856be5b829f7	3-0	ace6e662ae8cddf77f3fc0a124256da670fcd809bf654dd4ca7856be5b829f7
34a4dfd9379cf8cb7bbcbf1fa095fe4d8e440b1554a0b070ccb3ad8b6421c4c8	4-0	34a4dfd9379cf8cb7bbcbf1fa095fe4d8e440b1554a0b070ccb3ad8b6421c4c8
05559f5a01341cf698b303bdaba59847042f0c924cd1e494a827457bba2b462c	5-0	05559f5a01341cf698b303bdaba59847042f0c924cd1e494a827457bba2b462c
0e0ca932277ac6302660780e6b44dd8372fe0f4311e0dbfa7d0d2002bbe28f45	6-0	0e0ca932277ac6302660780e6b44dd8372fe0f4311e0dbfa7d0d2002bbe28f45
1485c31d570023126eee6917a733d0f9529bb8a1da651ac96b3b2b3548bb1142	7-0	1485c31d570023126eee6917a733d0f9529bb8a1da651ac96b3b2b3548bb1142
83fb9b14b299742cd220b1a3ae1cf99428f477b40a6c0c2b0f9e7f852ac67857	8-0	83fb9b14b299742cd220b1a3ae1cf99428f477b40a6c0c2b0f9e7f852ac67857
44df4292d52c6829a7b4b00240a17a33ca1483eab1145d16ccc0856f4e742df9	9-0	44df4292d52c6829a7b4b00240a17a33ca1483eab1145d16ccc0856f4e742df9
	empty	d1fb77910376c1a0b5b58d89bcf8e1a00289a7293d667f94c6129f4b61653ac0

## Normal Replicas with multiple clients

```
{
  'Name'      : "Normal Replicas with multiple clients",
  'faultyReplicas': 1,
  'timeoutDelta': 2500,
  'clients'   : 2,
  'requests'  : 10,
  'clientTimeout': 5,
  'testcase'  : {
    "type"      : "normal",
    "specialArguments": {}
  }
},
```

This is similar to normal execution flow, instead, here there are two clients which are provided.

```
(py36) vscode → /workspaces/diemBFT/DiemBFT/src (release x) $ python testdb.py
```

Commits for replica 0	Block transaction	Block ID
Parent Block ID		
0	0	0
0	1-0	abfe654d2af449d71f83d2d113109e69ea2d34de22705e9ac18c291208c3ba86
abfe654d2af449d71f83d2d113109e69ea2d34de22705e9ac18c291208c3ba86	1-1	134bdc7978baa76191e550dda2ea6b45f035a3b6fdfaecbadcbcd147a8439773
3315f2ce128418ce538399e828bf50fc9174e484661193aa0e3aa05f91e7b15	10-0	57c849d5be42242d8c711b5efabd96590ee78ecf6a8df189715a6cd29c0a235
79af4ffdea0a7140d6bd250f3fdd88fae219efaaae70475d266d4caa82647f8b	10-1	8a0f444b00a66ae453e7b9e034175476b9fee5a021b4b84cc07f9a6a218a9007
d839b90ef624bcd70a5b8bc293e6bf7c9afdf1490a44b65e895285501e6615a1f	2-0	1b26d6317d23284476251b0d97f98614b88cf41a9360b5d8897c7063bbec6252
134bdc7978baa76191e550dda2ea6b45f035a3b6fdfaecbadcbcd147a8439773	2-1	d839b90ef624bcd70a5b8bc293e6bf7c9afdf1490a44b65e895285501e6615a1f
b6cda38a0aad2f8f1a8a28a37384bf6cee242db625971fb414d358a903dd4c75	3-0	fab77da5879ae4479e0e4fc11470da8068c3b433d8f54ed754959c88904db347
1b26d6317d23284476251b0d97f98614b88cf41a9360b5d8897c7063bbec6252	3-1	b6cda38a0aad2f8f1a8a28a37384bf6cee242db625971fb414d358a903dd4c75
4ed501340dadeea8694d50a8d652ecaf10b8606e2ecf207611b0509cd2806d5	4-0	d9b346099149c6763d718782eb17b57d4ca32f49cec985eb2e92876c47a89381
fab77da5879ae4479e0e4fc11470da8068c3b433d8f54ed754959c88904db347	4-1	4ed501340dadeea8694d50a8d652ecaf10b8606e2ecf207611b0509cd2806d5
85ad58b0973f5d794e73882417c6920cb207a11597850b3e6fd7960955940ffc	5-0	0582ee4be28c72ad485ed27548c35782cb3118edf3b4229f1b8c615e492643dd
d9b346099149c6763d718782eb17b57d4ca32f49cec985eb2e92876c47a89381	5-1	85ad58b0973f5d794e73882417c6920cb207a11597850b3e6fd7960955940ffc
0582ee4be28c72ad485ed27548c35782cb3118edf3b4229f1b8c615e492643dd	6-0	099598db92f719badec7fe8462520b9f004c87c0c1251e15f67bd170d108e0fc
099598db92f719badec7fe8462520b9f004c87c0c1251e15f67bd170d108e0fc	6-1	175c90d4fc17b13209199faa4ba5816adca2a769d054302060d36f4fcb20c46
175c90d4fc17b13209199faa4ba5816adca2a769d054302060d36f4fcb20c46	7-0	dcf3c7165e7b80014bf2e1c03c057f24b7a9d3e2efb085f1035ead6220197bdb
dcf3c7165e7b80014bf2e1c03c057f24b7a9d3e2efb085f1035ead6220197bdb	7-1	f40a467f51e6a01d0afc53136fd6419892d571f8dc305ae74c07e00de94cfa75
f40a467f51e6a01d0afc53136fd6419892d571f8dc305ae74c07e00de94cfa75	8-0	8f459ecfd8fc0d6ae1eb6c88d7f97ff92bb414618b883d8bf00ee93056a99e4
8f459ecfd8fc0d6ae1eb6c88d7f97ff92bb414618b883d8bf00ee93056a99e4	8-1	0d39dbd170326102c567c6fd14eb88b10eadf41a392e91b8ed75367026b78980
0d39dbd170326102c567c6fd14eb88b10eadf41a392e91b8ed75367026b78980	9-0	c709d09a0f619abd020ae2d5d88a7f2673ed249f2df4553cc0605e0074af29ce
c709d09a0f619abd020ae2d5d88a7f2673ed249f2df4553cc0605e0074af29ce	9-1	57c849d5be42242d8c711b5efabd96590ee78ecf6a8df189715a6cd29c0a235
57c849d5be42242d8c711b5efabd96590ee78ecf6a8df189715a6cd29c0a235	dummy1-0	c709d09a0f619abd020ae2d5d88a7f2673ed249f2df4553cc0605e0074af29ce

## Client requests timeout, re-submission, and handling request de-duplication

```
{
  'Name'      : "Client small timeout with request resubmission and handling de-deuplication",
  'faultyReplicas': 1,
  'timeoutDelta': 2500,
  'clients'   : 1,
  'requests'  : 10,
  'clientTimeout': 0.5,
  'testcase'  : {
    "type"      : "normal",
    "specialArguments": {}
  }
},
```

In this test case we set the client timeout to be very low i.e 500ms due to which, the client keeps resending the request on timeout. On receiving a duplicate client request the replicas should handle them and only commit the unique ones.

```
(py36) vscode → /workspaces/diemBFT/DiemBFT/src (release x) $ python testdb.py
```

Commits for replica 0 Parent Block ID	Block transaction	Block ID
0	0	0
0	1-0	5334759456abea3d021606c7f8fef2566643ed3fbec777bf8a2068997971fc93
92e8d6073ee32142c9f3a6c9031614cdab4d5c058d1fe2eb9450108573e72b20	10-0	8c8e4b202fc9e69761d63d045418eca674743fa2d5790824a8dc4d6b28d388e9
5334759456abea3d021606c7f8fef2566643ed3fbec777bf8a2068997971fc93	2-0	4fc1c03add94a587b408c8bad051827fb081ce7db666566aa0c3976617e6d3d7
4fc1c03add94a587b408c8bad051827fb081ce7db666566aa0c3976617e6d3d7	3-0	33f12e3c1040c0c47b111af992bb9424114f097928a8c3822de877d4d1916407
33f12e3c1040c0c47b111af992bb9424114f097928a8c3822de877d4d1916407	4-0	1d3781f41c8798ba1ffbf5a5ac1e47cb46742957ef0d64746e7a2aae205153bda
1d3781f41c8798ba1ffbf5a5ac1e47cb46742957ef0d64746e7a2aae205153bda	5-0	f5f728ae0fd7c9372884d26946458359fbb6217ed4ff526c05b2e29b2be49cc3
f5f728ae0fd7c9372884d26946458359fbb6217ed4ff526c05b2e29b2be49cc3	6-0	e9ed28f5dff94ca114d9e6082360bd78fba1bff734d183cecb28674161b16ee3
e9ed28f5dff94ca114d9e6082360bd78fba1bff734d183cecb28674161b16ee3	7-0	f4d8271f0b16c765a0566609ef291465e575439a08ee6fa98e72ca9ebb65e48c
f4d8271f0b16c765a0566609ef291465e575439a08ee6fa98e72ca9ebb65e48c	8-0	c98a1a9a103515c87fff8cca4df47e232e9c1132ce68ce4723cfba76fe3f667f
c98a1a9a103515c87fff8cca4df47e232e9c1132ce68ce4723cfba76fe3f667f	9-0	92e8d6073ee32142c9f3a6c9031614cdab4d5c058d1fe2eb9450108573e72b20

## Omission Failure

```
{
  'Name'      : "Faulty replica having omission failures",
  'faultyReplicas': 1,
  'timeoutDelta' : 2500,
  'clients'    : 1,
  'requests'   : 5,
  'clientTimeout' : 5,
  'testcase'   : {
    "type"      : "omission",
    "specialArguments": {}
  },
}
```

In the following test case the faulty replicas cause omission failures but do not provide a Vote Message. This indicates the tolerance of DiemBFT to delays in the network.

```
(py36) vscode → /workspaces/diemBFT/DiemBFT/src (release x) $ python testdb.py
```

Commits for replica 0	Block transaction	Block ID
Parent Block ID		
0	0	0
0	1-0	dd131503e2a6ba571d70abccba82f67ea49663269f9f6c1924907256fe3ffef1
dd131503e2a6ba571d70abccba82f67ea49663269f9f6c1924907256fe3ffef1	2-0	df5091e16d6f062d5c1df9cebb72bc61e1317c30cf8c854fc72e3ebc12134db0
df5091e16d6f062d5c1df9cebb72bc61e1317c30cf8c854fc72e3ebc12134db0	3-0	a52e27aea4a73b35ef6b35e0958be4aa08827db59aae0814a12ec81a1cecf7d9
a52e27aea4a73b35ef6b35e0958be4aa08827db59aae0814a12ec81a1cecf7d9	4-0	b38190ae961321774b2098e41cb695f1fb4ec3d3e41dd58de0b35e3d29039068
b38190ae961321774b2098e41cb695f1fb4ec3d3e41dd58de0b35e3d29039068	5-0	79e709099801de37be024c002afde54b56566230acc517cd4fb8c17e06211c3d

# Forge Signature

In the following test case, the faulty validator tries to forge the QC signature. Which causes a timeout and later leads the chain to recover.

```
{
  'Name'      : "Forge signature",
  'faultyReplicas': 1,
  'timeoutDelta': 500,
  'clients'   : 1,
  'requests'  : 5,
  'clientTimeout': 2,
  'testcase'  : {
    "type"      : "forge_signature",
    "specialArguments": {}
  },
},
```

All ledger entries will be committed. As shown in the above images.

Commits for replica 3		
Parent Block ID	Block transaction	Block ID
3	0	0
3	1-0	1708a78df391c47ba9bd3a91aec2418941599fe5160be1e9e76461298e8c115c
a641429a906a14a5637de2518d80a94fcb8c1aec4f2a7327eaa9514a5f6349f4	1-0	2eddb844d184e723499ed05b2897c4e3626508adefef052f644e4fb1893d649cc
2eddb844d184e723499ed05b2897c4e3626508adefef052f644e4fb1893d649cc	2-0	3afe115c7e201db06b90255b1b14424880ea84160f1853fe0bf726dd571efa73
1708a78df391c47ba9bd3a91aec2418941599fe5160be1e9e76461298e8c115c	2-0	c27e8f708c44b3cd486fa7aae59ce4112f2859f5113f2bec61fce55e47ca45dd
3	3-0	beca0a1f9e9f100a9cffbea8986232fdff2886c1855758eeff0da7dd84d576d1
c27e8f708c44b3cd486fa7aae59ce4112f2859f5113f2bec61fce55e47ca45dd	3-0	f280a6aaa5e55445c33c560e6d27a6c83fedd037019aff9a41f23920af04a77
beca0a1f9e9f100a9cffbea8986232fdff2886c1855758eeff0da7dd84d576d1	4-0	12caaa7d880e0cd5acf4ec83c0714a53c0453da660c08b33da3bfe418fd46e9
f280a6aaa5e55445c33c560e6d27a6c83fedd037019aff9a41f23920af04a77	4-0	db39cb7ea6d01f7bc894204975c133d7d9e471620e5418a4c9892b46640ffb91
db39cb7ea6d01f7bc894204975c133d7d9e471620e5418a4c9892b46640ffb91	5-0	856c638c40c0914cda5b013f36cf6c5a6e843d9d4db3c3072b466c7f83dbf26d
12caaa7d880e0cd5acf4ec83c0714a53c0453da660c08b33da3bfe418fd46e9	5-0	a641429a906a14a5637de2518d80a94fcb8c1aec4f2a7327eaa9514a5f6349f4
3afe115c7e201db06b90255b1b14424880ea84160f1853fe0bf726dd571efa73	empty	10f0f109a6f71eabced02627f9cccec9b5ba4515cbc03cba315a145fbcda9aca

## Delay Failures

```
{
  "Name"      : "5. Faulty replicas having delay failures",
  "faultyReplicas": 1,
  "timeoutDelta" : 2500,
  "clients"    : 1,
  "requests"   : 5,
  "clientTimeout" : 5,
  "testcase"   : {
    "type"      : "delay",
    "specialArguments": {
      "randomSeed" : 50,
      "faulty_round": 1,
      "start": 4,
      "end": 7
    }
  }
}
```

In delay failure, we simulate network delay using a random seed to timeout a message which is to be sent.

All ledger entries will be committed as shown in the above images.

Commits for replica 3		
Parent Block ID	Block transaction	Block ID
0	0	0
0fa117445d4ff1b614ab028fa9b67b14d8b9f8d442b357baeb0463f04cd19d5e	1-0	0fa117445d4ff1b614ab028fa9b67b14d8b9f8d442b357baeb0463f04cd19d5e
583bf3d086a31fbd6c7a405b9c4ad3a7f9322b2e87a54abbf2a904488eadbcae	2-0	583bf3d086a31fbd6c7a405b9c4ad3a7f9322b2e87a54abbf2a904488eadbcae
5593e8a830836324b6059401fc1bb200a79c925b78ca26644902cdafd349cc89	3-0	5593e8a830836324b6059401fc1bb200a79c925b78ca26644902cdafd349cc89
a1f3bd3c807149792b36c4cabbdee36f9db64175db44635863f02e9184a47880	4-0	a1f3bd3c807149792b36c4cabbdee36f9db64175db44635863f02e9184a47880
	5-0	3db653b9ecc15731cb5e935abb360a6f3e4153dc4bf94a8d7b63cf447bd769ba



# Test Case Report - Phase 2 Test Cases

## Proposal Drop

In the third round, proposal messages to validators 1 and 2 are dropped from the leader

```
{#6
  "Name"      : "Proposal Messages Dropped at validators 1 and 2",
  "faultyReplicas": 1,#2 Please change in File diem_replica_proposalDrop.da line 153
  "timeoutDelta" : 2500,
  "clients"      : 1,
  "requests"     : 5,
  "clientTimeout" : 5,
  "testcase"     : {
    "type"       : "ProposalDrop",
    "specialArguments": {
      "faultyRound" : 2
    }
  }
}
```

## Proposal Delay

In the third round, proposal messages to validators 1, 2, and 5 are delayed.

```
{
  "Name"      : "Proposal Messages Delayed at validators 1 and 2 and 5",
  "faultyReplicas" : 2,
  "timeoutDelta" : 2500,
  "clients"    : 1,
  "requests"   : 5,
  "clientTimeout" : 5,
  "testcase"   : {
    "type"      : "ProposalDelay",
    "specialArguments": {
      "randomSeed" : 50,
      "faultyRound" : 2
    }
  }
}
```

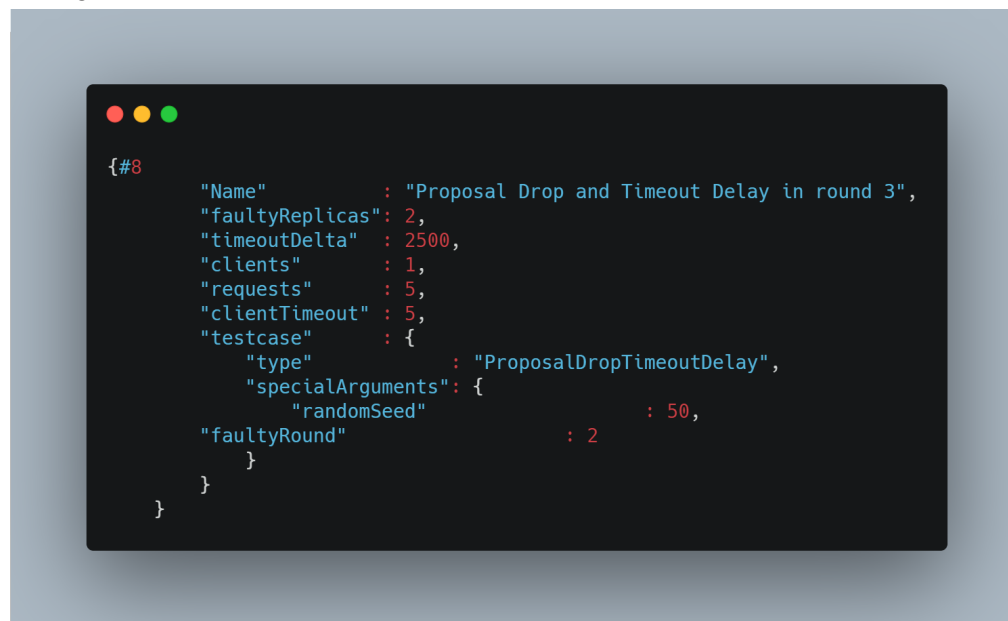
## Proposal Drop + Timeout Delay

In the third round, all proposal messages are dropped, and all timeout messages are delayed.

Ledger entries

Commits for replica 0	Block transaction	Block ID
Parent Block ID		
0	0	0
44fc7c7d7895f95c71366b7882920ecbe62281ff5a0d841f1007b56df53b79e7	1-0	bd54cf0a8dbc6ef63d5a97ba585fb1106f08bd2409221fe2a0cce0d45cc1557f
f9661cc26ffcb66598b360f1f7409181bdf34025b3c8cd4c4015f5bcc36dfc	1-1	7f5c319608f0b9c5e7ff3ac2154d2017c2ad56bb74aefb3d77d3dcf5edc2d8db
b67d960b51a68893092f8ed41bf931b3728fa1831ad106cd721ba08f743095f2	1-2	44fc7c7d7895f95c71366b7882920ecbe62281ff5a0d841f1007b56df53b79e7
0	1-3	b67d960b51a68893092f8ed41bf931b3728fa1831ad106cd721ba08f743095f2
7f5c319608f0b9c5e7ff3ac2154d2017c2ad56bb74aefb3d77d3dcf5edc2d8db	1-4	7b750541b03ce33dd39db80cdf8471c440ef091e5561d99663f7778e3b10212e
cac91e9d3984a474694abf994981edcfa2fc44e816ca6bac2308a21f9cbc94ff	2-0	7b1d429cd7e7e9bbd3b46cb9d21c63275edaa0907cc544cb4dced445230e0d3d
9b0e7d233b26dcc37b89fdd563df93d51fe9f420185cdbc7870ae6bb822b1784	2-1	108e05b5cb36fea3f3c0dd7835761d831ab7d4419b3ee2d37c6f889800954c01
4237d73b60816f83c6f94674d64a5c854e513bea8ef194d947e3041464261ac3	2-2	660b31870a7c41ee57169c4e18d0779bb30d62a5eb64a4696c0b8572adca5f56
bd54cf0a8dbc6ef63d5a97ba585fb1106f08bd2409221fe2a0cce0d45cc1557f	2-3	b74a33deeffecda9cf899c068aeb0425ecd9636234a2865909aa727c5f7a84a
3b398319fd8fd81c578f7bb6148097cd546480e4d5013f38f36afc0d0a5e6a95	2-4	8742062f7acc2cf1ea9aee3f929c809dbd047ab9cc555e9587378e4a21a7c949
7b1d429cd7e7e9bbd3b46cb9d21c63275edaa0907cc544cb4dced445230e0d3d	3-0	20b23390f4932869e561359fdd016cd6241bcee6f256026342c1fa76920f5f4a
8742062f7acc2cf1ea9aee3f929c809dbd047ab9cc555e9587378e4a21a7c949	3-1	8b3a52f9ed3acda8f189f4bf61ead68d0b0f4914c27470299370af3afa0c81e9
660b31870a7c41ee57169c4e18d0779bb30d62a5eb64a4696c0b8572adca5f56	3-2	e29eb97cabf7e6741eb4ebe4cd63ad6c2a9278e32ab7dce79ae10b638ccc1a3f
b74a33deeffecda9cf899c068aeb0425ecd9636234a2865909aa727c5f7a84a	3-3	4373d73b60816f83c6f94674d64a5c854e513bea8ef194d947e3041464261ac3
8b3a52f9ed3acda8f189f4bf61ead68d0b0f4914c27470299370af3afa0c81e9	4-0	29145b1ff79a130593ff706b820d678e1565b7f50bedf2bc73a4a5825575a5f7
20b23390f4932869e561359fdd016cd6241bcee6f256026342c1fa76920f5f4a	4-1	9b0e7d233b26dcc37b89fdd563df93d51fe9f420185cdbc7870ae6bb822b1784
29145b1ff79a130593ff706b820d678e1565b7f50bedf2bc73a4a5825575a5f7	4-2	45754040f1fc4a62fe9b14c78c12add4e6a0250246ea9ce2dbcd07d036850a5
32fa37cb582a7b574a672303c81fa8998dc9be86ba605388f912c98eb03e419a	4-3	bfe0235f50bf6bdb1e6bce845bf0621f22e41e067d83717a12ce5fafaebcf87db
e29eb97cabf7e6741eb4ebe4cd63ad6c2a9278e32ab7dce79ae10b638ccc1a3f	4-4	f9661cc26ffcb66598b360f1f7409181bdf34025b3c8cd4c4015f5bcc36dfc
45754040f1fc4a62fe9b14c78c12add4e6a0250246ea9ce2dbcd07d036850a5	5-0	8204b26b6e23657303675f9932015ba2e9c489c2e180bd18179117780ad9c9e6
108e05b5cb36fea3f3c0dd7835761d831ab7d4419b3ee2d37c6f889800954c01	5-1	3b398319fd8fd81c578f7bb6148097cd546480e4d5013f38f36afc0d0a5e6a95
8204b26b6e23657303675f9932015ba2e9c489c2e180bd18179117780ad9c9e6	5-2	c6a6833fd0b8b1d2c6ebe554e16b11eb508f378639fcb11b4c900ef80a126ae
bfe0235f50bf6bdb1e6bce845bf0621f22e41e067d83717a12ce5fafaebcf87db	5-3	ca91e9d3984a474694abf994981edcfa2fc44e816ca6bac2308a21f9cbc94ff
7b750541b03ce33dd39db80cdf8471c440ef091e5561d99663f7778e3b10212e	5-4	32fa37cb582a7b574a672303c81fa8998dc9be86ba605388f912c98eb03e419a
c6a6833fd0b8b1d2c6ebe554e16b11eb508f378639fcb11b4c900ef80a126ae	empty	3b370a6f4878e78829461c5020b3a61f07f47540adfe997bb5f35855f0b5e448
d804fe9672be6b913acfdaa720228a52ebfd0df72832ce7ca320dcel1a0d47ccd	empty	6a4f029e85ec6b3c80f465789e8da00a468a6c1c0197f23b6411b5405273634f
6a4f029e85ec6b3c80f465789e8da00a468a6c1c0197f23b6411b5405273634f	empty	772516b591dd0a0fa0b158791789af6f3f240b6f80f520a62b095ac6956b1e3b
3b370a6f4878e78829461c5020b3a61f07f47540adfe997bb5f35855f0b5e448	empty	d804fe9672be6b913acfdaa720228a52ebfd0df72832ce7ca320dcel1a0d47ccd

## Configuration



## Vote Drop

In the third round, vote messages from validators 3 and 4 are dropped or delayed. All requests get committed.

```
{
  "Name": "9. Vote Message drop for round 3",
  "faultyReplicas": 2,
  "timeoutDelta": 2500,
  "clients": 1,
  "requests": 5,
  "clientTimeout": 5,
  "testcase": {
    "type": "VoteDrop",
    "specialArguments": {
      "randomSeed": 50,
      "faultyRound": 2
    }
  }
}
```

## SetAttr Failure

In round 1, a SetAttr failure with src=3, dest='\_', type= MsgType.Vote, val=4, attr='current\_round'. In other words, when validator 3 sends its vote in round 1, it sets Pacemaker.current\_round to 4.

```
{#10
  "Name"      : "SetAttrFailure",
  "faultyReplicas" : 2,
  "timeoutDelta" : 2500,
  "clients"    : 1,
  "requests"   : 5,
  "clientTimeout" : 5,
  "testcase"   : {
    "type"      : "SetAttrFailure",
    "specialArguments": {
      "randomSeed" : 50,
      "faultyCurrentRound" : 4
    }
  }
}
```

## Failures in multiple rounds


### Proposal Drop Twice

Number of requests per client = 5. in the third and fourth rounds, proposal messages to validator 4 are dropped

```
{
  "Name": "11. Proposal drop twice",
  "faultyReplicas": 1,
  "timeoutDelta": 2500,
  "clients": 5,
  "requests": 5,
  "clientTimeout": 5,
  "testcase": {
    "type": "proposaldroptwice",
    "specialArguments": {
      "rounds": [
        3,
        4
      ],
      "validator": [
        4
      ],
      "subtype": "proposaldrop"
    }
  }
}
```

## Vote Delay Twice

In the second and third rounds, vote messages to validators 1 and 3 are delayed.



```
{
  "Name": "12. Vote delay twice",
  "faultyReplicas": 1,
  "timeoutDelta": 2500,
  "clients": 5,
  "requests": 5,
  "clientTimeout": 5,
  "testcase": {
    "type": "votedelay",
    "specialArguments": {
      "rounds": [
        2,
        3
      ],
      "validator": [
        1,
        3
      ],
      "subtype": "votedelay",
      "randomSeed": 50,
      "start": 4,
      "end": 7
    }
  }
}
```

## Probabilistic Failures

Probabilistic Drop with  $f=2$ : number of requests per client = 20. seed=1234567 for the PRNG (for failure probabilities). drop failures in rounds 2, 3, 5, 6, 8, and 9 with src = '\_', dest = '\_', type=MsgType.Wildcard, probability=0.15.

```
{
  "Name": "13. Probabilistic Drop with f=2:",
  "faultyReplicas": 2,
  "timeoutDelta": 2500,
  "clients": 5,
  "requests": 20,
  "clientTimeout": 5,
  "testcase": {
    "type": "probablisticdrop",
    "specialArguments": {
      "rounds": [
        2,
        3,
        5,
        6,
        8,
        9
      ],
      "subtype": "probablisticdrop",
      "randomSeed": 1234567,
      "start": 4,
      "end": 7,
      "probability": 0.15
    }
  }
}
```



Commits for replica 0		
Parent Block ID	Block transaction	Block ID
0	0	0
a8a32ba8d19dde87e27e6e71469ad192d188692811c5b643ca78f6bca5de34a0	1-0	80fc107351bde6efa508e308e39f08959d129c98b85d24ccf40c0f018e4657f4
0	1-1	ce48953307adec5cb632a87ec5d6a5db2b9906aec23a4e6ff55f11a0b4bef5c3
ce48953307adec5cb632a87ec5d6a5db2b9906aec23a4e6ff55f11a0b4bef5c3	1-2	a8a32ba8d19dde87e27e6e71469ad192d188692811c5b643ca78f6bca5de34a0
8f212127071367b97dcb5a766b23627a0245ce5985bbde6de33a470262a61c64	1-3	7030e19e602c608a2e782e898286ac70b0219cd37cb24e560f7d7620aed352b0
7030e19e602c608a2e782e898286ac70b0219cd37cb24e560f7d7620aed352b0	1-4	01ee94eb0bf600d5717af74078fe8249ad407572188edea347651e7645602044
0f0444cfea4454e2a468580509fa02d2c4f137dc68f7517539a8be7d8be487a7	10-0	9f1fcb8de2a399f38346942695e445571a6dac2a3fefcbcc0da8eb426e41056c
44817ab63999ea392aa8d6d4b80bf39068c15bba99790ae8ffa5da985dea76bd	10-1	cl720972e9297c871f4e2c46b482f484077cbe81874811f5fc1065c718a85946
541830e482153940a014d38cb5f3fe3d53d63f316ee5a1fee651ee12d234d127	10-2	fd0f025d13378d376d08577001acbbf5e35795353fab304f545d9fe793ef4c442
751279e82172021670cf5c8f69d0dc2ff5dad4b5ac274a21618e369861dlbfa8	10-3	fe820967a1fe7959a2b98fd5fce169b1994ef93b21a64407a1cb01a9239ac3ba
0bf1dcb7f5a5ba0a9acd24d464b991246c55d2d4b08b9e8124dcd1ec9f822ad	10-4	17b746adf768eb8eb39c97c3408755ace74ac33849cd98bb4f4f3c8b288fbc1c6
1e0450c8224857cc11a66762761fde25d5f4fc22f9f6a4fba91657e285a36157	11-0	fecf09519dc3937b93dc6f6bb3d31b20bcb1f9372fcd9557b0d2a8a929dd9cc2
fdf025d13378d376d08577001acbbf5e35795353fab304f545d9fe793ef4c442	11-1	75c102810f78189ffff39443f80984a5da2a987f45111b57200eb12308d484b44
7f34e24fe0689b640cee5541a077be8e9a9382290abebaf4766688e95c8c34de	11-2	92900066cba28577da5caab912216bf4d31b3f5df5f7676249fc899c578b4a56e
17b746adf768eb8eb39c97c3408755ace74ac33849cd98bb4f4f38c208fbc1c6	11-3	d53e9a6dbb9077ca71cafed5f872a78cf37f293045a40cddb2169bbd379496ab
25d2a918f94b4c1d07cbf368a39d336d60b3441ed99b9a72aea0aa8700067b4	11-4	5a32d1f2dd61cb6695d9552de999b138f679b9589c42f8b842508dde7e2a9e0b
5e9d063d6cbf1ec9608891f9e7807bc623dd9d1ff34b8de65fab1180a76abaa1	12-0	5d7e6e926c2d38dff26202589e60c0d62571fdadd8b2b6dd28b0100582e41f3e
92900066cba28577da5caab912216bf4d31b3f5df5f7676249fc899c578b4a56e	12-1	3b8087c74b2cf66f830c0f0ffe5f2293272b1a263427e9edd096f8e15c02550
0f62e37540f4492f20cdc372290cdddfc9738265fddb2d0af6b4b91402fb70	12-2	f965b785f769a0a0204ed827c42dc3b39cc534e7e85b7d77022701c62c142e0
f4857d0bee8b9c9a261cb9e17556181c3808fcd5d698201f6c14b3a1c41e0c62	12-3	25d2a918f94b4c1d07cbf368a39d336d60b3441ed99b9a72aea0aa8700067b4
c742afcad5a65585e6ac628f598f8223877ff5fe46289e689659900691ae2cda	12-4	ee6ce7e65e6ea219443895c9377a090ecaa59119ea47fd78b5d2583e0ca16da5
fe820967a1fe7959a2b98fd5fce169b1994ef93b21a64407a1cb01a9239ac3ba	13-0	287f44496559a9b83210c010344262da25c075a0828fe1ea5768a75ff44fbc0b
f965b785f769a0a0204ed827c42dc3b39cc534e7e85b7d77022701c62c142e0	13-1	30cb4c8a6252f66b7cf28b7a69ed8ff327b16e965376f96d7e3570612a721459
9f1fcb8de2a399f38346942695e445571a6dac2a3fefcbcc0da8eb426e41056c	13-2	4168f102800f8673185afc3b9be8595d0644a2b8e9dddf2e8213f452ca8cb321
e66d3028a31bdd4546e1cec7ad64ee359d6d42f5979c2a9031c68a427829634	13-3	1d5555cf609e8a4f8fe7e4f0e8abea53554063e3d1f875856955c386694661ac
ee6ce7e65e6ea219443895c9377a090ecaa59119ea47fd78b5d2583e0ca16da5	14-0	2ccb5f4bc441f0e98855c96ff555fedeeaaef854292ef9511edc6eac6eaf729d
d53e9a6dbb9077ca71cafed5f872a78cf37f293045a40cddb2169bbd379496ab	14-1	f4857d0bee8b9c9a261cb9e17556181c3808fcd5d698201f6c14b3a1c41e0c62
146d542c9b50a3d2f7df121a79c16374ee1fa9b0568ef771e2d6b925f9dc3c	14-2	d7ac1eba8e5bf8bdff510db3c89f660513d1fd3499f5850e9955b428c1599b3dd
5d7e6e926c2d38dff26202589e60c0d62571fdadd8b2b6dd28b0100582e41f3e	14-3	751279e82172021670cf5c8f69d0dc2ff5dad4b5ac274a21618e369861dlbfa8
1d5555cf609e8a4f8fe7e4f0e8abea53554063e3d1f875856955c386694661ac	14-4	e3d34f071147497d7b83ca0db4998992254ab5cd3a01de27d2743092be529c6d
2ccb5f4bc441f0e98855c96ff555fedeeaaef854292ef9511edc6eac6eaf729d	15-0	27077836b1afae4569d28eafeb94cf6d146e3be73d38ea11c8f3d3ed60800bfc
5a32d1f2dd61cb6695d9552de999b138f679b9589c42f8b842508dde7e2a9e0b	15-1	c742afcad5a65585e6ac628f598f8223877ff5fe46289e689659900691ae2cda
d7ac1eba8e5bf8bdff510db3c89f660513d1fd3499f5850e9055b428c1599b3dd	15-2	bf95c00cdd2c18c59e26dd8d91594eb62be94d835fd9e4d9d03c1cd88ae4b2edc
bd8dab18b77c86453f94ed40b7a7e9453db5f6fcbb1ed5322305588208cbe00c	15-3	0bf1dcb7f5a5ba0a9acd24d464b991246c55d2d4b08b9e8124dcd1ec9f822ad
e3d3d70711d7497d7b83ca0db4998992254ab5cd3a01de27d2743092be529c6d	15-4	38470bb8af9d28b53ff59b7d0df00e6ea2dc2c19c131bbc6d19caea1eda00edf
27077836b1afae4569d28eafeb94cf6d146e3be73d38ea11c8f3d3ed60800bfc	16-0	e66d3028a31bdd4546e1cec7ad64ee359d6d42f5979c2a9031c68a427829634
0595e3ff52b09b4667609f02d92baa4607435bbcd6ce978de2807c5f5f0cf751	16-1	039fe49b87507c87fb97a5344de18b9ecd0bdff07193b3c94ac20a0da6059d60c
38470bb8af9d28b53ff59b7d0df00e6ea2dc2c19c131bbc6d19caea1eda00edf	16-2	6733bf42bb31f4212a28c1c8aec05e2348b618ac407d7821394350bf7e0c789
479364bd056ec534448c793369d98c90b2240b3be5b965a9d85994dc2351de8b	16-3	d3fa42dc5b928b94d14d24a65d82f98de63dea12ea48a9b9481f7a765631e924
58394650277381e949c0b62e89e10cfeca0cee6af4388b8f79409aa6d14bf4c1	16-4	8a1f1eb23c359e3366fab47824d57895b2ab089a477ae65f185023523beef94
5b1d1bdc4deabe435d41321db4f37d72a86f1a96c8277507cab9f55da620198	17-0	1164e2409456e513790ee818f5f62d7c7fc72da0513332847b3853d87eb274fe
039fe49b87507c87fb97a5344de18b9ecd0bdff07193b3c94ac20a0da6059d60c	17-1	a4c3cbacc9abd7ec857285d8bf15da5638045fe7fd7546776e21bee9cddb1883
6733bf42bb31f4212a28c1c8aec05e2348b618ac407d7821394350bf7e0c789	17-2	6aed54ca4f2e11de58dbb85bbffec8429753edf5b6fbd4a800bc7624b76d40f
d3fa42dc5b928b94d14d24a65d82f98de63dea12ea48a9b9481f7a765631e924	17-3	0dd02b542ba3e2a4dda2329cc3283511de8c893862446dec4339df9f91fb21c8