TwinBFT Documentation and Test Report

Team Name: RVP
Raj Patel, 114363611
Prince Kumar Maurya, 114354075
Vishal Singh, 114708875

Scenario Generation

As explained in the **phase_3/pseudo_code/scenario_generator** file, our **twinBFT/src/scenario_generator** generates scenarios and stores them in the config folders as **twin_*.json** file. These json files are passed as parameters to the scenario_executor as mentioned in the User Manual.

To generate the scenario run below command from the

```
### Command to run the scenario executor code from the source folder
- ```   python scenario_generator.py   ```
```

The above command will generate 5 scenarios which will be stored in twinBFT/Loyal_Byzantine_Generals_phase_2/config folders as twin_*.json files

A notable upgrade from our submission in phase3 is the provision to **drop/delay** type based messages from validators. The generated scenario with drop/delay provision is explained in the Configuration section.
The logic for drop/delay is explained in the pseudocode int the **drop_delay** function

```
Function drop_delay(
    total_nodes
  ) {
    //drop_sict contains information for Message drops
    drop_dict = {}
    //delay dictionary contains information of delay messages
    delay_dict = {}

    for i in range (1, R+1):
        drop_dict[i]["Vote"]=[]
        drop_dict[i]["Proposal"]=[]
        drop_dict[i]["Timeout"]=[]

    //Randomly populate the Drop Information for each round for each message type
```

```
    for r in range (1, R):
        for n in range(0, total_nodes):
            if randint(0, 4) == 1:
                drop_dict[r]["Vote"].append(n)
            if randint(0, 4) == 1:
                drop_dict[r]["Proposal"].append(n)
            if randint(0, 4) == 1:
                drop_dict[r]["Timeout"].append(n)
    drop_dict={"drop_round_msg":drop_dict}

    //Randomly populate the Drop Information for each round for each message type
    for i in range (1, R+1):
        delay_dict[i]["Vote"]=[]
        delay_dict[i]["Proposal"]=[]
        delay_dict[i]["Timeout"]=[]
    for r in range (1, R):
        for n in range(0, total_nodes):
            if randint(0, 4) == 1:
                delay_dict[r]["Vote"].append(n)
            if randint(0, 4) == 1:
                delay_dict[r]["Proposal"].append(n)
            if randint(0, 4) == 1:
                delay_dict[r]["Timeout"].append(n)
    delay_dict={"delay_round_msg":delay_dict}

    return drop_dict, delay_dict
  }
```

Scenario Executor:

The test executor takes the scenario json and runs it by spawning the appropriate **validators including twins** along with the **client and Network Playground.** We have provisioned **online safety and liveness checks** in our twinBFT implementation. The Network Playground assesses continuously the state of the system and performs safety and liveness checks. The network playground callbacks the Scenario Executor to close the twin whenever a safety/liveness check is violated or when all the client requests are completed . The callback is triggered in Network Playground using the below awai statement.

```
await(self.completed == True)
```

Safety check:

We check for safety violations **whenever a new round is encountered** in our system . The checking is done by checking the commits made by the validators(excluding twins) in their ledge files in twinBFT/Loyal_Byzantine_Generals_phase_2/ledgers/config_* files. We are basically **ensuring that the commits made by each level by the validators are the same.** The changes for the same is made in the pseudocode file twinBFT/phase_3/pseudocode/scenario_executor file under function

```
Function SafetyCheck(){
    //check if there are differing commits
    ledger_dict = {}
    maxlines = 0
    //extract data from ledger
    for n in range(0, number_of_nodes):
        level = 0
        filename = "validator_" + str(n) + ".ledger"
        //Read line by line from the ledger and insert in the ledger_dict for that
node
        ledger_dict[n] = Lines


    for l in range(0, maxlevel):
        blocks = set()
        for n in range(0, self.number_of_nodes):
            if l < len(ledger_dict[n]):
                blocks.add(ledger_dict[n][l])
        //check if there are duplicate commits at the same level
        if len(blocks) > 1:
            //"Safety Check violated"
            return False

    return True
```

Liveness check:

For checking liveness violations, we have to mainly ensure that the system is not stuck in a round. To handle this we assign a TLL (Time to live parameter) for each round such that if more than the TTL limit of messages are passed into the system, it implies that our system is stuck. Currently, we have setted this parameter to 100. Below is the implementation for the same

```
Function LivenessCheck(current_round){
    //check if round of the system has progressed
    if current_round > self.liveness_round:
      self.liveness_round = current_round
      self.current_liveness_TTL = self.liveness_TTL
      return True
    //More than the TTL limit of messages have been passed in the system for a round
    //This implies that the system is stuck and liveness i violated
    if(self.current_liveness_TTL <= 0):
        return False
    //decrease the TLL for the current round
    self.current_liveness_TTL-=1
          return True
  }
```

Another level of liveness check is done by us in the end . Basically when the system exits, we check the ledger files and ensure that atleast one commit is

Network Playground:

Our Network playground simply acts as a tunnel which forwards messages within nodes. And based on data collected from these messages, the Network playground judges the state of the system. And informs the scenario executor accordingly.

Configuration

Configuration template :

We have used a json file to pass our configuration, we use **twin_config.json** as the main file. It contains a list of scenarios that are individually run on the twinBFT.

Below is a sample configuration of a scenario used in twinBFT.

```
{   "number_of_nodes":4,
        "number_of_twins":1,
        "round_leaders":{
            "0":1,"1":1,"2":1,"3":1,"4":1,"5":3,"6":2
        },
        "round_partitions":{
        "0":[[[0,1,2,3],[3]]],
        "1":[[[0,1,2,3],[4]]],
        "2":[[[0,1,2,3],[4]]],
        "3":[[[0,1,2,3],[4]]],
        "4":[[[0,1,2,3],[4]]],
        "5":[[[0,1,2,3],[4]]]
        },
        "drop_round_msg":{
        "1":{"Vote":[1,2],"Proposal":[],"Timeout":[]},
        "2":{"Vote":[],"Proposal":[2],"Timeout":[]},
        "3":{"Vote":[],"Proposal":[],"Timeout":[]},
        "4":{"Vote":[],"Proposal":[],"Timeout":[]},
        "5":{"Vote":[4],"Proposal":[],"Timeout":[]},
        "6":{"Vote":[4],"Proposal":[],"Timeout":[]},
        "7":{"Vote":[4],"Proposal":[],"Timeout":[]},
        "8":{"Vote":[],"Proposal":[],"Timeout":[]},
        "9":{"Vote":[],"Proposal":[],"Timeout":[]},
        "10":{"Vote":[],"Proposal":[],"Timeout":[]}
        },
        "delay_round_msg":{
        "1":{"Vote":[],"Proposal":[1],"Timeout":[]},
        "2":{"Vote":[4, 6],"Proposal":[3],"Timeout":[]},
        "3":{"Vote":[],"Proposal":[],"Timeout":[]},
        "4":{"Vote":[],"Proposal":[],"Timeout":[]},
        "5":{"Vote":[],"Proposal":[],"Timeout":[]},
        "6":{"Vote":[],"Proposal":[],"Timeout":[]},
        "7":{"Vote":[],"Proposal":[],"Timeout":[]},
        "8":{"Vote":[],"Proposal":[],"Timeout":[]},
        "9":{"Vote":[],"Proposal":[],"Timeout":[]},
        "10":{"Vote":[],"Proposal":[],"Timeout":[]}
        }
    }
```

As seen above Information about each round is present in the json config. The leader assignment and the network partition for each round can be easily interpreted from the file.

For message **Drops** and **Delays**, we use separate sections of **drop_round_message** and **delay_round_message** respectively. Here for every round we have information regarding the drop/delay of **type based messages**(votemsg, timeout, proposal) . For example

```
"drop_round_msg":{
        "1":{"Vote":[2],"Proposal":[],"Timeout":[4]},
"delay_round_msg":{
        "2":{"Vote":[4, 6],"Proposal":[3],"Timeout":[]},
```

The above information means that for round 1, the Vote Message from validator 1 and Timeout Message from Validator 4 are dropped. Similarly, for round 2, the Vote Message from validator 4 and 6 is delayed, and Proposal message from validator 3 is delayed as well.

Logs

Our twin generates comprehensive log entries for each scenario which can be found in the **DiemBFT/twinBFT/logs** folder. In the logs folder , logs for each scenario in a separate config folder (**DiemBFT/twinBFT/logs/config_*)**.This way we are segregating the logs for each scenario.

The **NetworkPlayground_*.log** and **scenario_executor_*.log** files are mainly used to collect logs for major events occurring in twinBFT

User Manual

The user manual can be found in **DiemBFT/twinBFT/UserManual.md** file. The Sequence of Instructions is mentioned step by step.

Test Report

Testing Safety Violation :

To test Safety violations, we introduced bugs in the DiemBFT code. Following are the types of bugs we introduced.

Type1 - forming QC in 2f votes instead for 2f+1 :

We changed the diemBFT code in the BlockTree section in the process_vote() function of diemBFT.
The QC should be formed only when at least 2*f +1 votes are received where f is the number of faulty nodes.
This is done at **line 108 of the block_tree.da file** in Loyal_Byzantine_Generals_phase_2
After Introducing this bug we got duplicate commits in the form of different block id hashes on the same levels.
As explained in the SafetyCheck() section because of online checking our twinBFT exists on detecting safety violations.
Below is a config that we use to test Safety. Note That we need to make a change in the configuration of scenario executor to change quorum size . This is done at line 169 in scenario_generator.da

```
'exclude_size': 0,
            ->"quorum_size" : 2*(int(number_of_nodes-1)/3),
            'failure_config': FailureConfig(
                failures=[],
```

(twinconfig_safety.json)

```
    {
        "number_of_nodes": 4,
        "number_of_twins": 1,
        "round_leaders": {
            "0" : 0,
            "1" : 0,
            "2" : 0,
            "3" : 0
        },
        "round_partitions": {
            "0" : [
                [[0,1,2],[3,4]]
            ],
            "1" : [
                [[0,1,2],[3,4]]
            ],
            "2" : [
                [[0,1,2], [3,4]
                ]
            ],
            "3" : [
                [[0,1,2],[3,4]]
            ]
        },
        "drop_round_msg":{
            ……………..
         },
         "delay_round_msg":{
            ………..
         }
    }
```

After running the above config we get differing commits in the ledger at the same level like below.

Validator 0:

0-0 - b'b2bd4f060be6a95d49a368118f019118a5c66cb07a3c3445ea7c1f23398211f0'

0-1 - b'413f0679439c62a98a6aad95ca58f4528a09be301579bffec00921bf1b3287cc'

Validator 1:

0-0 - b'b2bd4f060be6a95d49a368118f019118a5c66cb07a3c3445ea7c1f23398211f0'

0-1 - b'413f0679439c62a98a6aad95ca58f4528a09be301579bffec00921bf1b3287cc'

Validator 2:

0-0 - b'c2710cf1bd88372db66f2bb92fa643c1a69cc51c1c128b58f5912c978c1cbcee'

Validator 3:

0-0 - b'c2710cf1bd88372db66f2bb92fa643c1a69cc51c1c128b58f5912c978c1cbcee'

0-1 - b'4151df3ff14d7bcacaf83587c6f3c6dedfa6630f418aa53798bcc0feeeffbac4'

Validator 4;

0-0 - b'c2710cf1bd88372db66f2bb92fa643c1a69cc51c1c128b58f5912c978c1cbcee'

0-1 - b'4151df3ff14d7bcacaf83587c6f3c6dedfa6630f418aa53798bcc0feeeffbac4'

As we can see above, we get differing commits . It can even be checked in our log file of NetWork Playground that SafetyCheck() is being violated and the system closes down immediately.

```
(diem37) vscode ➜ .../twinBFT/DiemBFT/twinBFT/Loyal_Byzantine_Generals_phase_2 (feature/block_sync2 ✗ ) $ cat logs/config0/NetworkPlayGround_0.log | grep -i "Violation"
2021-12-03 04:39:15.942312 NetworkPlayGround_0 [Violation] Safety Check violated []
2021-12-03 04:39:16.076823 NetworkPlayGround_0 [Violation] Safety Check violated []
```

Testing Liveness Violation :

For liveness violation we generate partitions where all nodes are separated. This way no communication would happen and the network playground timeout would get triggered. This is a very simple test case to test. Basically we are initiating a timer for a round for 30 seconds. If the round of the system is not incremented, then Liveness of the system is violated

Below is a config that we use to test Safety

(twinconfig_liveness.json)

```
{
    "number_of_nodes": 4,
    "number_of_twins": 1,
    "round_leaders": {
        "0" : 0,
        "1" : 0,
        "2" : 0,
        "3" : 0
    },
    "round_partitions": {
        "0" : [
             [
               [0], [1] , [2] , [3], [4]
             ]
        ],
        "1" : [
            [
              [0], [1] , [2] , [3], [4]
            ]
        ],
        "2" : [
            [
              [0], [1] , [2] , [3], [4]
            ]
        ],
        "3" : [
            [
              [0], [1] , [2] , [3], [4]
            ]
        ]
    },
    "drop_round_msg":{
        ………….
     },
     "delay_round_msg":{
        …………..
     }
}
```

This can also be seen in the log file of NetworkPlayground

(diem37) vscode ➜ .../twinBFT/DiemBFT/twinBFT/Loyal_Byzantine_Generals_phase_2 (feature/block_sync2 ✗) $ cat logs/config0/NetworkPlayGround_0.log | grep -i "Violation"

2021-12-03 02:58:57.929561 NetworkPlayGround_0 [Violation] Liveness of the system violated []

BugFixes in Loyal_Byzantine_Generals_phase_2:

Hash Validation:

As mentioned in the phase 4 doc, we have ensured that validation for blocks before it is committed. Similarly hash validation of VoteMsg is taking place before considering the vote.
Below is the implementation

```
def hash_validation_block(block):
        qc = block.qc
        if qc is None:
            id = Cryptography.hash(block.author, block.round,
                                   block.payload)
        else:
            id = Cryptography.hash(block.author, block.round,
                                   block.payload, qc.vote_info.id, qc.signatures)
        if id == block.id :
            return True
        else:
            return False

    def hash_validation_votemsg(vote_msg):
        if Cryptography.hash(vote_msg.vote_info) ==
vote_msg.ledger_commit_info.vote_info_hash:
            return True
        else:
            return False
```

Following are the places where this check is being performed.
In BlockTree.da file

```
    def execute_and_insert(self, block):
        LOGGER.log('BlockTree: execute_and_insert', block)

        if not Cryptography.hash_validation_block(block):
            LOGGER.log('[HashValidation]BlockTree: Block Validation failed')
            return
```

In validator.da, in the receive handler of the votemsg the hash is validated

```

        if not Cryptography.hash_validation_votemsg(vote_msg):
            LOGGER.log_action("VoteMsg hash validation failed", False)
            return

```

SyncUP:

In-order to sync-up lagging nodes and to ensure the liveness of the system we are making use of Proposal Messages and TCs (Timeout certificate). The sync-up is triggered lazily, that is whenever a node encounters a message (vote, timeout or proposal) with a higher round. It triggers the synchronization procedure, by which it requests the sender to aid in syncing up by sending **‘SyncRequest’** with its **‘high_commit_qc’.** As soon as the sender receives the **SyncRequest.** It checks it’s **proposal_cache** for valid proposals / TCs from high_commit_qc round onwards. And sends all replies back to the validator requesting for sync. The proposal cache is updated whenever a TC is formed or whenever a valid proposal is received and processed. Details about sync-up can be found in the code twinBFT/src/validator.da.

We are checking the SyncUP by generating the scenario below .
twinconfig_sync.json

```

    {
        "number_of_nodes": 4,
        "number_of_twins": 1,
        "round_leaders": {
            "0": 1,
            "1": 2,
            "2": 3,
            "3": 1,
            "4": 1,
            "5": 3,
            "6": 2
        },
        "round_partitions": {
            "0": [
                [[0,1,2],[3,4]
                ]
            ],
            "1": [
                [[0,1,2],[3,4]
                ]
            "2": [
                [[4,3,1],[0,2]
                ]
            ],
            "3": [
                [[4,3,1],[0,2]]
            ],
            "4": [
                [[4,3,1],[0,2]
                ]
            ],
            "5": [
                [[4,3,1],[0,2]
                ]


            ],
            "6": [
                [
                    [[0,1,2],[3,4]
                ]
            ]
        },
        "drop_round_msg":{
            ………….
         },
         "delay_round_msg":{
            …………..
         }
    }
```

Explanation:

Here in Round 1 and Round 2 QC is formed in validators 0, 1 and 2. By this time, the Validators 3 and 4 are lagging.Further in Round 3 and Round 4, the sync up is done for Validators 4 and 3 from Validator 1 and logs are committed till Round 5. Like previously, Validators 0 and 2 are now lagging behind which are synced up by validator 1 in round 6. Thus by the end of execution, all the validators are synced up .

The logs from the Sync Requests can be found in the Validator Logs as well like example below

```
sgType: Proposal | To: validator_2 | Msg:
Timestamp:  2021-12-03  03:47:34.691656  |  ProcessId:  validator_0  |  Event:  Received
SyncTC Request at round 3 | MsgType: SyncTC | To: parent_process | Msg:
Timestamp:  2021-12-03  03:47:34.772294  |  ProcessId:  validator_0  |  Event:  Received
SyncTC Request at round 3 | MsgType: SyncTC | To: parent_process | Msg:
2021-12-03 03:48:12.083874 validator_0 SyncRequest trigged on recieving Timeout with
round 7 and current pacemaker round 3  []
Timestamp: 2021-12-03 03:48:12.116367 | ProcessId: validator_0 | Event: round 7 Send
| MsgType: SyncRequest | To: validator_1 | Msg: None
Timestamp:  2021-12-03  03:48:12.679040  |  ProcessId:  validator_0  |  Event:  Received
SyncTC Request at round 3 | MsgType: SyncTC | To: parent_process | Msg:
Timestamp:  2021-12-03  03:48:12.708914  |  ProcessId:  validator_0  |  Event:  Received
SyncTC Request at round 3 | MsgType: SyncTC | To: parent_process | Msg:
```

Tackling Duplicate Commits:

The bug present in the Mempool was that a node which would have missed receiving a proposal would create a new proposal with an older transaction. This resulted in a previously committed transaction being committed twice. To prevent these duplicate commits, we make use of **request_cache (**which holds which transactions are committed**)**. Thus a proposal with previously committed transactions would not be voted upon. So if a new request is already in this request_cache (basically duplicate request being considered), then validate_and remove_transactions() will return False.

```
def validate_and_remove_transactions(self, txns):
        LOGGER.log('MemPool: validate_and_remove_transactions', txns)
        to_remove = []
        for txn in txns:
            if txn['req_id'] in self.pending_txns and txn['req_id'] not in
self.request_cache:
                to_remove.append(txn['req_id'])
            else:
                LOGGER.log(
                    'MemPool: validate_and_remove_transactions: returns', False,
txn['req_id'] in self.pending_txns, txn['req_id'] not in self.request_cache)
                return False

        for req_id in to_remove:
            LOGGER.log('MemPool: Removed transaction ', req_id)
            del self.pending_txns[req_id]
        LOGGER.log('MemPool: validate_and_remove_transactions: returns', True)
        return True
```

BlockTree process_vote:

When processing votes, pending votes stored a list of vote messages. Incase a vote message was sent twice pending_votes would append the vote to pending votes list and a QC would be formed with just 2*f votes. To fix this we have added a check such that only if a vote signature is unique it would be added to the pending_votes.

```
    def process_vote(self, vote_msg):
        LOGGER.log('BlockTree: process_vote', vote_msg,self.validator_id)
        '''process qc(v.high commit qc)'''
        self.process_qc(vote_msg.high_commit_qc)
        '''vote idx ← hash(v.ledger commit info)'''
        vote_idx = Cryptography.hash(vote_msg.ledger_commit_info)

        '''pending votes[vote idx] ← pending votes[vote idx] ∪ v.signature'''
                if vote_idx in self.pending_votes and vote_msg.signature not in
self.pending_votes[vote_idx]:
            self.pending_votes[vote_idx].append(vote_msg.signature)
            self.pending_signers[vote_idx].append(vote_msg.sender)
        else:
            self.pending_votes[vote_idx] = [vote_msg.signature]
            self.pending_signers[vote_idx] = [vote_msg.sender]
```

twinconfig_leader.json

```
    {
        "number_of_nodes": 4,
        "number_of_twins": 1,
        "round_leaders": {
            "0" : 0,
            "1" : 0,
            "2" : 0,
            "3" : 0,
            "4" : 0,
            "5" : 0,
            "6" : 0
        },
        "round_partitions": {
            "0" : [
                [[0,1,2],[3,4]
                ]
            ],
            "1" : [
                [[0,1,2],[ 3,4]
                ]
            ],
            "2" : [
                [[0,1,2],[3,4]
                ]
            ],
            "3" : [
                [[0, 1,2], [3,4]
                ]
            ],


            "4" :  [
                [[4,3,1],[0,2]
                ]
            ],
            "5" : [
                [[0,1,2],[3,4]
                ]
            ]
        },
        "drop_round_msg":{
            ………….
         },
         "delay_round_msg":{
            …………..
         }
}
```

The commit made in the ledgers are as below
0-0 - b'8172913a521738bcc915e1bc1ee396bd676c6085f4fa8457eecfbc315293f74d'
0-1 - b'a0b31be403200ac539f65310057868bb3c84f466f90520d1ef08379607ac8a11'

twinconfig_3.json

```
{
       "number_of_nodes": 4,
       "number_of_twins": 1,
       "round_leaders": {
           "0": 0,
           "1": 2,
           "2": 2,
           "3": 2,
           "4": 0,
           "5": 0,
           "6": 0,
           "7": 3
       },
       "round_partitions": {
           "0": [
               [[3,1,2,4,0]
               ]
           ],
           "1": [
               [[0,1,2],[3,4]
               ]
           ],
           "2": [
               [[4,3,1],[0,2]
               ]
           ],
           "3": [
               [[3,1,2,4,0]
               ]
```

```
            ],
            "4": [
                [[4,3,1],[0,2]
                ]
            ],
            "5": [
                [[4,3,1],[ 0,2]
                ]
            ],
            "6": [
                [[3,1,2,4,0]
                ]
            ],
            "7": [
                [[0, 1,2],[3, 4]
                ]
            ]
        },
```

Explanation :
In this configuration, We have assigned mainly Validators 0 and 2 as leaders for the majority of the rounds. In this scenario too Sync up is working . For example the Validator 2 is lagging behind in round 6 and by the end of the execution, all the Validators are in Sync. Detailed logs for the same can be found in the log files

```
Timestamp: 2021-12-03 04:09:34.514596 | ProcessId: validator_2 | Event: Received SyncTC Request at round 5 | MsgType: SyncTC | To: parent_process | Msg:
Timestamp: 2021-12-03 04:09:34.534148 | ProcessId: validator_2 | Event: Received SyncTC Request at round 5 | MsgType: SyncTC | To: parent_process | Msg:
2021-12-03 04:09:46.384598 validator_2 Higher timeout round encountered syncing the node timeout msg round 7 current pacemaker round 8  []
2021-12-03 04:09:46.389945 validator_2 Received High Commit QC with round 6 when processing sync request []
2021-12-03 04:09:46.395932 validator_2 Sync Request proposal cache {1: (<object_types.ProposalMsg object at 0x7fd1d0502410>, 4, 1), 2: (<object_types.ProposalMsg object at 0x7fd1d050db90>, 2, 2), 3: (<object_types.ProposalMsg object at 0x7fd1d0578b90>, 2, 3), 4: <object_types.TC object at 0x7fd1d050d910>, 5: (<object_types.ProposalMsg object at 0x7fd1d0522450>, 4, 5), 6: (<object_types.ProposalMsg object at 0x7fd1d052b290>, 4, 6), 7: <object_types.TC object at 0x7fd1d0557150>, 8: (<object_types.ProposalMsg object at 0x7fd1d04aab10>, 3, 8)} []
```

Below are the commits made in the ledger

```
0-0 - b'3d827e23131790f1366e509aabf9e09b53fa831dccd78ed3d297874f749e76fb'
0-1 - b'aa24f7af8a836d58276447d4b1fff9f942561136cbc51a8c312b1aa7f8cbf7e1'
0-2 - b'91d6ed713ba4a6c02b73674df76a3e8867269d4acca95b752be634fdcbc657b8'
0-3 - b'0fda3323a55a274ce865330b52fc38768eae3ab941a0719b34321471b9d11168'
```

ReadMe :

Checkout the twinBFT/twinREADME.md file