



DLT Implementation & Internals

Assignments:

Part 1: Problem set for Custom DLT Implementation

Part 2: Problem set for Substrate

14.07.2021

Waqas Ahmed

Student # 1911706

Course: DLT5400: DLT Implementation and Internals



L-Università
ta' Malta

Part 2:

1. Smart Contract Upload and Retrieval

Example Smart Contract (Bytes):

```
0x3063736430323334626e3334626e3362346e62336e3462336e
```

Upload Contract

Pallet Interactor

Interaction Type ☒ Extrinsic ☐ Query ☐ RPC ☐ Constant

templateModule

uploadContract

sc Bytes

Unsigned or Signed or SUDO

Events

system:ExtrinsicSuccess:: (phase=["applyExtrinsic":0])-0
DispatchInfo: {"weight":161650000,"class":"Mandatory"," PaysFee":"Yes"}
An extrinsic completed successfully. \[info]

Pallet Interactor

Interaction Type ☒ Extrinsic ☐ Query ☐ RPC ☐ Constant

templateModule

uploadContract

sc 0x3063736430323334626e3334626e3362346e62336e3462336e

Unsigned or Signed or SUDO

Finalized. Block hash:
0x79bff2f4f0028523357fb5fec11de167a2bf458116419d4852cc20098deb252a

Events

system:ExtrinsicSuccess:: (phase=["applyExtrinsic":0])-4
DispatchInfo: {"weight":161650000,"class":"Mandatory"," PaysFee":"Yes"}
An extrinsic completed successfully. \[info]

system:ExtrinsicSuccess:: (phase=["applyExtrinsic":1])-3
DispatchInfo: {"weight":1000,"class":"Normal"," PaysFee":"Yes"}
An extrinsic completed successfully. \[info]

templateModule:SCUploaded:: (phase=["applyExtrinsic":1])-2
AccountId: 5GrwvaEF5zXb26Fz9rcQpDWS57CtERHpNehXCPcNoHGKutQY,
Option<u32>: 75
Event emitted when a smart contract is uploaded. [origin, sc]

Retrieve Contract

Pallet Interactor

Interaction Type ☒ Extrinsic ☐ Query ☐ RPC ☐ Constant

templateModule

retrieveContract

sc_address Option<u32>

Leaving this field as blank will submit a NONE value

Unsigned or Signed or SUDO

Events

system:ExtrinsicSuccess:: (phase=["applyExtrinsic":0])-0
DispatchInfo: {"weight":161650000,"class":"Mandatory"," PaysFee":"Yes"}
An extrinsic completed successfully. \[info]



L-Università
ta' Malta

Pallet Interactor

Interaction Type ☒ Extrinsic ☐ Query ☐ RPC ☐ Constant

templateModule

retrieveContract

sc_address 75

Leaving this field as blank will submit a NONE value

Unsigned or Signed or SUDO

Events

- system:ExtrinsicSuccess:: (phase=["applyExtrinsic":0])-8**
DispatchInfo: {"weight":161650000,"class":"Mandatory","paysFee":"Yes"}
An extrinsic completed successfully. \[info\]
- system:ExtrinsicSuccess:: (phase=["applyExtrinsic":1])-7**
DispatchInfo: {"weight":10000,"class":"Normal","paysFee":"Yes"}
An extrinsic completed successfully. \[info\]
- templateModule:SCRetrieved:: (phase=["applyExtrinsic":1])-6**
AccountId: 5GrwvaEF5zXb26Fz9rcQpDWS57CtERHpNehXCPcNoHGKutQY,
Bytes: 0x3063736430323334626e3334626e3362346e62336e3462336e
Event emitted when a smart contract is retrieved. [origin, sc]

Code

pallets/template/src/lib.rs

```
#![cfg_attr(not(feature = "std"), no_std)]

// Re-export pallet items so that they can be accessed from the crate namespace.
pub use pallet::*;

#[frame_support::pallet]
pub mod pallet {
    use frame_support::{dispatch::DispatchResultWithPostInfo, pallet_prelude::*};
    use frame_system::pallet_prelude::*;
    use sp_std::vec::Vec;
    use sp_std::convert::TryInto;

    /// Configure the pallet by specifying the parameters and types on which it depends.
    #[pallet::config]
    pub trait Config: frame_system::Config {
        /// Because this pallet emits events, it depends on the runtime's definition of an
        event.
        type Event: From<Event<Self>> + IsType<<Self as frame_system::Config>::Event>;
    }

    // Pallets use events to inform users when important changes are made.
    // Event documentation should end with an array that provides descriptive names for
    parameters.
    // https://substrate.dev/docs/en/knowledgebase/runtime/events
    #[pallet::event]
    // #[pallet::metadata(Option<u32> = "Metadata")]
}
```



L-Università
ta' Malta

```

#[pallet::metadata(T::AccountId = "AccountId")]
#[pallet::generate_deposit(pub(super) fn deposit_event)]
pub enum Event<T: Config> {
    /// Event emitted when a smart contract is uploaded. [origin, sc]
    SCUploaded(T::AccountId, Option<u32>),
    /// Event emitted when a smart contract is retrieved. [origin, sc]
    SCRetrieved(T::AccountId, Vec<u8>),
}

#[pallet::error]
pub enum Error<T> {
    /// The smart contract has already been uploaded.
    SCAlreadyUploaded,
    /// The smart contract does not exist, so it cannot be retrieved.
    NoSuchSC,
}

#[pallet::pallet]
#[pallet::generate_store(pub(super) trait Store)]
pub struct Pallet<T>(_);

#[pallet::storage]
pub(super) type SC<T: Config> = StorageMap<_, Blake2_128Concat, Option<u32>, Vec<u8>, ValueQuery>;

#[pallet::hooks]
impl<T: Config> Hooks<BlockNumberFor<T>> for Pallet<T> {}

// Dispatchable functions allows users to interact with the pallet and invoke state changes.
// These functions materialize as "extrinsics", which are often compared to transactions.
// Dispatchable functions must be annotated with a weight and must return a DispatchResult.
#[pallet::call]
impl<T: Config> Pallet<T> {
    #[pallet::weight(1_000)]
    pub(super) fn upload_contract(
        origin: OriginFor<T>,
        sc: Vec<u8>,
    )

```



```

    ) -> DispatchResultWithPostInfo {

        // Check that the extrinsic was signed and get the signer.
        // This function will return an error if the extrinsic is not signed.
        // https://substrate.dev/docs/en/knowledgebase/runtime/origin
        let sender = ensure_signed(origin)?;

        // Get the block number from the FRAME System module.
        let current_block_uncasted =
<frame_system::Module<T>::block_number();//::from(10u32);
        let current_block = TryInto::<u32>::try_into(current_block_uncasted).ok();

        // Verify that the specified smart contract has not already been uploaded.
        ensure!(!SC::<T>::contains_key(current_block), Error::<T>::SCAlreadyUploaded);

        // Store the smart contract with the sender and block number.
        SC::<T>::insert(current_block, &sc);
        // Emit an event that the smart contract was uploaded.
        Self::deposit_event(Event::SCUploaded(sender, current_block));
        Ok(()).into()
    }

#[pallet::weight(10_000)]
fn retrieve_contract(
    origin: OriginFor<T>,
    sc_address: Option<u32>,
) -> DispatchResultWithPostInfo {
    // Check that the extrinsic was signed and get the signer.
    // This function will return an error if the extrinsic is not signed.
    // https://substrate.dev/docs/en/knowledgebase/runtime/origin
    let sender = ensure_signed(origin)?;

    // Verify that the specified smart contract has been uploaded.
    // ensure!(SC::<T>::contains_key(&sc), Error::<T>::NoSuchSC);
    let sc = SC::<T>::get(&sc_address);
    // Emit an event that the smart contract was retrieved.
    Self::deposit_event(Event::SCRetrieved(sender, sc));
}

```



```

        Ok(()).into())
    }
}
}

```

2. Limited JVM-like Interpreter

Attempted to write an `execute_contract` method based on newly designed bytecode operations given below and utilizing built-in instructions but didn't succeed (due to very limited rust knowledge and experience).

3. Report

(i) a description of the bytecode operations you implemented;

Mnemonics	Opcode (in hex)	Description
<code>iconst_2</code>	05	load the int value 2 onto the stack
<code>irem</code>	70	logical int remainder
<code>ifeq</code>	99	if value is 0, branch to instruction at <code>branchoffset</code> (signed short constructed from unsigned bytes <code>branchbyte1 << 8 branchbyte2</code>)
<code>goto</code>	a7	goes to another instruction at <code>branchoffset</code> (signed short constructed from unsigned bytes <code>branchbyte1 << 8 branchbyte2</code>)
<code>return</code>	b1	return void from method
<code>blockno</code>	cb	push the constant u32 onto the stack
<code>emit</code>	cc	push the constant string onto the output



- JVM has left the opcodes (cb-df) unassigned for future use, so we have used them as per our need.

(ii) listing any functionality that you did not implement.

Theoretically, the specifications have been drafted as per the given instructions, however, didn't implement the JVM as part of lib.rs



Part 1:

Git repo: https://github.com/wakqasahmed/dlt_implementation/tree/main/assignment

1. SBB Completion

```

X Default (node) X Default (node) X Default (node) X Default (node)
ck time: 1427.8465711369233ms
PoW
block# 18 found: 000075ebabf53df1aadf1d2d0
22c06cfb0afe5a58d3791bbf9096bc85c505a9 | ti
me taken: 2297.4024119377136ms | average bl
ock time: 1476.1552289591896ms
PoW
block# 19 found: 000085861d33db7c5bcbba7ed0
3d7d107901fa71337d402c7a5b309b40a3ea61 | ti
me taken: 434.98714303970337ms | average bl
ock time: 1421.3569086476375ms
PoW
block# 20 found: 00007c694b859d77192ae2e55c
dd0937d561a4e254341f128e8e34fd7acf21be | ti
me taken: 4997.443950176239ms | average blo
ck time: 1600.1612607240677ms
PoW
block# 21 found: 0000a2d33566b167107baa58bd
034123c06b9779fd2607590653f1732c81769 | ti
me taken: 1316.6939420700073ms | average bl
ock time: 1586.6628169786363ms
PoW
block# 22 found: 0000a023f917f33bf6bd6e3feb
9fdeb11f4d0ebce43d8e8c7c3b894a38d04f2c | ti
me taken: 1968.5644371509552ms | average bl
ock time: 1604.0219815319235ms
PoW
block# 23 found: 0000542f60162825d656670823
f759180f7650dcbe97c2b0f0388f12d74b21cf | ti
me taken: 8761.065191984177ms | average blo
ck time: 1915.1977732907171ms
PoW
block# 24 found: 000005ee0468ba27fb5ca911cb
026e2643f870d3567b3b53d41d1d2a497bf693 | ti
me taken: 8918.744784116745ms | average blo
ck time: 2207.0122320751348ms
PoW
block# 25 found: 0000ec589e458397bf57af4458
81e6c6c203e2e7fd6d596239fe01e9d613cdf2 | ti
me taken: 9261.522982120514ms | average blo
ck time: 2489.19266207695ms
PoW
> 1
prompt>Model:
1) Account
2) UTXO
> 1
prompt>Encoding:
1) JSON
2) Byte
> 1
Configured successfully
syncing in progress...
prompt>Command:
1) b OR balance
2) l OR ledger
3) s <publicKey> OR send <publicKey> e.g. s
end 04440a759412d...
> PoW
block# 1 found: 0000b3547042a5f9da539c44843
506d884f5acf971b9bd9a4443657b1f2bd971 | tim
e taken: 3513.9965319633484ms | average blo
ck time: 3513.9965319633484ms
PoW
block# 2 found: 00007edd436723163a7e473440
4c86076429ac666d67bc651c6e686fcb5a03 | tim
e taken: 645.0667729377747ms | average bloc
k time: 2079.5316524505615ms
PoW
block# 3 found: 0000af49de42c87e5d784a35e08
214aa1404267ed17d4a14c41720c4299e4910 | tim
e taken: 19763.789266824722ms | average blo
ck time: 7974.284190575282ms
PoW
block# 4 found: 00003fd7ed24a7a6a4116e4e5d7
9fd4c4fefc03f6834a39ebe12176dc5afb019 | tim
e taken: 5683.89185500145ms | average block
time: 7401.686106681824ms
PoW
block# 5 found: 00001d0def5c700cd6536d2f5a7
2c61e720912c84c9977995123c9359ba2458 | tim
e taken: 6240.776314973831ms | average bloc
k time: 7169.504148340226ms
PoW
node.
prompt>Consensus Algorithm:
1) PoW (Proof of Work)
2) PoT (Proof of Turn)
> 1
prompt>Model:
1) Account
2) UTXO
> 1
prompt>Encoding:
1) JSON
2) Byte
> 1
Configured successfully
syncing in progress...
prompt>Command:
1) b OR balance
2) l OR ledger
3) s <publicKey> OR send <publicKey> e.g. s
end 04440a759412d...
> PoW
block# 1 found: 00001c267c39301037d69f5defc
c40e9866d6fb8cb3c180968ade46912f960a1 | tim
e taken: 3983.25096988678ms | average block
time: 3983.25096988678ms
PoW
block# 2 found: 00001c6ff22dca16093e217fd93
ad167c6e0b2428ec92585bc090ed82b9d8801 | tim
e taken: 3173.642467021942ms | average bloc
k time: 3578.446718454361ms
PoW
block# 3 found: 0000c843cb5ae3218bc090e2b
9deb2cc5419db8cf4b4a7f681e294244d5fb | tim
e taken: 4620.298658847809ms | average bloc
k time: 3725.7306985855103ms
PoW
block# 4 found: 00004c2ead79d57865b682d6a0d
398f6d02dc43e40c5de5b101622d5a2eb3f51 | tim
e taken: 6381.828951835632ms | average bloc
k time: 4389.755261898041ms
PoW
waqas.ahmed@192 assignment % node -r esm in
dex.mjs 8004
{
  activeNodes: 0,
  currentNode: { ip: null, port: null },
  nodePublicKeys: [],
  initPeers: [Function: initPeers],
  broadcastMessage: [Function: broadcastMes
sage]
}
Node Public Key: 0422a1bbff3c69811e695d36c
773a245bf51dd67ea1ea5dddb24b08597b93f066e9
21c3975a82296198dc05fee8ef4078e571277ba2dcb
5cfbf9d47b077a78071
{ address: '0.0.0.0', family: 'IPv4', port:
8004 }
Current Port: 8004
server listening 0.0.0.0:8004
Please choose the options to configure your
node.
prompt>Consensus Algorithm:
1) PoW (Proof of Work)
2) PoT (Proof of Turn)
> 1
prompt>Model:
1) Account
2) UTXO
> 1
prompt>Encoding:
1) JSON
2) Byte
> 1
Configured successfully
syncing in progress...
prompt>Command:
1) b OR balance
2) l OR ledger
3) s <publicKey> OR send <publicKey> e.g. s
end 04440a759412d...
>

```



2. Configurable Blockchain

```
>< Default (node)
waqas.ahmed@192 assignment % node -r esm index.mjs 8001
{
  activeNodes: 0,
  currentNode: { ip: null, port: null },
  nodePublicKeys: [],
  initPeers: [Function: initPeers],
  broadcastMessage: [Function: broadcastMessage]
}
Node Public Key: 0469aaf9217806c617432274fad1ddcf79ce4512e06cb23120ebe144bc73605c77deabb084d5aa0238cd15eb94c66d886550b3b22252c83001a62f0580f0470a0
{ address: '0.0.0.0', family: 'IPv4', port: 8001 }
Current Port: 8001
server listening 0.0.0.0:8001
Please choose the options to configure your node.

prompt>Consensus Algorithm:
1) PoW (Proof of Work)
2) PoT (Proof of Turn)
> 2
prompt>Model:
1) Account
2) UTXO
> 1
prompt>Encoding:
1) JSON
2) Byte
> 1
Configured successfully
syncing in progress...
prompt>Command:
1) b OR balance
2) l OR ledger
3) s <publicKey> OR send <publicKey> e.g. send 04440a759412d...
>
```



3. Blocktime Calculation

Bitcoin Difficulty Calculation Formula [\[1\]](#)

$$\text{new_difficulty} = \text{old_difficulty} \times (2016 \text{ blocks} \times 10 \text{ minutes}) / (\text{the time took in minutes to mine the last 2016 blocks})$$

SBB Difficulty Calculation Formula

$$\text{new_difficulty} = \text{old_difficulty} \times (100 \text{ blocks} \times 10000 \text{ ms}) / (\text{the time took in ms to mine the last 100 blocks})$$

Genesis difficulty = 1

100 blocks = 110ms

Average Block Time = 1.10 milliseconds

$$\text{new_difficulty} = 1 * (100 * 10000) / (110 * 100) = 91$$

Genesis difficulty = 2

100 blocks = 692ms

Average Block Time = 6.92 milliseconds

$$\text{new_difficulty} = 2 * (100 * 10000) / (692 * 100) = 28.90$$

Genesis difficulty = 3

100 blocks = 8224ms

Average Block Time = 82.24 milliseconds

$$\text{new_difficulty} = 3 * (100 * 10000) / (8224 * 100) = 3.65$$

Genesis difficulty = 4

100 blocks = 122300ms

Average Block Time = 1223 milliseconds



```
new_difficulty = 4 * (100 * 10000) / (122300 * 100) = 0.33
```

```
Genesis difficulty = 5
```

```
100 blocks = 2262000ms
```

```
Average Block Time = 22620 milliseconds
```

```
new_difficulty = 5 * (100 * 10000 ) / (2262000 * 100) = 0.02
```

- Difficulty: `_4 zeros_`
- Average Block Time: `_1223 milliseconds_`
- Test # of Blocks: `_100_`
- Maximum Nonce Limit: `_50000000_` (to prevent infinite loop)
- Method: `_stored each block mine time in an array and computed average_`

Please Note: ``performance.now()`` is used instead of typical ``new Date().getTime()`` for comparison, as suited better to measure performance according to the following article: <https://developers.google.com/web/updates/2012/08/When-milliseconds-are-not-enough-performance-now>

```
$ node -r esm index.mjs 8001
```

```
Node Public Key:
```

```
04995cca93aa091a81409ce4d7e21c296959ce22edba86642e4125db851f7cf27908cf0f2cdcc
524ce32d1f4c38b1651303c1c6f5fec495683b699d1acdd0e11a8
```

```
wallet.publicKey:
```

```
04995cca93aa091a81409ce4d7e21c296959ce22edba86642e4125db851f7cf27908cf0f2cdcc
524ce32d1f4c38b1651303c1c6f5fec495683b699d1acdd0e11a8
```

```
{ address: '0.0.0.0', family: 'IPv4', port: 8001 }
```

```
server listening 0.0.0.0:8001
```

```
syncing in progress...
```

```
prompt>command> You are the first SBB miner, history is being written... The
SBB genesis block
```



block# 1 found:

0000734a58a322096864ccb77008cd2f815ec630c2d56f1214640201ad0052db | time
taken: 417.7852739095688ms | average block time: 417.7852739095688ms

block# 2 found:

000076e122cd3d14cd2606140d6acd824680082c3e84c0c2e4ec4521cd02fc6a | time
taken: 3238.020353913307ms | average block time: 1827.902813911438ms

block# 3 found:

0000f9bd4dab0e344f5ad01cc07a6e0c91b32c5a114c08caa21e7b1116d89223 | time
taken: 326.27406990528107ms | average block time: 1327.359899242719ms

block# 4 found:

0000e14a9c3016f7523f0e67f7d24b77836b2222e45b8a19fab7b328c3e1226a | time
taken: 401.649307012558ms | average block time: 1095.9322511851788ms

block# 5 found:

0000dd83915aa170f0ddebc7b2a8a310437534bcd85442b0a91a4097426365d3 | time
taken: 6.875347971916199ms | average block time: 878.1208705425263ms

.
. .
.

block# 96 found:

0000ea79fba479a4567ed9efc21090fa075ced107d4cd14c0e370b8c617e35a0 | time
taken: 839.1855989694595ms | average block time: 1236.1563069125016ms

block# 97 found:

00000e42f3506de9c3345be1e697801901e1214272953aa0adde2566318540e0 | time
taken: 1649.7632240056992ms | average block time: 1240.420295748514ms

block# 98 found:

0000e95e93e25137a2451a41af2d814ffe3afa5c3165cdb451232cd4d8e20b32 | time
taken: 367.73071002960205ms | average block time: 1231.515299975872ms

block# 99 found:

00006d0effff4708a6059c7f41e1cd620efd1c8ef4340927835211bcdfa1ea917 | time
taken: 508.8213880062103ms | average block time: 1224.215361471128ms

block# 100 found:

0000582530f84bcaf32bf56f2a26a89145b8c243c5dc43117210fb11b431bf1a | time
taken: 1118.7283039093018ms | average block time: 1223.1604908955096ms



- Difficulty: `_5 zeros_`
- Average Block Time: `_22620 milliseconds_`
- Test # of Blocks: `_100_`
- Maximum Nonce Limit: `_50000000_` (to prevent infinite loop)
- Method: `_stored each block mine time in an array and computed average_`

```
$ node -r esm index.mjs 8001
```

```
Node Public Key:
```

```
049ca6a4defc2708dc3b1cd17412d517d1e3405e3aacc50f1249dc4f8823bd30210d4368720de
cfc32f6c46d658c685a4593ca5f2e7c033e65590c3cb89c4a943b
```

```
wallet.publicKey:
```

```
049ca6a4defc2708dc3b1cd17412d517d1e3405e3aacc50f1249dc4f8823bd30210d4368720de
cfc32f6c46d658c685a4593ca5f2e7c033e65590c3cb89c4a943b
```

```
{ address: '0.0.0.0', family: 'IPv4', port: 8001 }
```

```
server listening 0.0.0.0:8001
```

```
syncing in progress...
```

```
prompt>command> You are the first SBB miner, history is being written... The
SBB genesis block
```

```
block# 1 found:
```

```
000003f522d093c5d12c197f18190408610f772baba5b612c8b4572b6ae27355 | time
taken: 5974.994300961494ms | average block time: 5974.994300961494ms
```

```
block# 2 found:
```

```
00000a5ce377fe0561efee6d7972a4d01e23fb22fcc447fe60e7de1d2ad5940e | time
taken: 3685.7001559734344ms | average block time: 4830.347228467464ms
```

```
block# 3 found:
```

```
00000358f45998edb65d32f4cfcba2a65e8fb0a01ae1f39c0dffcd845f2ddec | time
taken: 12747.79771900177ms | average block time: 7469.4973919789ms
```

```
block# 4 found:
```

```
000007306ec85f52e70a2cc7a7ab469f33c45f3d041ce2a9e77d9c1e8ff59727 | time
taken: 1822.564227938652ms | average block time: 6057.764100968838ms
```

```
block# 5 found:
```

```
00000ae9463bc75044f52b2324ce9ef4dda3c191eec247a5490cecf0b06adb73 | time
taken: 27306.965486049652ms | average block time: 10307.604377985ms
```

```
.
```



.

.

block# 96 found:

000002e3fc320e072a10a0cddb291136876d7cb3a7605bca1e0cf211a074b13f | time
taken: 17598.125045895576ms | average block time: 22803.48438367496ms

block# 97 found:

0000005f652eecd55a94c7a3a472dba3040416d8f16e61cd7651c03210c6bdfd | time
taken: 11692.683795928955ms | average block time: 22688.940047718814ms

block# 98 found:

000004e1c281397bf5cc97327f56cc27002f8ee10e5d093e4d6ff860d5f0efd3 | time
taken: 14165.0240598917ms | average block time: 22601.96131314915ms

block# 99 found:

000001c2955aeef1cffd4fbd75255dcf29738d32459729bb65fd331f80b5ef49 | time
taken: 15955.567401051521ms | average block time: 22534.82602110776ms

block# 100 found:

0000061c267eac786b8955cdcd0d6c9e8af266c110a7ff9984572598b347974e | time
taken: 31129.543534994125ms | average block time: 22620.773196246624ms

Thoughts on Blocktime Calculation

The result obtained by mining for 100 blocks with a difficulty of **5 zeros** (whose average is about **22620 ms per block**) is significantly higher than the targeted block time of 10 seconds / block.

In contrast, the difficulty level of 4 zeros averaged about **1223 ms per block** is closer to targeted block time of 10 seconds / block.



4. Proof-of-Turn

The algorithm has been implemented as per the pseudocode given. For the sake of simplicity, a simple version of Peer discovery has been implemented.

Initial Peer Discovery and getting their public keys

0.0) GET_PEERS message - return publicKey of this node and all other peers its connected to

GET_PEERS - request publicKey of peers from neighbour

- Respond with PEERS message
 - Contains public keys of peers

Request:

- <OPID>: 'p'
- <OBJ>: null/None/empty

Response:

- <OPID>: 'q'
- <OBJ>: {'peers': []}

0.1) Upon receiving PEERS message - add peers publicKeys to an array (where missing)

PEERS - contains the public keys of the neighbouring peers

- <OPID>: 'q'
- <OBJ>: {'peers': []}

The way PoT works is, every node finds the **min** of **lsb64** and checks if the node itself is eligible to mine or not. If not, it just waits for the **<block_time>** before starting to compute eligibility. By that time, the eligible node produces the block and propagates to the peers.



5. Account Model DB

Redis is used as the database for the given task which is quite similar but more popular than LevelDB which bitcoin core uses [\[2\]](#)[\[3\]](#).

The screenshot shows three terminal windows. The left window is a Bitcoin node console where a user assigns a node to a specific IP and port (8002) and then interacts with the node's command line. The middle window shows the node's configuration and status, including the public key and the fact that it is listening on port 8002. The right window is a Redis CLI where the user sets a key-value pair for the node's public key and then retrieves it, confirming the successful storage and retrieval of the data.

```

X Default (node)
1) Account
2) UTXO
> 1
prompt>Encoding:
1) JSON
2) Byte
> 1
Configured successfully
syncing in progress...
prompt>Command:
1) b OR balance
2) l OR ledger
3) s <publicKey> OR send <publicKey> e.g. send 04440a75941
2d...
> You are the first SBB miner, history is being written...
The SBB genesis block
b
You (04f1357b21645697d0c95b46bd2564c85454e7a55d79e206cd829
9a9b4b6147043d92c7e880b49d988dddbfa5a8465f7a694f8ffee2161e
6a40eb2f815f1993e78) have 2 SBB tokens.
prompt>Command:
1) b OR balance
2) l OR ledger
3) s <publicKey> OR send <publicKey> e.g. send 04440a75941
2d...
> b
You (04f1357b21645697d0c95b46bd2564c85454e7a55d79e206cd829
9a9b4b6147043d92c7e880b49d988dddbfa5a8465f7a694f8ffee2161e
6a40eb2f815f1993e78) have 5 SBB tokens.
prompt>Command:
1) b OR balance
2) l OR ledger
3) s <publicKey> OR send <publicKey> e.g. send 04440a75941
2d...
> s 04fbd1ee41d3cda9b9a29dfe9beba00303bd0749d4c78870e1367e
6610b5ecc63f4faec850413a39850e423a7ffdeb8e9fffe09d3640a036
d10559695b90b338ab
prompt>Command:
1) b OR balance
2) l OR ledger
3) s <publicKey> OR send <publicKey> e.g. send 04440a75941
2d...
>

X Default (node)
waqas.ahmed@192 assignment % node -r esm index.mjs 8002
{
  activeNodes: 0,
  currentNode: { ip: null, port: null },
  nodePublicKeys: [],
  initPeers: [Function: initPeers],
  broadcastMessage: [Function: broadcastMessage]
}
Node Public Key: 04fbd1ee41d3cda9b9a29dfe9beba00303bd0749d
4c78870e136766610b5ecc63f4faec850413a39850e423a7ffdeb8e9fff
e09d3640a036d10559695b90b338ab
{ address: '0.0.0.0', family: 'IPv4', port: 8002 }
Current Port: 8002
Server listening 0.0.0.0:8002
Please choose the options to configure your node.

prompt>Consensus Algorithm:
1) PoW (Proof of Work)
2) PoT (Proof of Turn)
> 1

X Default (redis-cli)
127.0.0.1:6379> GET 04f1357b21645697d0c95b46bd2564c85454e7a
55d79e206cd8299a9b4b6147043d92c7e880b49d988dddbfa5a8465f7a6
94f8ffee2161e6a40eb2f815f1993e78
"5"
127.0.0.1:6379> GET 04fbd1ee41d3cda9b9a29dfe9beba00303bd074
9d4c78870e136766610b5ecc63f4faec850413a39850e423a7ffdeb8e9f
ffe09d3640a036d10559695b90b338ab
"1"
127.0.0.1:6379>

```

6. UTXO Model

UTXO model is not implemented.

7. Design a Well-Defined Byte Encoding

Message encoding is implemented but the byte-encoding of payload is not implemented.

8. Evaluation

I have only implemented Byte-encoding partially, while skipping UTXO Model (as it require quite a bit of time), theoretically, PoT results in significant reduction of the CPU usage as only the eligible node is going to do the mining process, however, in the current



unoptimized implementation, eligibility has to be determined by every node, thus increasing an overhead a bit on each node.

Moreover, Byte-encoding is compressed in nature and more machine friendly, which should result in low memory usage, faster transfer and efficient I/O.

9. Report:

Provide documentation for:

(i) the average block time and target consecutive "0"s calculation including resultant average times and final parameters used.

Details have been mentioned above, in summary,

- Difficulty: 4 (target consecutive zeros)
- Average block time (closest): 1223 ms

(ii) The protocols used.

- Well-defined message encoding

(iii) Any flaws that you see with aspects of the systems.

- Inefficient peer discovery
- Not scalable (due to imperfect implementation)
- PoT and PoW can't run simultaneously, PoW nodes might take over due to skipping eligibility determination.

(iv) Analysis and thoughts regarding the evaluation metrics extracted.

- Mentioned above under the separate heading.

(v) Any parts of the requirements that you did not implement.

- UTXO Model
- Payload byte-encoding
- Block authenticity validation (though tested with hard-coded message in index.js)

(vi) A description of what a merkle tree is, and how it could be used in the implementations to provide better efficiency.

- Merkle trees facilitate the verification of content securely and efficiently. Technically, it is defined as a tree data structure where leaves are tagged with its own block content hash while the branches/non-leaf are tagged with the hash of its child nodes. Merkle tree ensures that data blocks received from other peers in a P2P network are received undamaged and unaltered, in other words, tamper-proof and



immutable. It also ensures that the other peers are not lying by sending fake blocks.[\[4\]](#)

- Current implementation is similar to linear linked list which is prone to modification, so by definition, it's not immutable. Yes, every block has the hash of the previous block which makes it difficult for the miner to attack once the block is deep in the list, yet, it is inefficient as compared to merkle tree.

