IT465 Assignment 4 - Report

Sachin Prasanna - 211IT058

October 24, 2024

1 Problem Statement

1) Explain Protocol BA

2) Implement any : https://developer.algorand.org/solutions/

2 Solution to Question 1

2.1 BA* Protocol

The BA* (or Byzantine Agreement) protocol is a consensus mechanism used in decentralized networks like Algorand to ensure all participants agree on a common value, even when some nodes may act maliciously. Byzantine Agreement protocols are essential in distributed systems where participants (nodes) may fail or behave unpredictably.

2.2 Key Properties of BA*:

- **Agreement:** All non-faulty nodes must agree on the same value, ensuring that there is no disagreement in the system.
- **Validity:** If all non-faulty nodes propose the same value, the final decision will be that value.
- **Termination:** The protocol must reach a conclusion (final decision) in a finite amount of time.
- **Fault Tolerance:** The protocol can function even if a fraction of the nodes (typically less than 1/3) behave maliciously or unpredictably.

2.3 Steps in the BA* Protocol:

1. **Input Proposal:** Each node in the network proposes an initial value. These values could represent a transaction, a block of transactions, or any other data relevant to the network.

2. **Voting Rounds:** The protocol proceeds in rounds, where nodes exchange their proposals and cast votes based on what they receive from others. Nodes collect votes from other nodes and adjust their proposals accordingly.
3. **Binary Voting:** In each round, nodes vote on a binary decision (0 or 1). The protocol aims to reach a supermajority (typically 2/3) agreement on one value.
4. **Commitment Phase:** Once a supermajority is reached on a particular value, nodes enter a commitment phase, where they lock their votes and refuse to change their value in future rounds.
5. **Final Agreement:** After sufficient rounds of voting, and with enough committed nodes, the system reaches a final decision. This decision is then broadcasted across the network.
6. **Fault Tolerance:** If some nodes act maliciously or do not follow the protocol, BA* ensures that their influence is limited, as long as less than one-third of the nodes are faulty. The remaining honest nodes will still reach a consensus.

2.4 Applications of BA*:

BA* is used in Algorand's consensus mechanism, ensuring that blocks of transactions are added to the blockchain in a secure and reliable manner. It is a key component in achieving Byzantine fault tolerance (BFT) in decentralized networks, allowing nodes to agree on a single value even in the presence of faulty or malicious nodes.

3 Solution for Question 2

3.1 Code

```
// Import Algorand utility libraries
import * as algokit from '@algorandfoundation/algokit-utils';

async function main() {

    const algorand = algokit.AlgorandClient.defaultLocalNet();

    const alice = algorand.account.random();
    const bob = algorand.account.random();

    console.log(alice.addr);

    const dispenser = await algorand.account.dispenser();

    await algorand.send.payment({
```

```
    sender: dispenser.addr,
    receiver: alice.addr,
    amount: algokit.algos(10)
});

console.log("Alice's Account", await
   algorand.account.getInformation(alice.addr));

const createResult = await algorand.send.assetCreate({
    sender: alice.addr,
    total: 100n,
});

console.log("Create Result", createResult);

const assetId = BigInt(createResult.confirmation.assetIndex!);
console.log('assetId', assetId);

await algorand.send.payment({
    sender: dispenser.addr,
    receiver: bob.addr,
    amount: algokit.algos(10)
});

console.log("Bob's MBR pre opt-in", (await
   algorand.account.getInformation(bob.addr)).minBalance);

await algorand.send.assetOptIn({
    sender: bob.addr,
    assetId
});

console.log("Bob's MBR post opt-in", (await
   algorand.account.getInformation(bob.addr)).minBalance);

await algorand.send.assetTransfer({
    sender: alice.addr,
    receiver: bob.addr,
    amount: 1n,
    assetId
});

console.log("Alice's Asset Balance", await
   algorand.account.getAssetInformation(alice.addr, assetId));
```

```
    console.log("Bob's Asset Balance", await
       algorand.account.getAssetInformation(bob.addr, assetId));

    algorand.newGroup().addPayment({
        sender: alice.addr,
        receiver: bob.addr,
        amount: algokit.algos(1)
    }).addAssetTransfer({
        sender: bob.addr,
        assetId,
        receiver: alice.addr,
        amount: 1n
    }).execute();

    console.log("Alice's Asset Balance", await
       algorand.account.getAssetInformation(alice.addr, assetId));
    console.log("Bob's Asset Balance", await
       algorand.account.getAssetInformation(bob.addr, assetId));
    console.log("Bob's MBR post transfer", (await
       algorand.account.getInformation(bob.addr)).minBalance);

    await algorand.send.assetTransfer({
        sender: bob.addr,
        receiver: alice.addr,
        assetId,
        amount: 0n,
        closeAssetTo: alice.addr
    });

    console.log("Bob's MBR post opt-out", (await
       algorand.account.getInformation(bob.addr)).minBalance);
}

main();
```

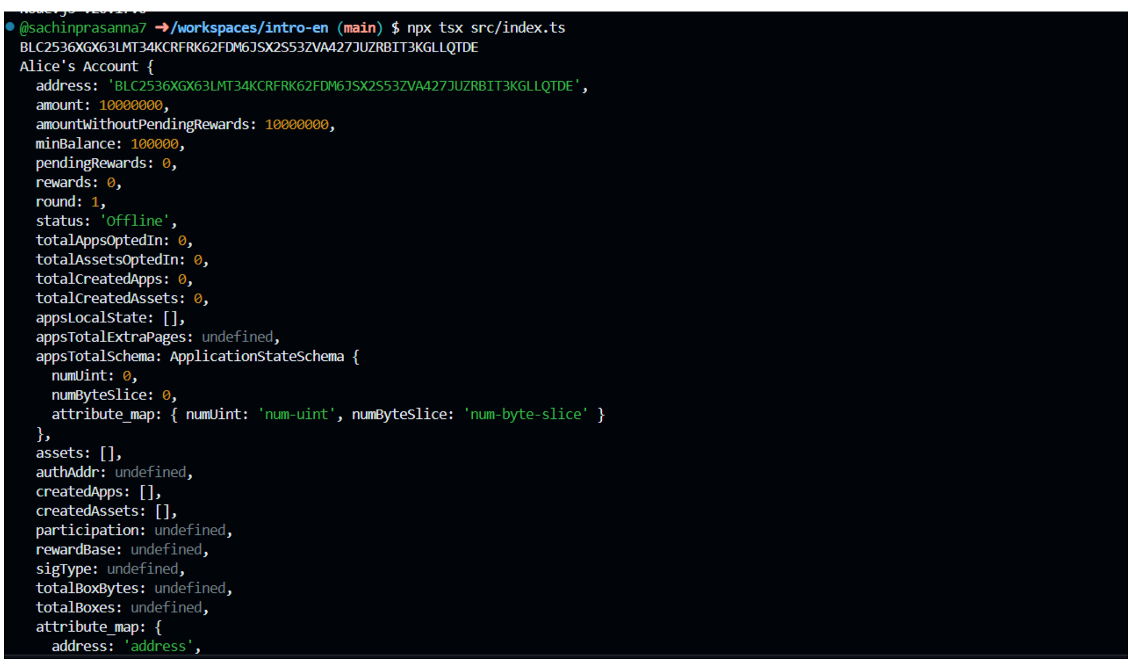
3.2 Functionalities

The Algorand-based solution using the `algokit` library provides the following functionalities:

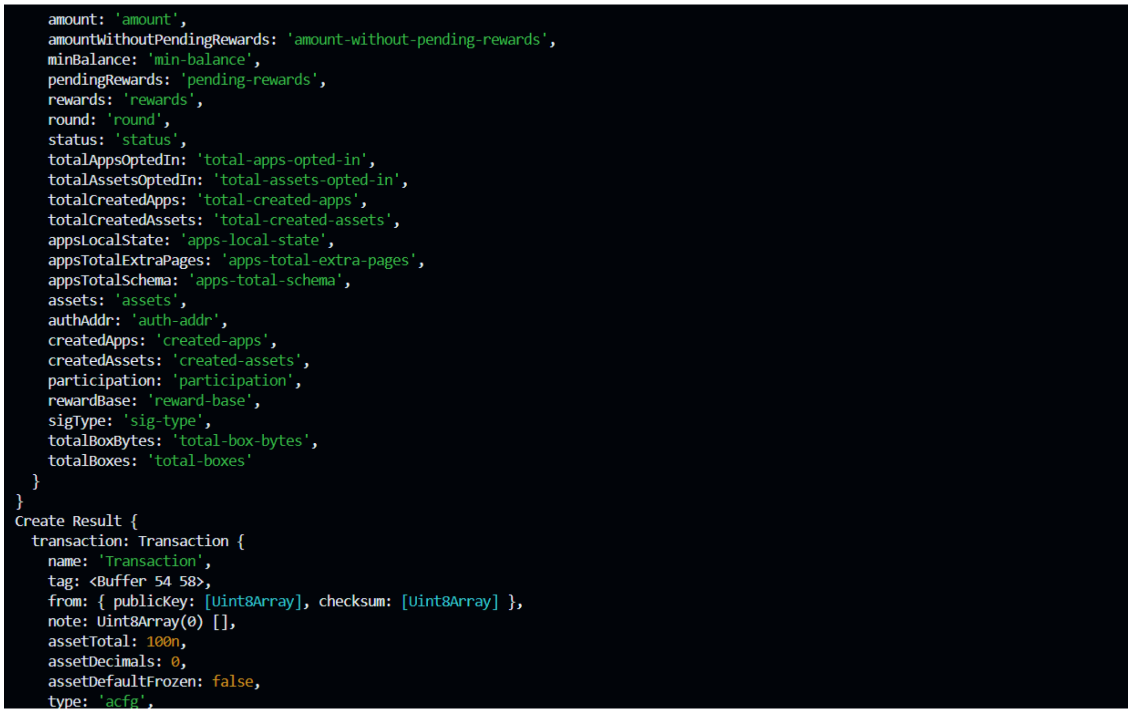
- **Account Generation:** Two random accounts are generated for Alice and Bob using Algorand's account generator.
- **Payment Transfer:** The Algorand dispenser sends payments to Alice and Bob, providing them with 10 Algos each.

- **Asset Creation:** Alice creates an asset with a total supply of 100 units, which can be transferred.
- **Opt-In to Asset:** Bob opts in to receive the asset created by Alice. Opting in is necessary to accept assets on the Algorand blockchain.
- **Asset Transfer:** Alice transfers 1 unit of the asset to Bob.
- **Group Transactions:** A group transaction is performed where Alice sends 1 Algo to Bob, and Bob transfers 1 unit of the asset back to Alice in a single atomic transaction.
- **Close Out Asset:** Bob transfers his remaining asset back to Alice and opts out of holding the asset, closing out his position.

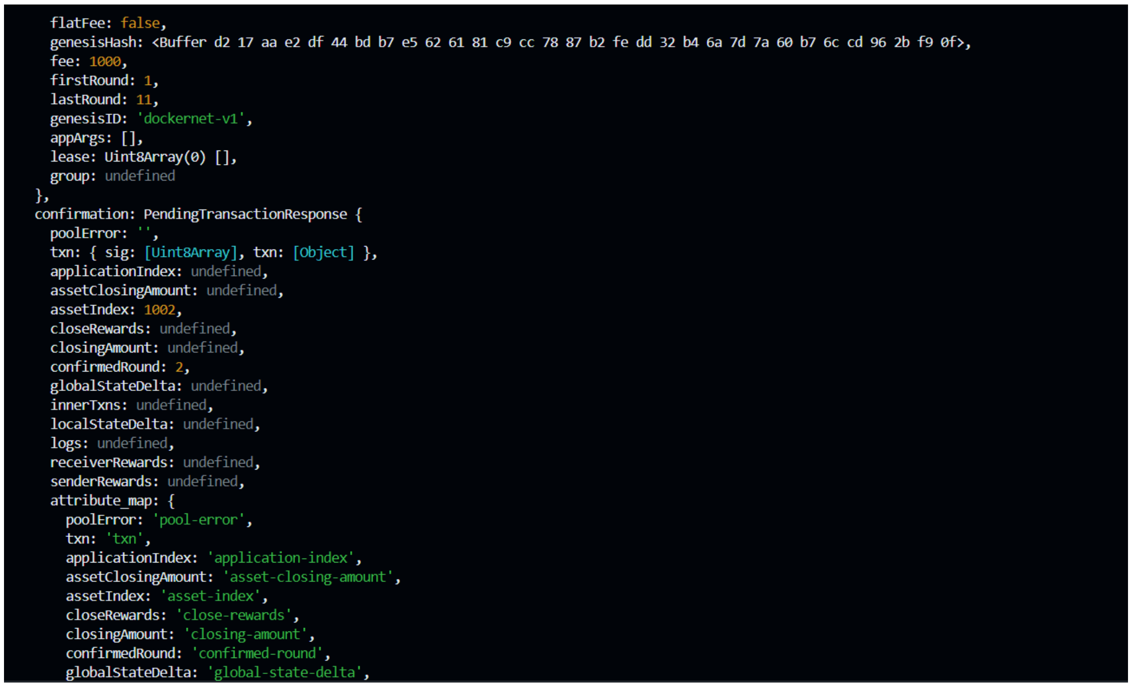
3.3 Outputs



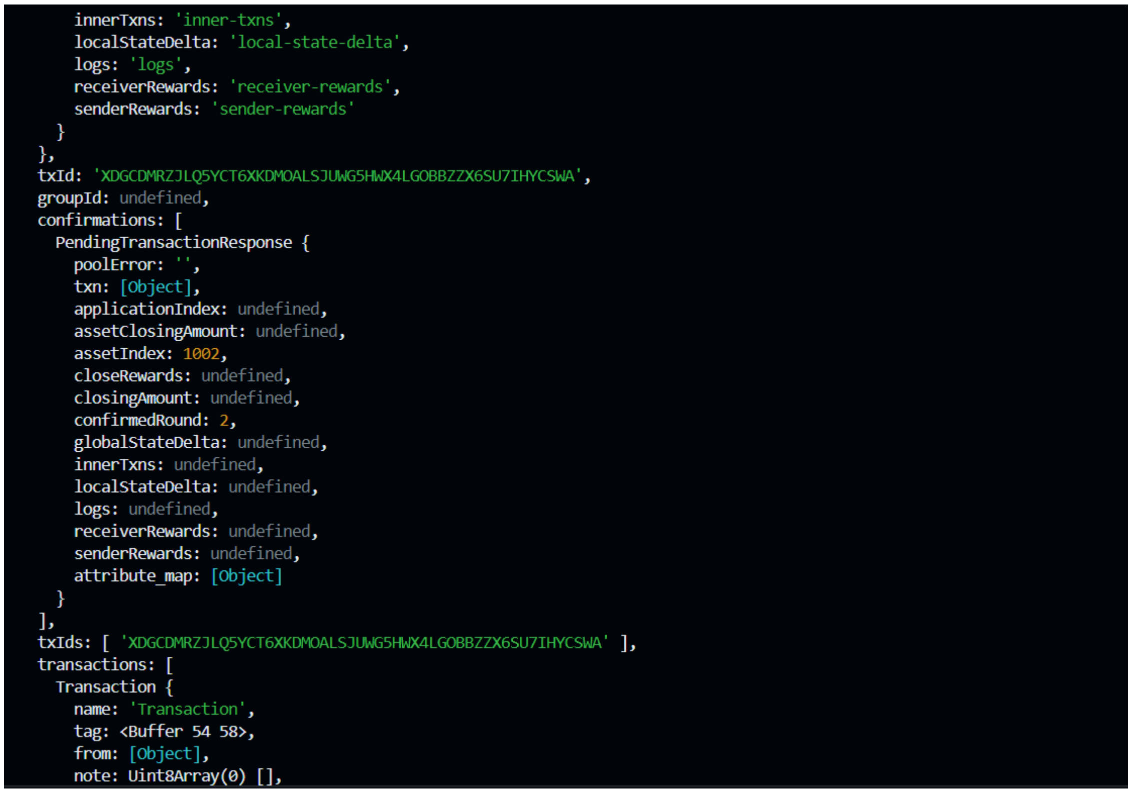
```
● @sachinprasanna7 ➜ /workspaces/intro-en (main) $ npx tsx src/index.ts
  BLC2536XGX63LMT34KCRFRK62FDM6JSX2S53ZVA427JUZRBIT3KGLLQTDE
  Alice's Account {
    address: 'BLC2536XGX63LMT34KCRFRK62FDM6JSX2S53ZVA427JUZRBIT3KGLLQTDE',
    amount: 10000000,
    amountWithoutPendingRewards: 10000000,
    minBalance: 100000,
    pendingRewards: 0,
    rewards: 0,
    round: 1,
    status: 'Offline',
    totalAppsOptedIn: 0,
    totalAssetsOptedIn: 0,
    totalCreatedApps: 0,
    totalCreatedAssets: 0,
    appsLocalState: [],
    appsTotalExtraPages: undefined,
    appsTotalSchema: ApplicationStateSchema {
      numUint: 0,
      numByteSlice: 0,
      attribute_map: { numUint: 'num-uint', numByteSlice: 'num-byte-slice' }
    },
    assets: [],
    authAddr: undefined,
    createdApps: [],
    createdAssets: [],
    participation: undefined,
    rewardBase: undefined,
    sigType: undefined,
    totalBoxBytes: undefined,
    totalBoxes: undefined,
    attribute_map: {
      address: 'address',
```



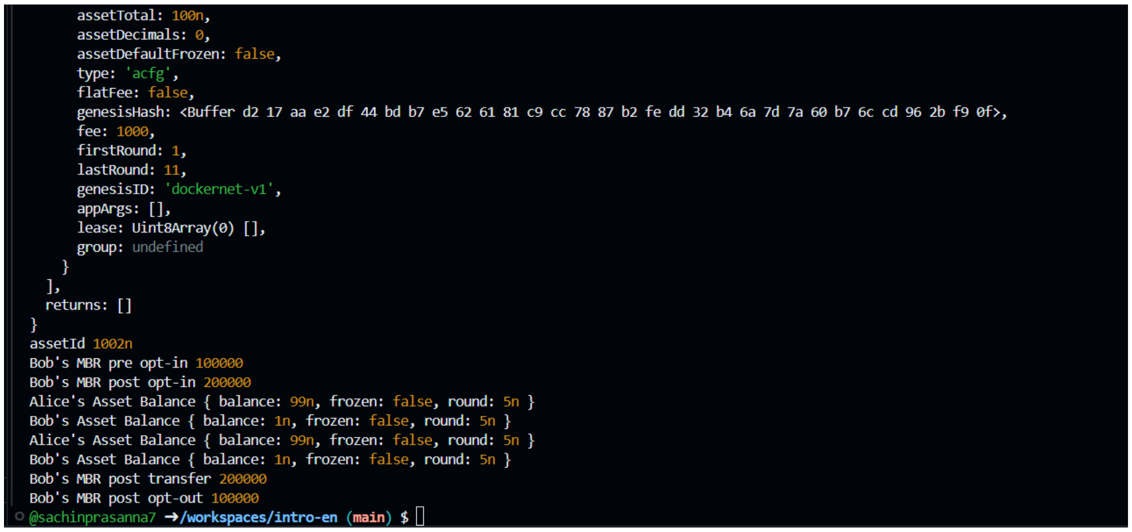
```
    amount: 'amount',
    amountWithoutPendingRewards: 'amount-without-pending-rewards',
    minBalance: 'min-balance',
    pendingRewards: 'pending-rewards',
    rewards: 'rewards',
    round: 'round',
    status: 'status',
    totalAppsOptedIn: 'total-apps-opted-in',
    totalAssetsOptedIn: 'total-assets-opted-in',
    totalCreatedApps: 'total-created-apps',
    totalCreatedAssets: 'total-created-assets',
    appsLocalState: 'apps-local-state',
    appsTotalExtraPages: 'apps-total-extra-pages',
    appsTotalSchema: 'apps-total-schema',
    assets: 'assets',
    authAddr: 'auth-addr',
    createdApps: 'created-apps',
    createdAssets: 'created-assets',
    participation: 'participation',
    rewardBase: 'reward-base',
    sigType: 'sig-type',
    totalBoxBytes: 'total-box-bytes',
    totalBoxes: 'total-boxes'
  }
}
Create Result {
  transaction: Transaction {
    name: 'Transaction',
    tag: <Buffer 54 58>,
    from: { publicKey: [Uint8Array], checksum: [Uint8Array] },
    note: Uint8Array(0) [],
    assetTotal: 100n,
    assetDecimals: 0,
    assetDefaultFrozen: false,
    type: 'acfg',
```



```
    flatFee: false,
    genesisHash: <Buffer d2 17 aa e2 df 44 bd b7 e5 62 61 81 c9 cc 78 87 b2 fe dd 32 b4 6a 7d 7a 60 b7 6c cd 96 2b f9 0f>,
    fee: 1000,
    firstRound: 1,
    lastRound: 11,
    genesisID: 'dockernet-v1',
    appArgs: [],
    lease: Uint8Array(0) [],
    group: undefined
  },
  confirmation: PendingTransactionResponse {
    poolError: '',
    txn: { sig: [Uint8Array], txn: [Object] },
    applicationIndex: undefined,
    assetClosingAmount: undefined,
    assetIndex: 1002,
    closeRewards: undefined,
    closingAmount: undefined,
    confirmedRound: 2,
    globalStateDelta: undefined,
    innerTxns: undefined,
    localStateDelta: undefined,
    logs: undefined,
    receiverRewards: undefined,
    senderRewards: undefined,
    attribute_map: {
      poolError: 'pool-error',
      txn: 'txn',
      applicationIndex: 'application-index',
      assetClosingAmount: 'asset-closing-amount',
      assetIndex: 'asset-index',
      closeRewards: 'close-rewards',
      closingAmount: 'closing-amount',
      confirmedRound: 'confirmed-round',
      globalStateDelta: 'global-state-delta',
```



```
      innerTxns: 'inner-txns',
      localStateDelta: 'local-state-delta',
      logs: 'logs',
      receiverRewards: 'receiver-rewards',
      senderRewards: 'sender-rewards'
    }
  },
  txId: 'XDGCDMRZJLQ5YCT6XKDMOALSJUWG5HWX4LGOBBZZX6SU7IHYCSWA',
  groupId: undefined,
  confirmations: [
    PendingTransactionResponse {
      poolError: '',
      txn: [Object],
      applicationIndex: undefined,
      assetClosingAmount: undefined,
      assetIndex: 1002,
      closeRewards: undefined,
      closingAmount: undefined,
      confirmedRound: 2,
      globalStateDelta: undefined,
      innerTxns: undefined,
      localStateDelta: undefined,
      logs: undefined,
      receiverRewards: undefined,
      senderRewards: undefined,
      attribute_map: [Object]
    }
  ],
  txIds: [ 'XDGCDMRZJLQ5YCT6XKDMOALSJUWG5HWX4LGOBBZZX6SU7IHYCSWA' ],
  transactions: [
    Transaction {
      name: 'Transaction',
      tag: <Buffer 54 58>,
      from: [Object],
      note: Uint8Array(0) [],
```



```
      assetTotal: 100n,
      assetDecimals: 0,
      assetDefaultFrozen: false,
      type: 'acfg',
      flatFee: false,
      genesisHash: <Buffer d2 17 aa e2 df 44 bd b7 e5 62 61 81 c9 cc 78 87 b2 fe dd 32 b4 6a 7d 7a 60 b7 6c cd 96 2b f9 0f>,
      fee: 1000,
      firstRound: 1,
      lastRound: 11,
      genesisID: 'dockernet-v1',
      appArgs: [],
      lease: Uint8Array(0) [],
      group: undefined
    }
  ],
  returns: []
}
assetId 1002n
Bob's MBR pre opt-in 100000
Bob's MBR post opt-in 200000
Alice's Asset Balance { balance: 99n, frozen: false, round: 5n }
Bob's Asset Balance { balance: 1n, frozen: false, round: 5n }
Alice's Asset Balance { balance: 99n, frozen: false, round: 5n }
Bob's Asset Balance { balance: 1n, frozen: false, round: 5n }
Bob's MBR post transfer 200000
Bob's MBR post opt-out 100000
○ @sachinprasanna7 ➜ /workspaces/intro-en (main) $ 
```