**Tutorial 16**

**Name**:Bhavin Patil

**Roll No:** 66

**TY-CS-D**

SIMULATION OF BLOCKCHAIN

**Blockchain Technology**

Transactions between parties in current systems are usually conducted in a centralised form, which requires the involvement of a trusted third party (e.g., a bank). However, this could result in security issues (e.g., unauthorised modifications). Blockchain technology has emerged to tackle these issues by allowing non-trusting entities to interact with each other in a distributed manner without the involvement of a trusted third party.

The term blockchain comes from the fact that data about multiple transactions is grouped into blocks. Each block is uniquely identified by its cryptographic hash, and each block is attached and linked to the one that came before it. This results in a chain of blocks. Once a block is generated and attached to the blockchain ledger, the transactions in that block cannot be modified by any node, since it would require the node to rewrite all subsequent blocks. This makes blockchain systems immutable and protected against double-spending attacks

**Modelling And Simulation Of Blockchain Technology**

Modelling is the method of producing an abstract model that represents a real system of interest, either existing or in design. A system model provides a close approximation of the system it represents by integrating most of its features while at the same time is much simpler than the real system. That is, an excellent model should consider the trade-off between simplicity and realism. A common practice when developing a model is to start with a simple one and then gradually increase the complexity of the model.

**Simulation** is a quantitative method, which ‘executes’ the model to mimic the behaviour of the system [12]. It is quite often that experimentation with a real system is not feasible, impractical or very expensive [72]. Simulation, however, allows experimentation with a model without having to interrupt the real system (if it exists) or implementing a new system for that purpose. With simulation, it is possible to explore different design trade-offs and configuration questions for the system at hand in a timely manner. Simulation can also be used to predict and describe how different conditions and scenarios impact the behaviour of the system. Thus, simulation can be used to answer “What if” questions and to experiment with new designs and policies without interrupting the real system.

In general, simulations are a useful tool that can be utilised to study the performance of a system (either existing or proposed one) under a variety of design configurations as well as over a long period of time. Simulation is used before building a new system or changing an existing one. This is to eliminate potential failures, avoid unseen problems and bottlenecks and to improve the performance of the system [72]. Simulation can be classified into two categories, namely, discrete-event simulation and continuous-event simulation [52]. A discrete-event simulation model is both dynamic and stochastic in which the state variables change at discrete moments in time. Human-made systems such as digital computer and information systems are most suitable represented as discrete-event simulation. The focus of this thesis is on discrete-event simulation. In Chapter 4, for instance, we propose a simulator tool named BlockSim that is based on a discrete-event simulation approach.

There are two approaches to develop simulation models and tools, namely, generalpurpose programming languages (e.g., C++, Java or Python) and special-purpose simulation languages (e.g., Arena and GPSS) [65]. The former is more flexible and familiar, while the latter provides several built-in features (e.g., statistics, event scheduler and animation) that reduce the time required to build models. As stated in [65], there is a debate and conflict about which method is preferable. Also worth noting are simulation frameworks that enable developing simulation models using general purpose languages, for instance, OMNeT++ and SimPy for developing models in C++ and Python. We select Python as a general-purpose programming language to develop and implement the BlockSim simulator\

**Python program to create Blockchain**

import datetime

import hashlib

import JSON

from flask import Flask, jsonify

class Blockchain:

def \_\_init\_\_(self):

self.chain = []

self.create\_block(proof=1, previous\_hash='0')

def create\_block(self, proof, previous\_hash):

block = {'index': len(self.chain) + 1,

'timestamp': str(datetime.datetime.now()),

'proof': proof,

'previous\_hash': previous\_hash}

self.chain.append(block)

return block

def print\_previous\_block(self):

return self.chain[-1]

def proof\_of\_work(self, previous\_proof):

new\_proof = 1

check\_proof = False

while check\_proof is False:

hash\_operation = hashlib.sha256(

str(new\_proof\*\*2 - previous\_proof\*\*2).encode()).hexdigest()

if hash\_operation[:5] == '00000':

check\_proof = True

else:

new\_proof += 1

return new\_proof

def hash(self, block):

encoded\_block = json.dumps(block, sort\_keys=True).encode()

return hashlib.sha256(encoded\_block).hexdigest()

def chain\_valid(self, chain):

previous\_block = chain[0]

block\_index = 1

while block\_index < len(chain):

block = chain[block\_index]

if block['previous\_hash'] != self.hash(previous\_block):

return False

previous\_proof = previous\_block['proof']

proof = block['proof']

hash\_operation = hashlib.sha256(

str(proof\*\*2 - previous\_proof\*\*2).encode()).hexdigest()

if hash\_operation[:5] != '00000':

return False

previous\_block = block

block\_index += 1

return True

app = Flask(\_\_name\_\_)

blockchain = Blockchain()

@app.route('/mine\_block', methods=['GET'])

def mine\_block():

previous\_block = blockchain.print\_previous\_block()

previous\_proof = previous\_block['proof']

proof = blockchain.proof\_of\_work(previous\_proof)

previous\_hash = blockchain.hash(previous\_block)

block = blockchain.create\_block(proof, previous\_hash)

response = {'message': 'A block is MINED',

'index': block['index'],

'timestamp': block['timestamp'],

'proof': block['proof'],

'previous\_hash': block['previous\_hash']}

return jsonify(response), 200

@app.route('/get\_chain', methods=['GET'])

def display\_chain():

response = {'chain': blockchain.chain,

'length': len(blockchain.chain)}

return jsonify(response), 200

@app.route('/valid', methods=['GET'])

def valid():

valid = blockchain.chain\_valid(blockchain.chain)

if valid:

response = {'message': 'The Blockchain is valid.'}

else:

response = {'message': 'The Blockchain is not valid.'}

return jsonify(response), 200

app.run(host='127.0.0.1', port=5000)

**Output (mine\_block):**

{

"index":2,

"message":"A block is MINED",

"previous\_hash":"2d83a826f87415edb31b7e12b35949b9dbf702aee7e 383cbab119456847b957c",

"proof":533,

"timestamp":"2020-06-01 22:47:59.309000"

}

**Output (get\_chain):**

{

"chain":[{"index":1,

"previous\_hash":"0",

"proof":1,

"timestamp":"2020-06-01 22:47:05.915000"},{"index":2,

"previous\_hash":"2d83a826f87415edb31b7e12b35949b9dbf702aee7e383cbab119456847b957c",

"proof":533,

"timestamp":"2020-06-01 22:47:59.309000"}],

"length":2

}

**Output(valid):**

{"message":"The Blockchain is valid."}