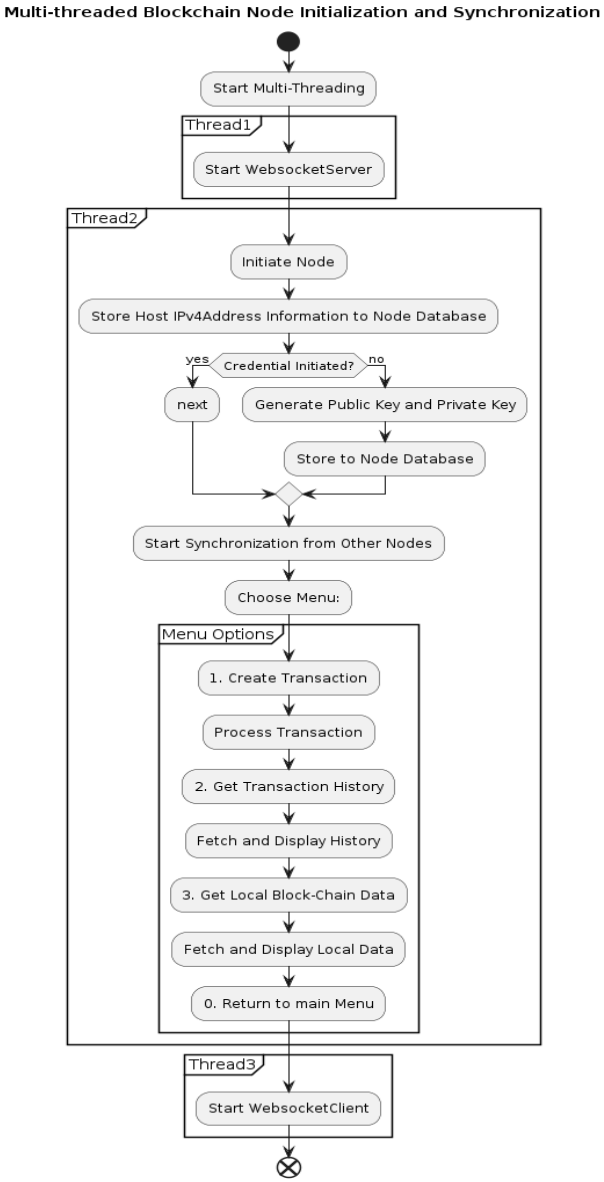
Permodelan Graf dalam Jaringan Peer-To-Peer pada Teknologi Block-Chain menggunakan WebSocket

By Abdan Hafidz

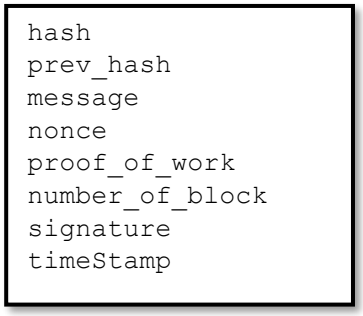
Instalasi

Requirements :
mnemonic==0.21
asyncio==3.4.3
websockets==12.0
urllib3==1.26.14
ecdsa==0.19.0



1. Blockchain

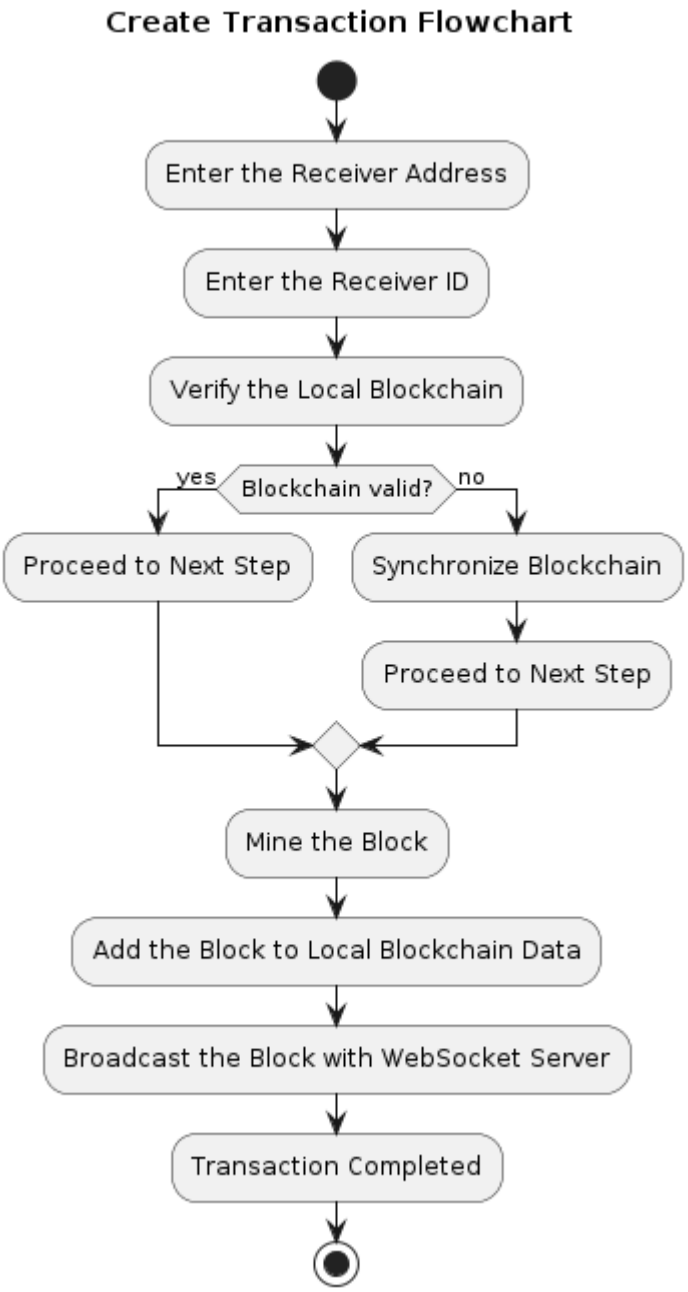
Penggunaan block chain sebagai model fitur yang diimplementasikan dalam teknologi berbasis desentralisasi dan memanfaatkan jaringan Peer-To-Peer sebagai media komunikasi.




```
class Block:
    def __init__(self, hash = "",
                 prev_hash = "",
                 message = "",
                 nonce = "",
                 proof_of_work = "",
                 number_of_block = 0,
                 timeStamp = "",
                 signature = ""):
        self.hash = hash
        self.prev_hash = prev_hash
        self.message = message
        self.nonce = nonce
        self.proof_of_work = proof_of_work
        self.number_of_block = number_of_block
        self.signature = signature
        self.timeStamp = str(datetime.datetime.now())
```

- `hash` : akan berisikan hasil pemetaan data dari fungsi yang diterapkan ke dalam bentuk sederhana yang disebut *digest.*
- `prev_hash` : adalah hash dari block sebelumnya.
- `message` : adalah informasi yang akan kita simpan ke dalam block.
- `nonce` : adalah angka acak yang nantinya harus melalui proses komputasi atau biasa disebut *mining* (penambangan) sehingga jika di-substitusikan ke fungsi hash akan memenuhi algoritma konsensus yang ditetapkan.
- `proof_of_work` : adalah jumlah langkah komputasi yang diperlukan untuk menemukan *nonce* yang memenuhi algoritma konsensus saat di-substitusikan.
- `number_of_block` : adalah nomor blok saat ini
- `signature` : adalah mekanisme kriptografi yang memastikan keamanan dan keaslian transaksi. Sebuah *signature* menjadi pengidentifikasi untuk informasi yang disimpan akan sesuai kepemilikannya dengan pembuat blok. Keabsahan blok diverifikasi salah satu langkahnya melalui verifikasi *signature* yang dimiliki.
- `timeStamp` : menunjukkan waktu lokal blok ditambang.

1.1. Penambangan Blok



Blok baru bisa dihasilkan dengan melakukan proses tambang. Penambangan dilakukan dengan cara menyimpan rekord informasi ke dalam model blok, kemudian kita akan mengisi atribut hash beserta nonce dengan serangkaian proses yang menerapkan algoritma konsensus.

1.1.1. Hashing dan Algoritma Konsensus

Atribut Hash akan berisikan hasil pemetaan data dari fungsi yang diterapkan ke dalam bentuk sederhana yang disebut digest. Adapun fungsi hashing yang digunakan adalah sebagai berikut :

```
sha256(currentBlock, prev_hash, signature, nonce,
timeStamp)
```

Nilai hash yang dihasilkan harus memenuhi algoritma konsensus, dalam proyek ini kami menerapkan kesepakatan bersama bahwa hasil hash yaitu berupa digest hexadesimal akan memuat prefix bit ‘0’ sebanyak nilai `DIFFICULTY`. Nilai

`DIFFICULTY` akan menjadi parameter terhadap nonce yang akan disubstitusikan ke dalam hash function.

1.1.2. Nonce Mining

Kita akan melakukan proses dalam mencari nilai nonce yang memenuhi sehingga ketika disubstitusikan ke fungsi hash akan memenuhi algoritma konsensus. Tentunya untuk menemukan nilai ini memerlukan komputasi yang cukup banyak karena kita akan melakukan pengujian satu persatu angka dari rentang 1 sampai 2^{256} dan berapa langkah diperlukan untuk menemukan nonce yang sesuai akan menjadi nilai dari atribute `proof_of_work`.

```
for mineNonce in range(1,2**(256)):
    hash_hex = str(Hash.hash(block.message,
previous_hash, signature, mineNonce, block.timeStamp))
    hash_binary = ''.join(format(ord(x), '08b') for x
in hash_hex)
            if(str(hash_binary)[:2] == "00"):
                block.prev_hash = previous_hash
                block.nonce = mineNonce
                block.hash = hash_hex
                block.proof_of_work = works
                break
            else:works+=1
```

1.1.3. Signature

Pembuatan *signature* memerlukan *private key* dan *public key*.Proyek ini menggunakan *Ecdsa SECP256k1* sebagai generator private key dan public key. Sebuah *signature* dihasilkan melalui pemetaan fungsi yang menjadika informasi dan *private key* sebagai parameter nantinya:

```
def sign_message(private_key, message):
    return private_key.sign(message.encode('utf-8'))
```

1.2. Verifikasi Blok

Blok yang dihasilkan, diterima, ataupun didistribusikan harus melalui tahap verifikasi keabsahan dan kesesuaian antara transaksi dan parameter validitas seperti hash dan signature.

```
def verify_block(self,
block:Union[DictObj,'Block',Block])->BlockVerifyStatus:
        HashHex = str(Hash.hash(block.message,
self.BlockChainData[-1]["hash"], block.signature,
block.nonce, block.timeStamp))
        PublicKey =
KeyGenerator.load_public_key_from_hex(KeyGenerator.
ParseSender(block.message))
        VerifySignature =
KeyGenerator.verify_signature(PublicKey,block.message,
block.signature)
        VerifyStatus = BlockVerifyStatus()
        if(self.BlockChainData[-1]["hash"] ==
block.prev_hash and
           block.hash == HashHex and
           VerifySignature and
           block.number_of_block ==
self.BlockChainData[-1]["number_of_block"] + 1):
            return True
```

1.2.1. Verifikasi Hash

Hash yang ada pada blok harus diverifikasi kesesuaiannya dengan rekord transaksi yang ada pada blok, begitu juga dengan atribut `prev_hash` harus divalidasi apakah benar – benar sesuai dengan nilai atribut hash pada blok sebelumnya.

```
else:
            if(self.BlockChainData[-1]["hash"] !=
block.prev_hash):
                VerifyStatus.PreviousHashBlockError =
True
                if(self.BlockChainData[-1]["prev_hash"]
== block.prev_hash):
                    VerifyStatus.BlocksFork = True
            if(block.hash != HashHex):
                VerifyStatus.InvalidBlockHash = True
```

1.2.2. Verifikasi Signature

Verifikasi *signature* dilakukan dengan cara memetakan public key dan informasi ke dalam sebuah fungsi.

```
def verify_signature(public_key, message, signature):
    try:
        return
public_key.verify(bytes.fromhex(signature),
message.encode('utf-8'))
    except BadSignatureError:
        return False
```

```
VerifySignature =
KeyGenerator.verify_signature(PublicKey,
block.message, block.signature)
```

```
if(not VerifySignature):
     VerifyStatus.InvalidSignature = True
```

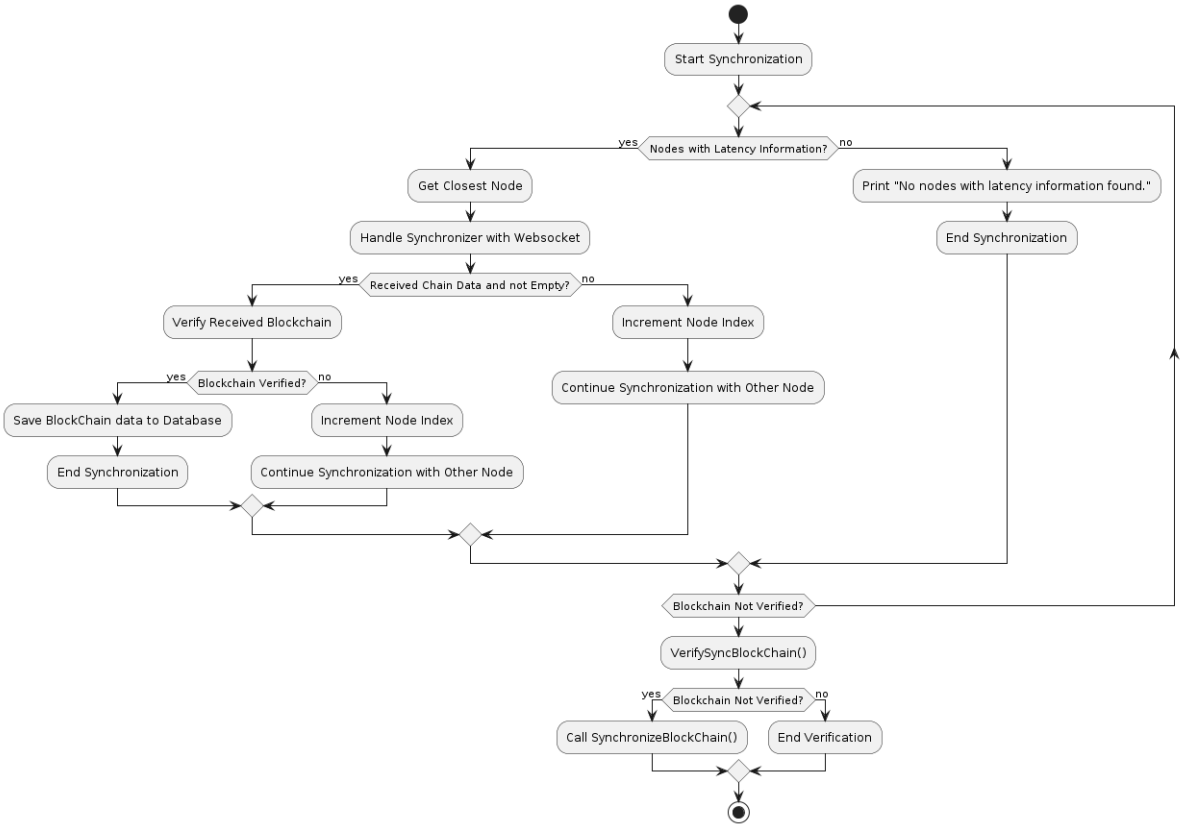
1.2.3. Verifikasi Urutan Blok

Atribut `number_of_block` harus divalidasi kesesuaiannya untuk menunjukkan apakah data blockchain yang dimiliki sinkron dengan orang lain.

```
if(block.number_of_block != self.BlockChainData[-
1]["number_of_block"] + 1):
            VerifyStatus.MissedBlock = True
            VerifyStatus.BlockNumber =
block.number_of_block
```

2. Sinkronisasi Antar Node

Node akan memenuhi beberapa kondisi nantinya, seperti : (1) Node baru menginisiasi diri di dalam jaringan; (2) Node tertinggal siaran blok akibat offline; (3) Node memiliki data block-chain lokal yang invalid; (4) Node menerima siaran block atau menambang blok kemudian menyiarkannya; (5) Kondisi lainnya di mana block-chain pada node tidak mengikuti algoritma konsensus dan ada perbedaan rantai atau data blok dengan node lainnya, untuk mengatasi hal ini diperlukan langkah sinkronisasi agar node bisa mengejar ketertinggalan block-chain atau mendapatkan data block-chain lokal yang valid.



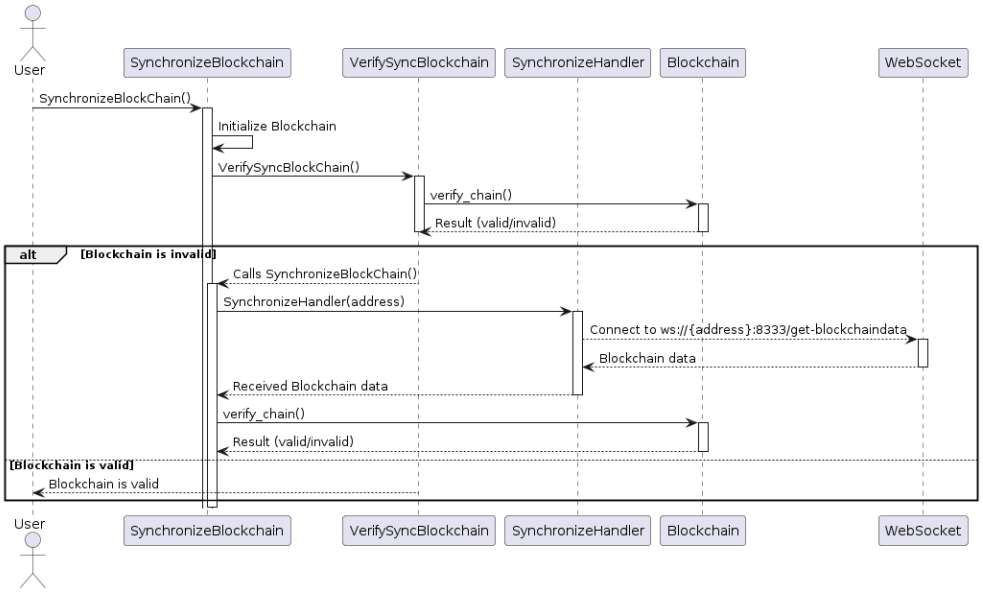
Pemilihan node yang akan dimintai sinkronisasi memerlukan pendekatan tertentu dengan konsep permodelan graf yang akan dibahas pada bab selanjutnya.

3. Peer-To-Peer Network

P2P Network memungkinkan kita untuk membuat komunikasi dalam jaringan tanpa harus melibatkan peladen atau perantara. Seseorang bisa berkomunikasi secara langsung. Pemilihan distribusi dengan jaringan ini sangat sesuai penerapannya dengan prinsip Desentralisasi. Pembuatan jaringan P2P dapat dilakukan dengan metode *socket programming* yang dipermudah dengan ketersediaan library websocket pada bahasa Pemrograman Python.

```
import asyncio
import websockets
```

3.1. Websocket Synchronization Network Handler

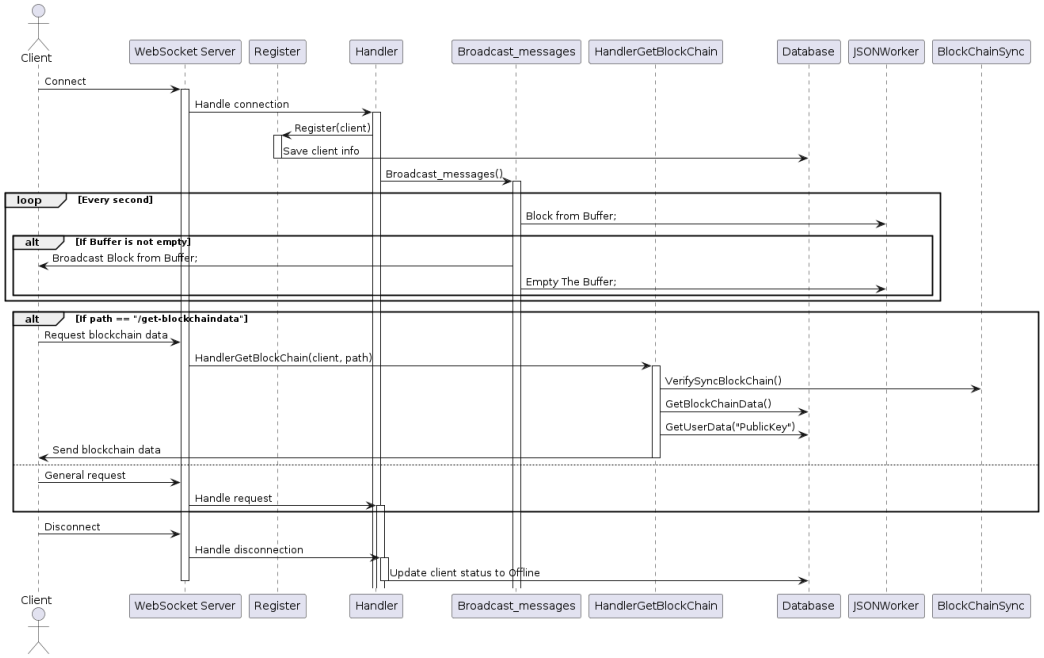


Saat sinkronisasi dilakukan, kita akan membuat koneksi ke seseorang yang kita sebut sebagai peer pada port 8333, di mana port ini telah kita tetapkan sebagai transport protocol untuk handling interaksi blockchain.

```
def SynchronizeHandler(address):
      with connect(f"ws://{address}:8333/get-
blockchaindata") as websocket:
        while True:
            message = websocket.recv()
            if(message != ""):
                data = json.loads(message)
                return data
```

Seseorang bisa memperoleh data block chain orang lain dengan mengakses jaringan melalui path “get-blockchaindata”. Peer yang kita pilih adalah peer dengan latensi minimum sesuai dengan yang telah dijelaskan pada bagian awal.

3.2. Websocket Server Broadcast Network Handler



Websocket akan membuat server di mana orang lain bisa terkoneksi dengan ini. Setelah blok berhasil ditambang, blok akan disimpan terlebih dahulu pada *buffer* kemudian disiarkan ke semua peer yang online.

```
async def main():
    hostname = socket.gethostname()
    NodeAddress = socket.gethostbyname(hostname)
    JSONWorker.EditDataJSON('Database/NodeDB.json',
"NodeAddress",
NodeAddress)
    print("Server Network Panel : ")
    async with websockets.serve(route,
DB.GetUserData("NodeAddress"), 8333):
        # print("Server Is Starting ... ")
        await asyncio.Future()
```

Peer akan dideteksi dan didata sebagai node di dalam jaringan saat melakukan koneksi ke server.

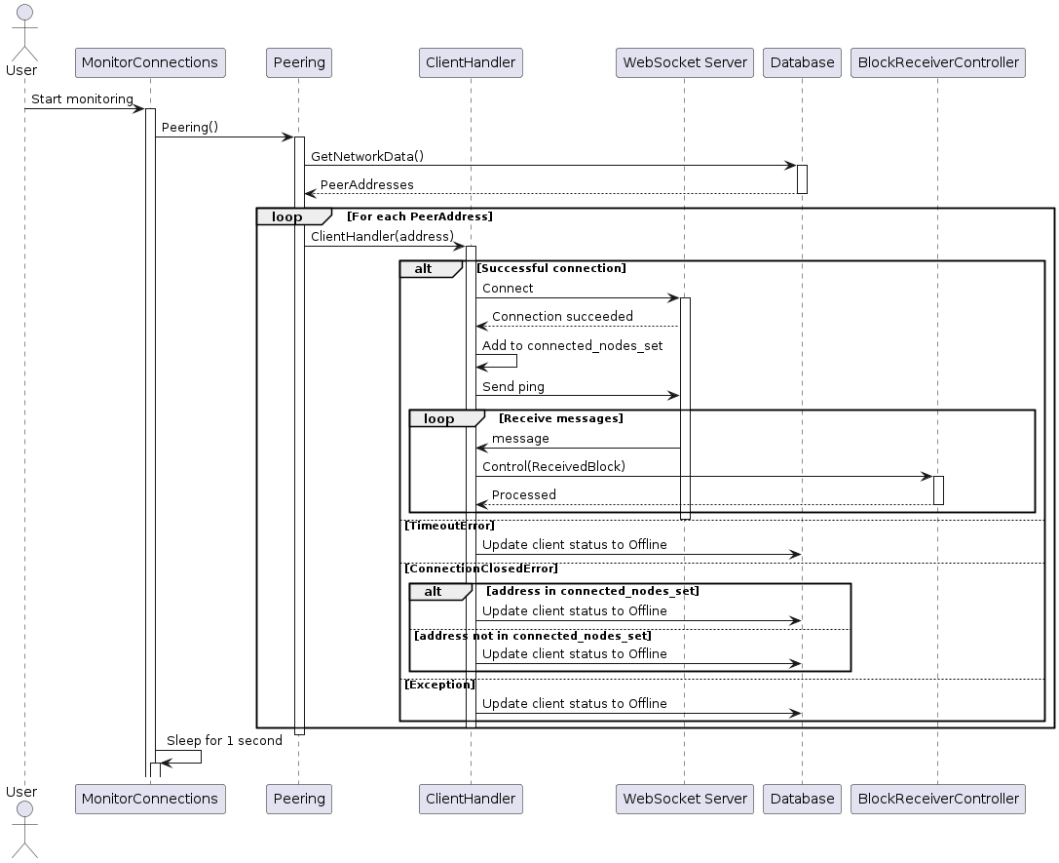
```
async def Register(websocket):
    client_host = websocket.remote_address[0]
    start_time = time.time()
    await websocket.ping()
    await websocket.recv()
    latency = time.time() - start_time
    dict = {"IP":client_host,
           "LastChanged":str(datetime.datetime.now()),
            "Latency":latency,
            "Status":"Online"
            }
    DB.CreateNetworkData(client_host, dict)
```

Penyiaran blok dilakukan ke semua node terhubung dan distribusi blok dilakukan secara asinkronus, di mana semua pesan dikirim secara bersamaan tanpa harus menunggu satu sama lain.

```
async def Handler(websocket):
    CONNECTIONS.add(websocket)
    await Register(websocket)
    await Broadcast_messages()
```

Server juga akan mencatat aktifitas dari node yang terhubung, jika node melepaskan koneksinya maka akan didata sebagai *offline node*.

3.3. Web Socket Client Network Handler



Blok yang disiarkan akan diterima oleh masing – masing node melalui handler yang dijalankan oleh websocket client. Pertama client akan membuat koneksi dengan semua peer dengan menghubungkannya ke alamat server pada port 8333.

```
async def ClientHandler(address):
    try:
        async with
websockets.connect(f"ws://{address}:8333") as
websocket:
            print(f"Connection to {address}
succeeded")
            connected_nodes_set.add(address)  # Tandai
bahwa node berhasil terkoneksi
            await websocket.send("ping")
            while True:
                message = await websocket.recv()
                if message != '':
                    ReceivedBlock =
json.loads(json.dumpmessage)
```

```
BlockReceiverController.Control(ReceivedBlock)
```

Blok yang diterima dari jaringan akan diverifikasi dan berikutnya ditambahkan ke database blockchain lokal. Proses ini juga akan melakukan penanganan terhadap salah satu skenario seperti blok forks yang akan di bahas pada bab berikutnya.

4. Konkurensi

Pekerjaan – pekerjaan seperti multithreading dan broadcasting akan sangat efisien jika beberapa proses dapat dilakukan dalam waktu bersamaan. Teknik Konkurensi diperlukan dalam hal ini dan ketersediaan library asyncio dapat membantu pengembangan.

```
import asyncio
```

Sebagai contoh pada client dan server handler dijalankan secara konkurens dengan menggunakan fitur fungsi asinkronus.

```
from Network import WSClientHandler
import asyncio

asyncio.run(WSClientHandler.main())
```

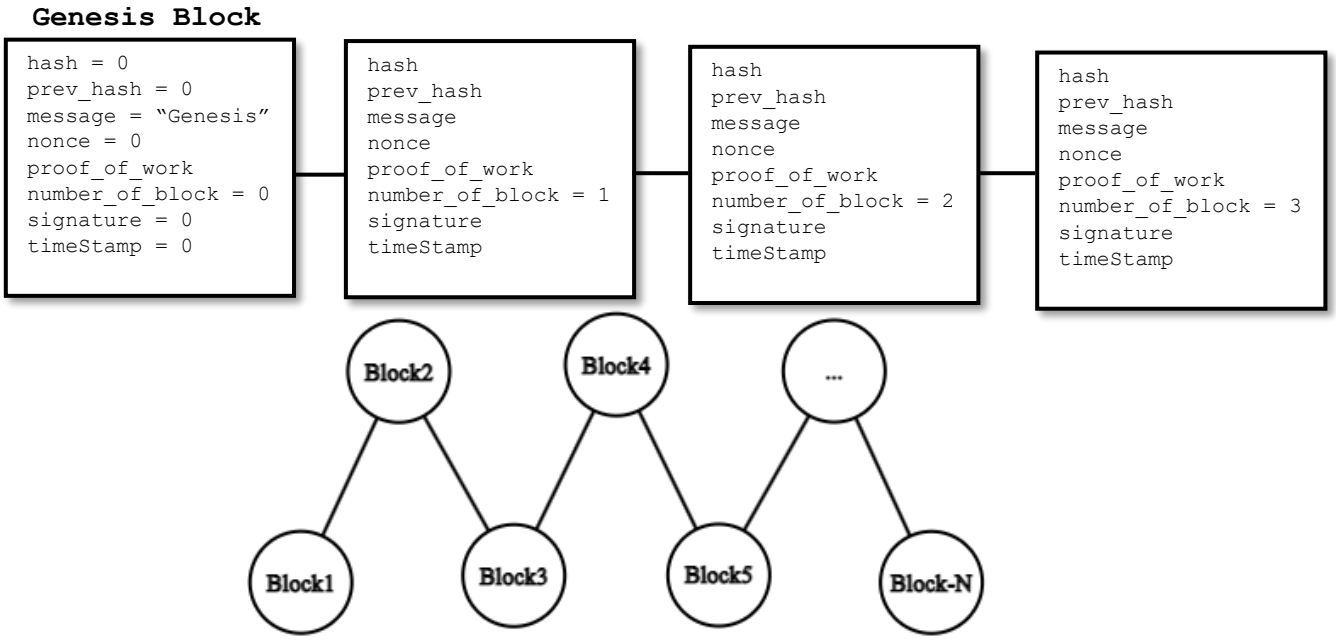
```
from Network import WSServerHandler
import asyncio

asyncio.run(WSServerHandler.main())
```

PENYELESAIAN MASALAH

Rumusan Masalah : Bagaimana permodelan graf dalam Permodelan Graf dalam Jaringan Peer-To-Peer pada Teknologi Block-Chain menggunakan WebSocket ?

1. **Full – Node Block Chain**

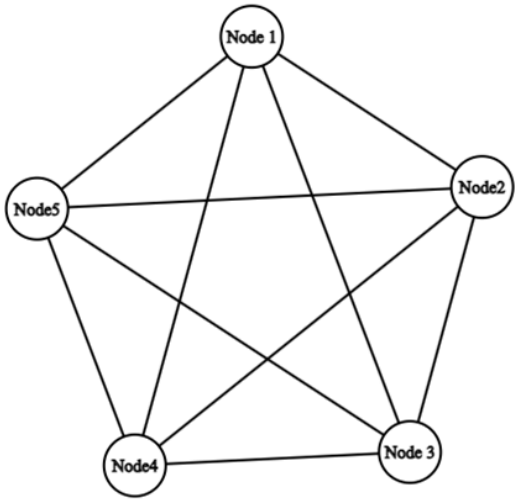
   Model blok memuat informasi dari blok sebelumnya yaitu `prev_hash` , sehingga dapat ditemukan bahwa antar blok saling terhubung dan keterhubungan antar blok ini akan membentuk sebuah rantai yang disebut rantai blok (*Block-Chain)*. Satu rantai blok yang mempunyai *N* blok dapat direpresentasikan sebagai *Strong Connected and Directed Graph* yang mempunyai *N – 1* sisi.



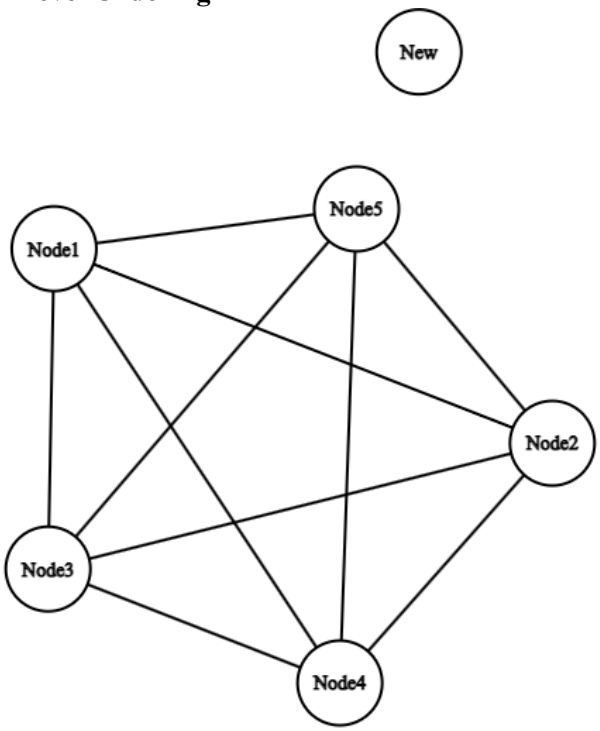
2. **Complete Graph Distributed Peer-To-Peer Network**

   Jaringan Peer-To-Peer adalah dasar dari desentralisasi yang dibahas, di mana dengan metode jaringan ini seseorang mampu berkomunikasi dengan orang lainnya tanpa harus melalui perantara atau pihak ketiga. Seseorang dalam jaringan atau kita sebagai node dapat direpresentasikan sebagai simpul graf.

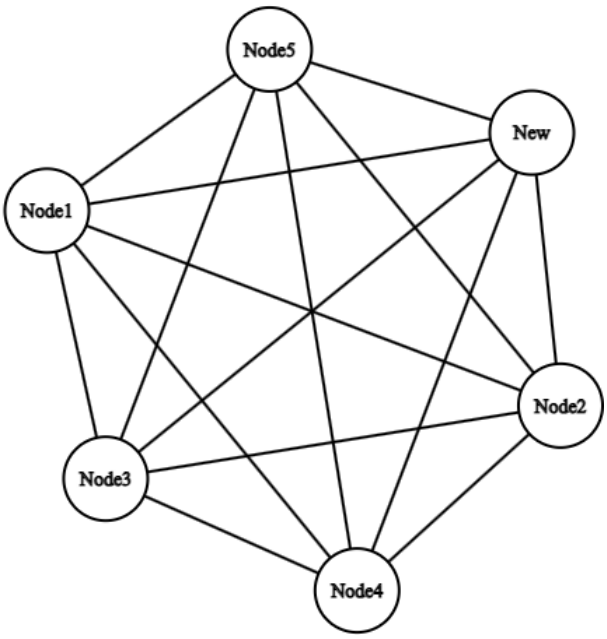
   Semua orang dalam jaringan terhubung satu sama lain, sehingga untuk jumlah orang dalam jaringan sebanyak N, setiap orang akan terhubung ke N – 1 orang lainnya. Ini mengakibatkan terbentuknya graf komplit dengan $\frac{N(N-1)}{2}$ sisi. Inisiasi simpul dalam graf atau seseorang yang bergabung dalam jaringan berdasarkan alamat *host-IP* orang tersebut.



3. **Inisiasi Node dengan Level Ordering**



Saat node baru bergabung dalam jaringan, maka ia akan mengkoneksikan dirinya dengan semua node yang ada, kemudian node akan meminta sinkronisasi data block-chain dan informasi node lain kepada node terdekat. Proses koneksi ke blok lain dilakukan dengan metode level – ordering, sehingga mampu mengkoneksikan semua node dalam waktu bersamaan tanpa harus bergantian, dalam hal ini diperlukan metode pemrograman konkurens dengan fungsi asinkronus.

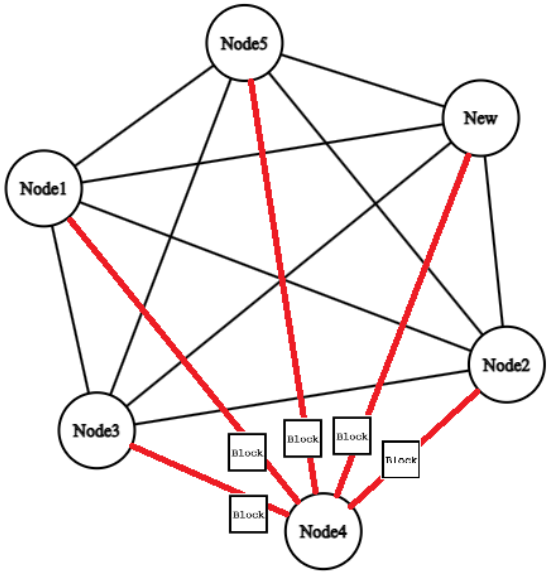


Perhitungan node terdekat dengan cara menentukan bobot sisi graf berdasarkan latensi koneksi antar dua node. Sinkronisasi node akan dibahas di bagian berikutnya.

```
async def Peering():
    NetData = DB.GetNetworkData()
    PeerAddresses = list(NetData["Nodes"])
    tasks = [ClientHandler(uri) for uri in PeerAddresses]
    await asyncio.gather(*tasks)
```

4. Penyiaran dengan Level-Ordering

Sebuah node menambang block kemudian disiarkan ke semua orang melalui jaringan, dari sini kita akan memahami bahwa blok akan terdistribusi secara konkurens atau semua node dapat menerima block dalam waktu bersamaan tanpa harus bergantian, ini adalah *level ordering transverse* dari sebuah graf.

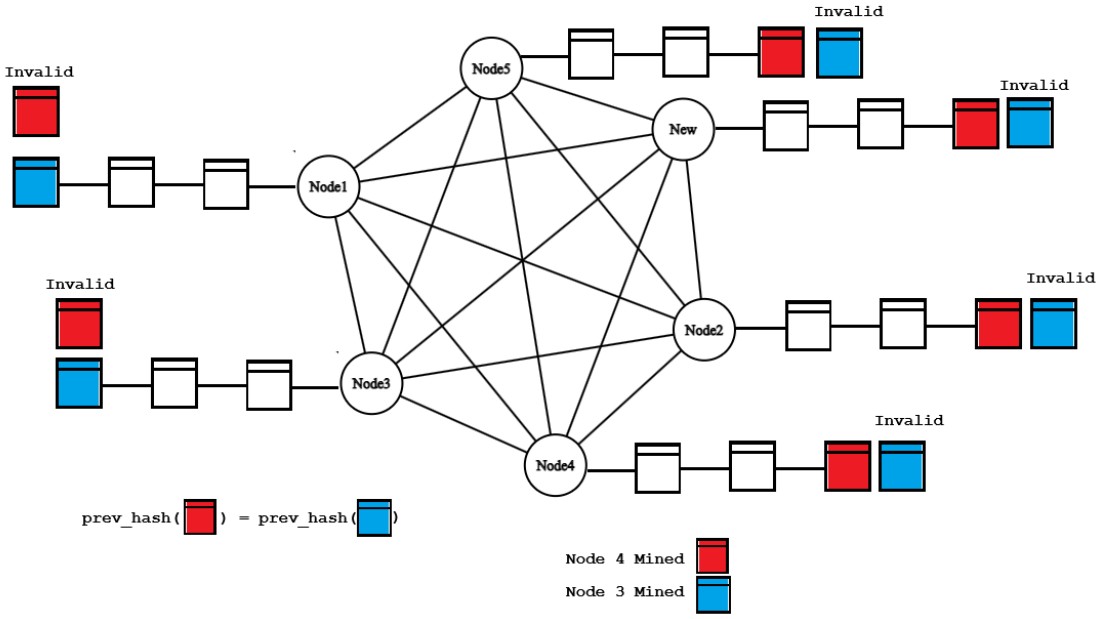



```
Node 4 Mined a block

Node 4 Siarans the Block to (Node
3, Node 1, Node 5, Node2, and
New) concurently
```

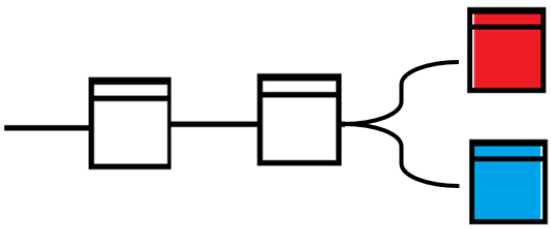
Konsep penjelajahan graf mendasari pengiriman blok dalam jaringan, perhitungan waktu dapat dilakukan dengan mengingat bahwa semakin jauh jarak *node*, maka proses pengiriman akan semakin lama, begitu juga dengan perhitungan berdasarkan latensi jaringan.

5. Bipartite Block Forks

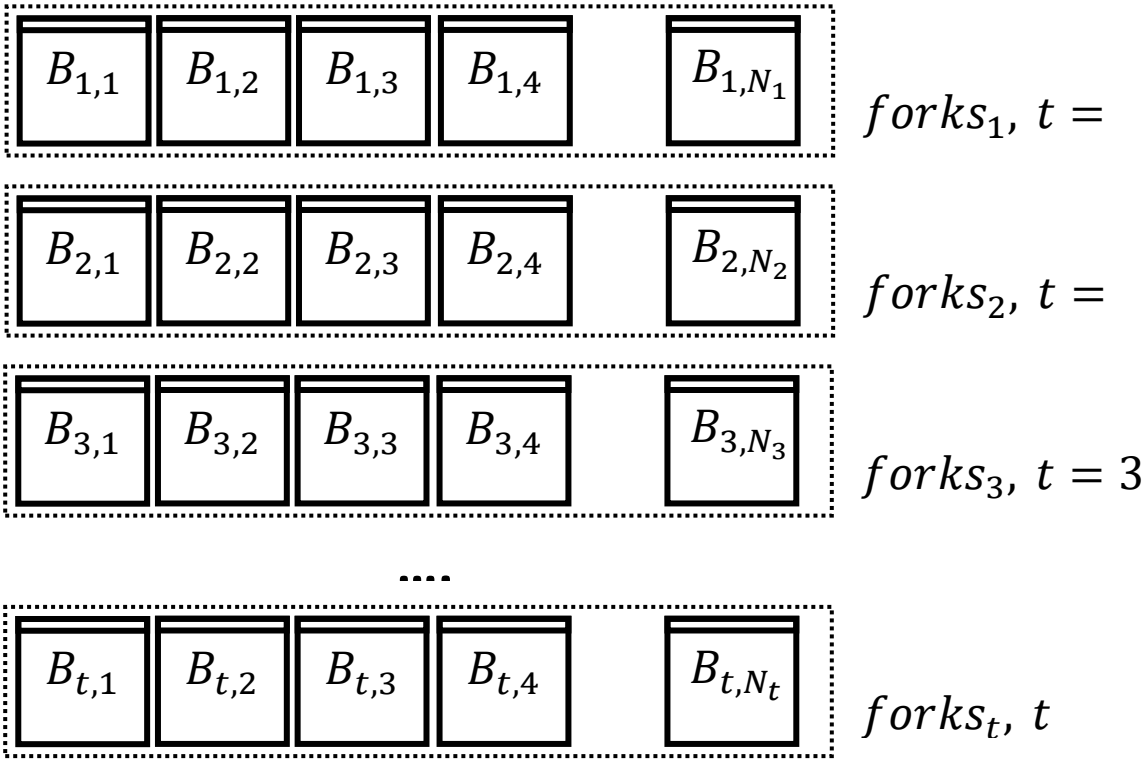
Block Forks adalah kondisi ketika dua atau lebih node menambang blok dalam waktu bersamaan dan blok yang dihasilkan memiliki nilai `prev_hash` yang sama, sehingga nantinya akan terbentuk banyak cabang sebelum blok dimasukkan ke dalam rantai untuk diverifikasi terlebih dahulu. Bab sebelumnya membahas mengenai bagaimana mengatasi masalah ini yaitu dengan melakukan penyortiran blok berdasarkan *proof-of-work* (PoW) terbesar.



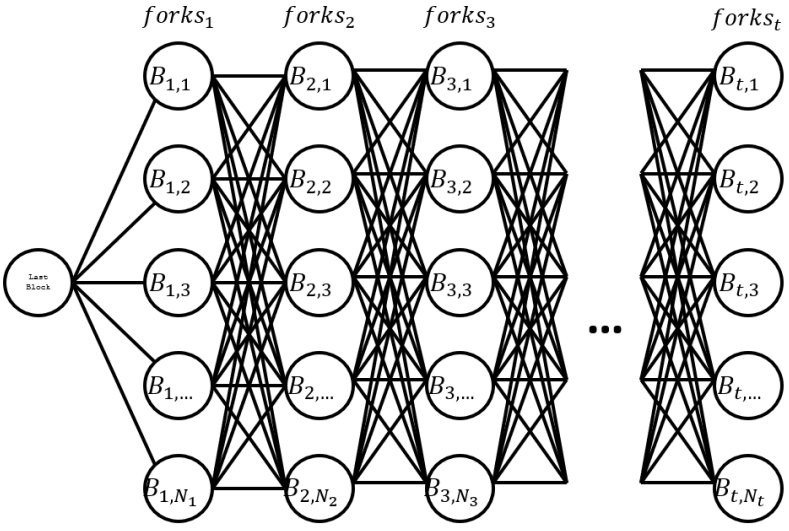
Block yang ditambang kedua node akan disiarkan dalam waktu bersamaan, kemudian setiap Node menerima blok dalam urutan yang berbeda, misalnya dalam ilustrasi di atas node 4 menambang blok merah dan node 3 menambang blok biru. Kedua node secara bersamaan mengirimkan blok dalam waktu yang sama, namun node lainnya seperti node 2, new node, dan node 5 menerima blok merah sebelum blok biru dikarenakan lebih dekat dengan node 4, berbanding terbalik dengan node 3 dan node 1 yang menerima blok merah setelah blok biru bergabung dalam rantai. Percabangan blok terjadi di sini, di mana ada dua jenis rantai yang terbentuk.



Block Forks adalah kondisi multinomial di mana setiap periode waktu tertentu kita misalkan pada rentang waktu tertentu ada N_t blok yang diterima secara bersamaan pada masing – masing waktu ke-t. Kita bisa memisalkan F_t sebagai semua blok yang mengalami forks pada waktu ke-t. Blok – blok pada F_t kita misalkan sebagai $B_{t,i}$ untuk $1 \leq i \leq N_t$



Kita dapat menentukan bahwa permasalahan ini dapat dimodelkan ke dalam sebuah bipartite graph dengan initiate node adalah blok terakhir pada rantai blok sebelum menerima forks dan setiap node pada F_t akan terhubung ke semua anggota F_{t+1} sehingga node $B_{i,t}$ terhubung ke $B_{t+1,j}$ untuk $1 \leq j \leq N_{t+1}$.



Node akan memilih block dengan `proof_of_work` paling besar pada setiap forks yang didapati dan menambahkannya ke rantai. Pencarian blok dapat dilakukan dengan menggunakan transversal Breadth-First-Search (BFS).

```
elif VerifyBlock.BlocksFork:
            LastBlock =  BlockChain.BlockChainData[-1]
            LastBlockProof  = LastBlock["proof_of_work"]
            LastBlockCreator =
ParseSender(LastBlock["message"])
            BlockProof = block.proof_of_work
            BlockCreator = ParseSender(block.message)
            if(LastBlockProof > BlockProof):
                if(BlockCreator ==
DB.GetUserData("PublicKey")):
                    MinedBlock =
BlockChain.mine_block(block)

JSONWorker.CreateDataJSON('Buffer/WaitBox.json',{"Draft"
:MinedBlock.__dict__})
                elif(BlockProof > LastBlockProof):
                    BlockChain.remove_block()
                    BlockChain.add_block(block)
                    if(LastBlockCreator ==
DB.GetUserData("PublicKey")):
                        MinedBlock =
BlockChain.mine_block(LastBlock)

JSONWorker.CreateDataJSON('Buffer/WaitBox.json',{"Draft"
:MinedBlock.__dict__})
```

6. Sinkronisasi Node

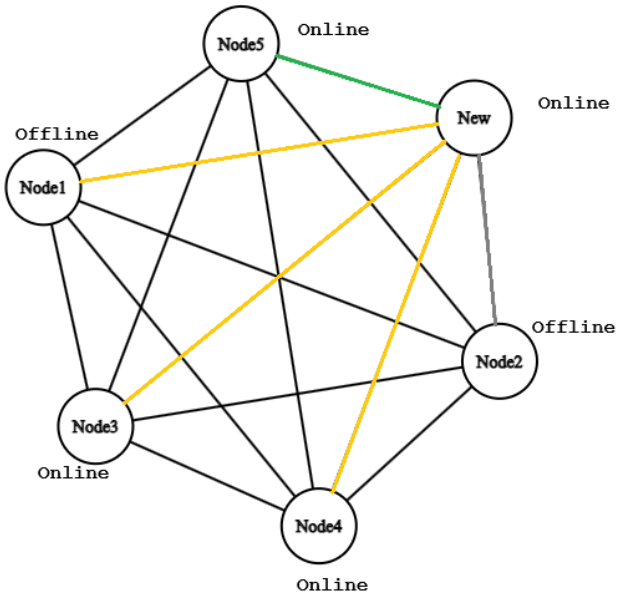
Node dapat mengejar Ketertinggalan blok atau ketidak-sinkronan rantai dengan cara mengunduh blok-chain dari node lainnya, kemudian node yang mendapatkan verifikasi rantai yang gagal juga akan memperbaharui data block-chain lokal dari unduhan tersebut. Pemilihan node yang akan disinkronisasi berdasarkan teknik yang menerapkan konsep ketetanggan dalam graf.

Node akan memilih node lain yang terdekat dengan algoritma *Nearest Neighbour* dalam koneksi berdasarkan pengukuran latensi koneksi.Pemilihan pendekatan ini mempertimbangkan keoptimalan algoritma untuk tipe graf berupa *Complete Graph*. Node juga harus memastikan bahwa tetangga yang dipilih sedang online, temukan node lain dengan latensi minimum selain itu.

Misalkan kita memperoleh informasi pada jaringan bahwa :

```
latency(New,2) < latency(New,5)< latency(New,1) < latency(New,4) < Latency(New,3)
```

Kita akan menemukan bahwa *nearest neighbour* dari node new adalah node 2, namun dikarenakan node 2 sedang offline, node new akan beralih ke node terdekat berikutnya yaitu node 5 untuk melakukan sinkronisasi.

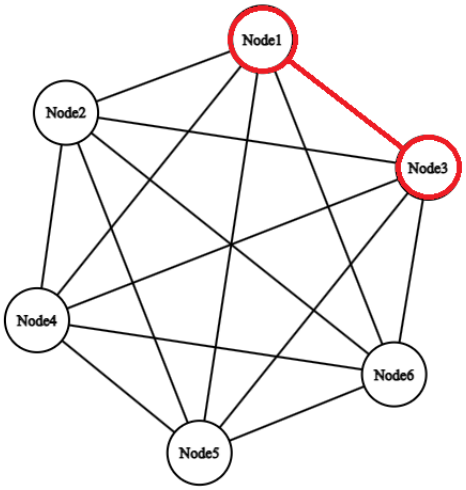


Namun, dalam hal sinkronisasi ditemukan beberapa kerentanan yang akan dibahas pada bagian ketujuh.

7. Byzantine Fault Tolerance (BFT)

Sinkronisasi yang dilakukan oleh suatu node dengan node lainnya berpotensi mengalami kerentanan seperti node lain yang diminta untuk sinkron mencoba untuk mengirimkan data block-chain yang invalid. Permasalahan ini dapat diatasi dengan beberapa solusi.

Ilustrasi di bawah ini akan menunjukkan, misalkan node 1 menerima block-chain yang invalid dari node 3, sehingga block-chain lokal milik node 1 menjadi invalid

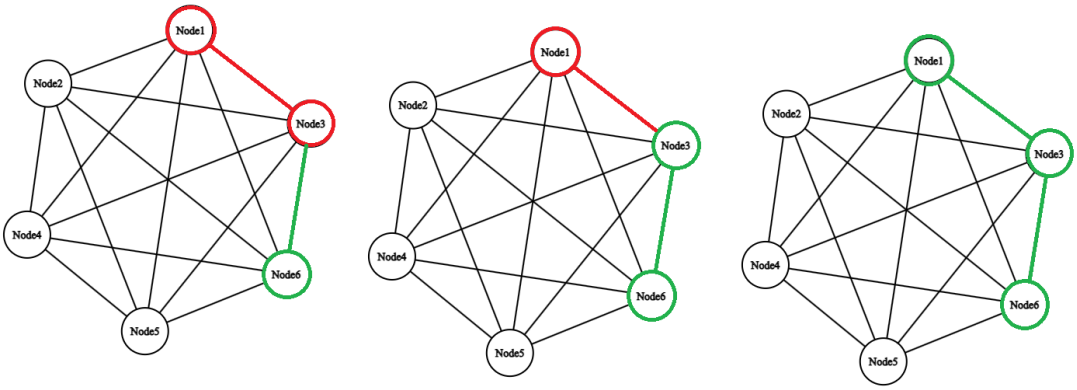


7.1. Naïve Nearest Neighbour (unmutual neighbour)

Permasalahan dapat teratasi dengan berharap secara naif yaitu saat node 3 menerima siaran blok, menambang blok baru, atau offline kemudian online kembali lalu mendapatkan sinkronisasi dari *nearest neighbour* yang terpercaya dengan berlaku asumsi *unmutual neighbour* :

$$nearest_{neighbour(node\ u)} = node\ v$$
$$nearest_{neighbour(node\ v)} \neq node\ u$$

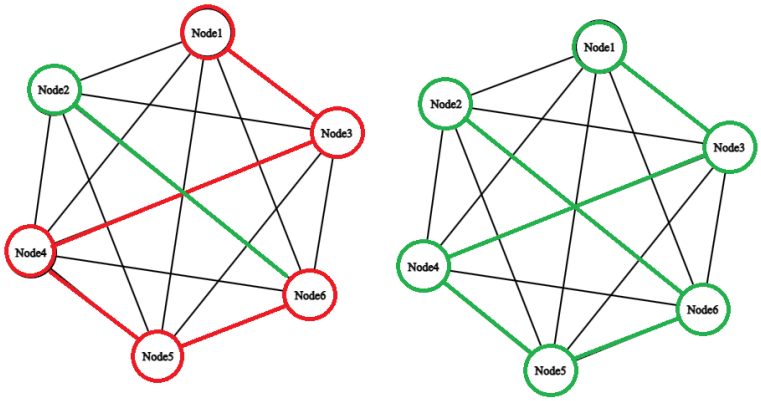
Kita juga bisa menkonstruksi agar Node 3 melakukan verifikasi data blockchainnya terlebih dahulu kemudian sinkronisasi sebelum mengirimkannya ke



node 1, sehingga dari sini ode 3 akan mendapatkan block-chain yang valid, sehingga saat node 1 meminta sinkronisasi dari node 3 ia juga akan menerima block-chain yang valid.

```
async def HandlerGetBlockChain(websocket, path):
        BlockChainSync.VerifySyncBlockChain()
        BlockChainData = DB.GetBlockChainData()
        UserData = DB.GetUserData("PublicKey")
        await websocket.send(json.dumps({
            "WalletAuthor":UserData,
            "BlockChainData" : BlockChainData}))
```

Kasus yang lebih besar seperti semua node memberikan sinkronisasi yang invalid , namun minimal ada satu node yang memberikan block-chain valid maka penjelajahan graf akan dilakukan.

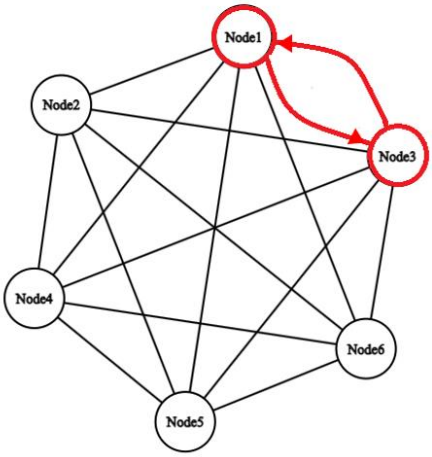


7.2. Depth First Search - Backtracking

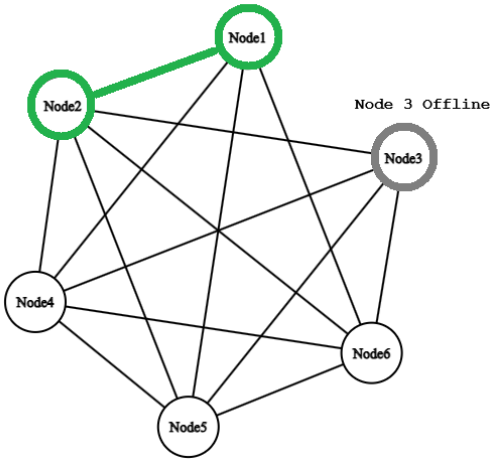
Solusi dengan menggunakan *Nearest Neighbour* merupakan solusi yang naif dikarenakan berpotensi adanya kasus *mutual neighbour* di mana :

$$nearest_{neighbour(node\ u)} = node\ v$$
$$nearest_{neighbour(node\ v)} = node\ u$$

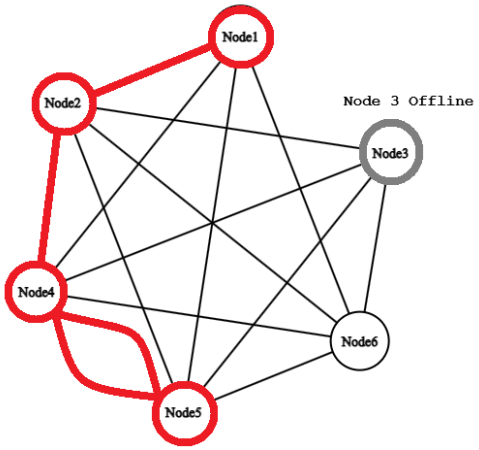
Jika hal ini terjadi maka penjelajahan graf hanya akan terjadi *infinite loop* dua node yang mutual saja.

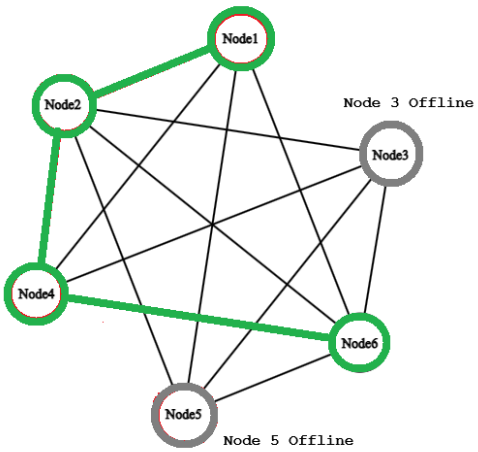


Solusi naif lainnya untuk masalah ini adalah kita berharap salah satu node *offline* sehingga node akan melakukan *backtracking* kemudian meminta sinkronisasi ke *nearest neighbour* lain yang valid.

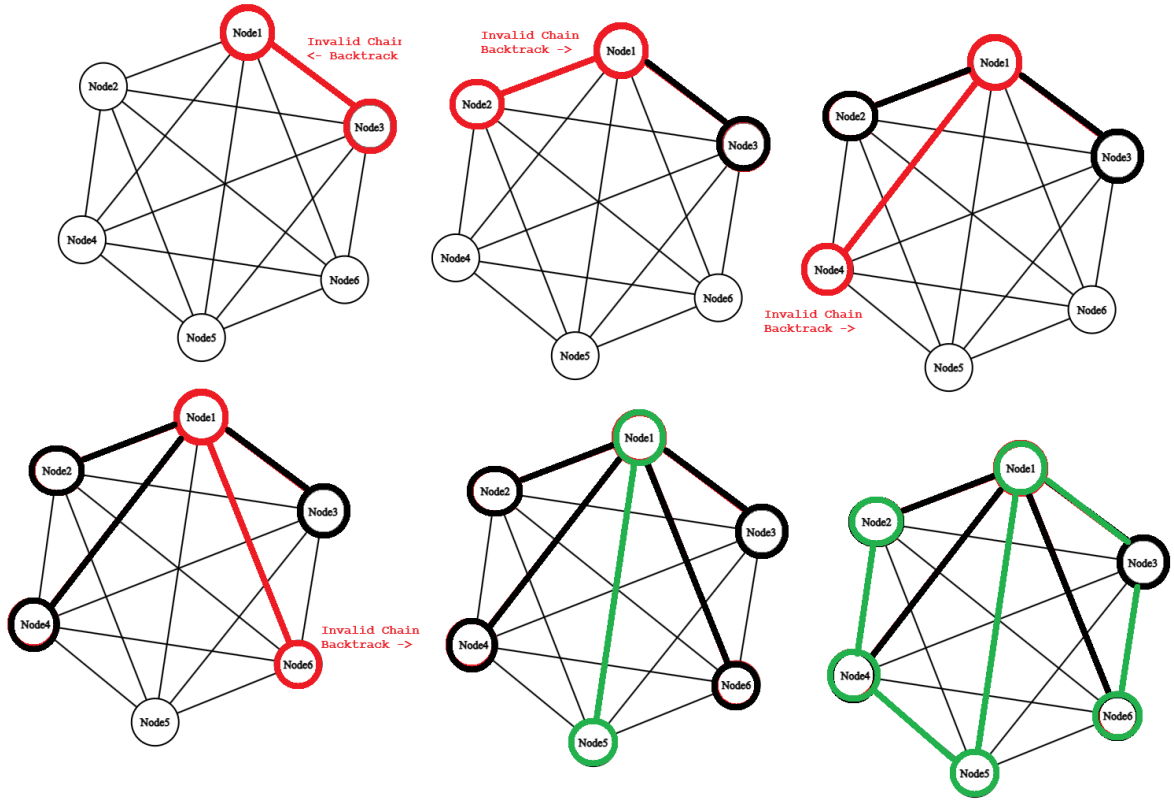


Berlaku juga untuk kasus yang lebih besar.





Kita mempertimbangkan kasus lain, walaupun kemungkinannya sangat kecil yakni di mana ketika terjadi loop, *mutual node* tidak akan pernah offline. Solusi ***O(n)*** untuk permasalahan ini adalah saat sinkronisasi dilakukan, jangan lupa untuk melakukan verifikasi blockchain yang diterima, jika sinkronisasi invalid maka *backtrack* dan minta sinkronisasi ke node yang lain.



```
def SynchronizeBlockChain():
    BlockChain = Blockchain()
    data  = DB.GetNetworkData()
    nodes = data["Nodes"]
    latency_nodes = {k: v for k, v in nodes.items() if
"Latency" in v}
    sorted_nodes = sorted(latency_nodes.items(), key=lambda
item: item[1]["Latency"])
    nodeIndex = 0
    while True:
        if latency_nodes:
            closest_node = sorted_nodes[nodeIndex]
            closest_ip = closest_node[0]
            ReceivedChain =
WSBlockChainSync.SynchronizeHandler(closest_ip)
            if(ReceivedChain["BlockChainData"] != '' and
ReceivedChain):

if(BlockChain.verify_chain(ReceivedChain["BlockChainData"])):

JSONWorker.CreateDataJSON('Database/BlockChainDB.json',Receiv
edChain["BlockChainData"])
                    break
                else:
                    nodeIndex+=1
                    continue
        else:
            print("No nodes with latency information found.")
```