Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India (Autonomous College Affiliated to University of Mumbai)

Keyur Pancholi (2020300047)
Jai Patel (2020300052)
Aaryan Purohit (2020300055)

Aim:

Implementation of clock synchronization

THEORY

Clock Synchronization

Distributed System is a collection of computers connected via the high speed communication network. In the distributed system, the hardware and software components communicate and coordinate their actions by message passing. Each node in distributed systems can share their resources with other nodes. So, there is need of proper allocation of resources to preserve the state of resources and help coordinate between the several processes. To resolve such conflicts, synchronization is used. Synchronization in distributed systems is achieved via clocks.

The physical clocks are used to adjust the time of nodes. Each node in the system can share its local time with other nodes in the system. The time is set based on UTC (Universal Time Coordination). UTC is used as a reference time clock for the nodes in the system.

The clock synchronization can be achieved by 2 ways: External and Internal Clock Synchronization.

1. **External clock synchronization** is the one in which an external reference clock is present. It is used as a reference and the nodes in the system can set and adjust their time accordingly.
2. **Internal clock synchronization** is the one in which each node shares its time with other nodes and all the nodes set and adjust their times accordingly.

There are 2 types of clock synchronization algorithms: Centralized and Distributed.

1. **Centralized** is the one in which a time server is used as a reference. The single time server propagates its time to the nodes and all the nodes adjust the time accordingly. It is dependent on single time server so if that node fails, the whole system will lose synchronization. Examples of centralized are- Berkeley Algorithm, Passive Time Server, Active Time Server etc.
2. **Distributed** is the one in which there is no centralized time server present. Instead the nodes adjust their time by using their local time and then, taking the average of the differences of time with other nodes. Distributed algorithms overcome the issue of centralized algorithms like the scalability and single point failure. Examples of Distributed algorithms are – Global Averaging Algorithm, Localized Averaging Algorithm, NTP (Network time protocol) etc.

Berkeley’s Algorithm

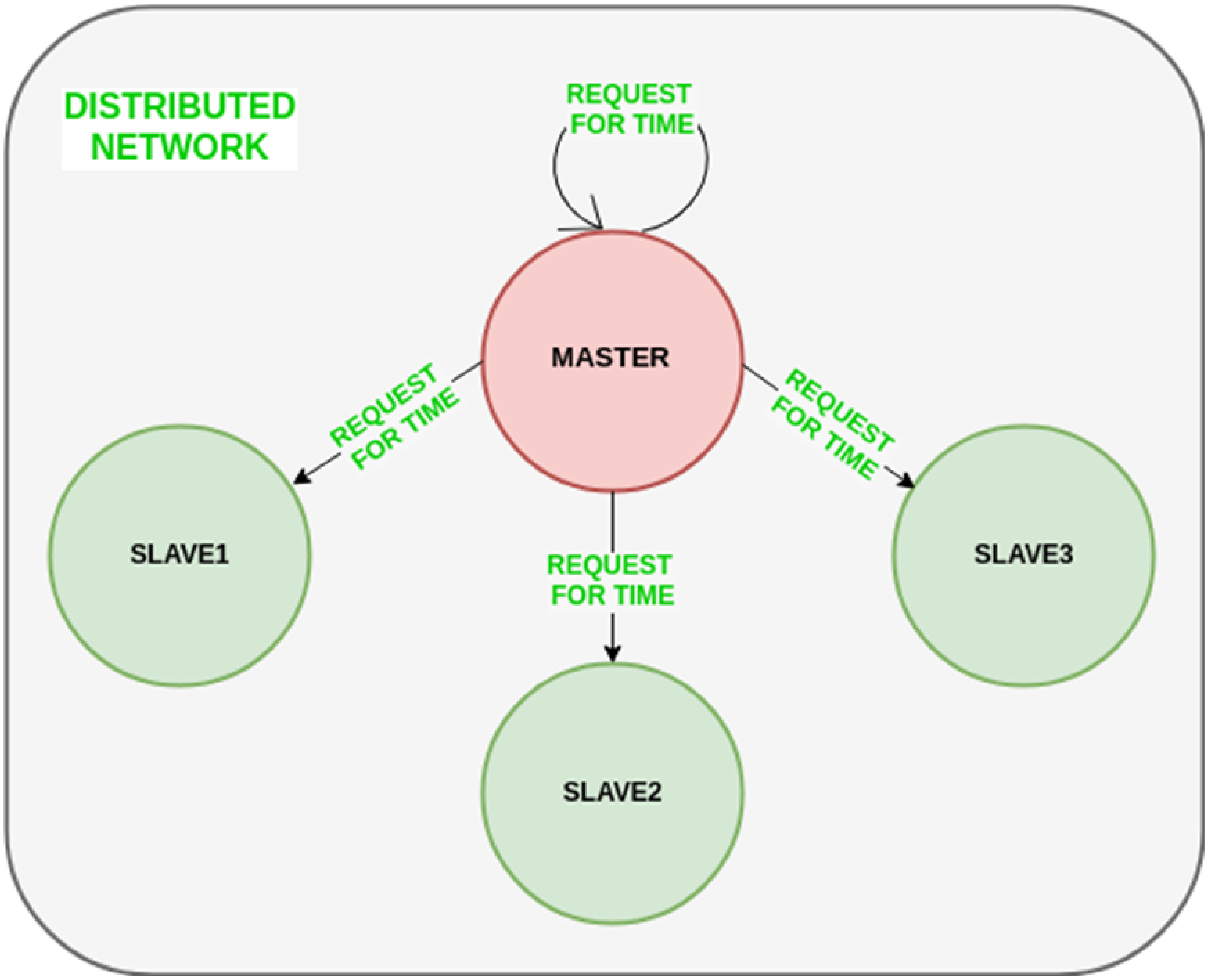
Berkeley’s Algorithm is a clock synchronization technique used in distributed systems. The algorithm assumes that each machine node in the network either doesn’t have an accurate time source or doesn’t possess a UTC server.

Algorithm

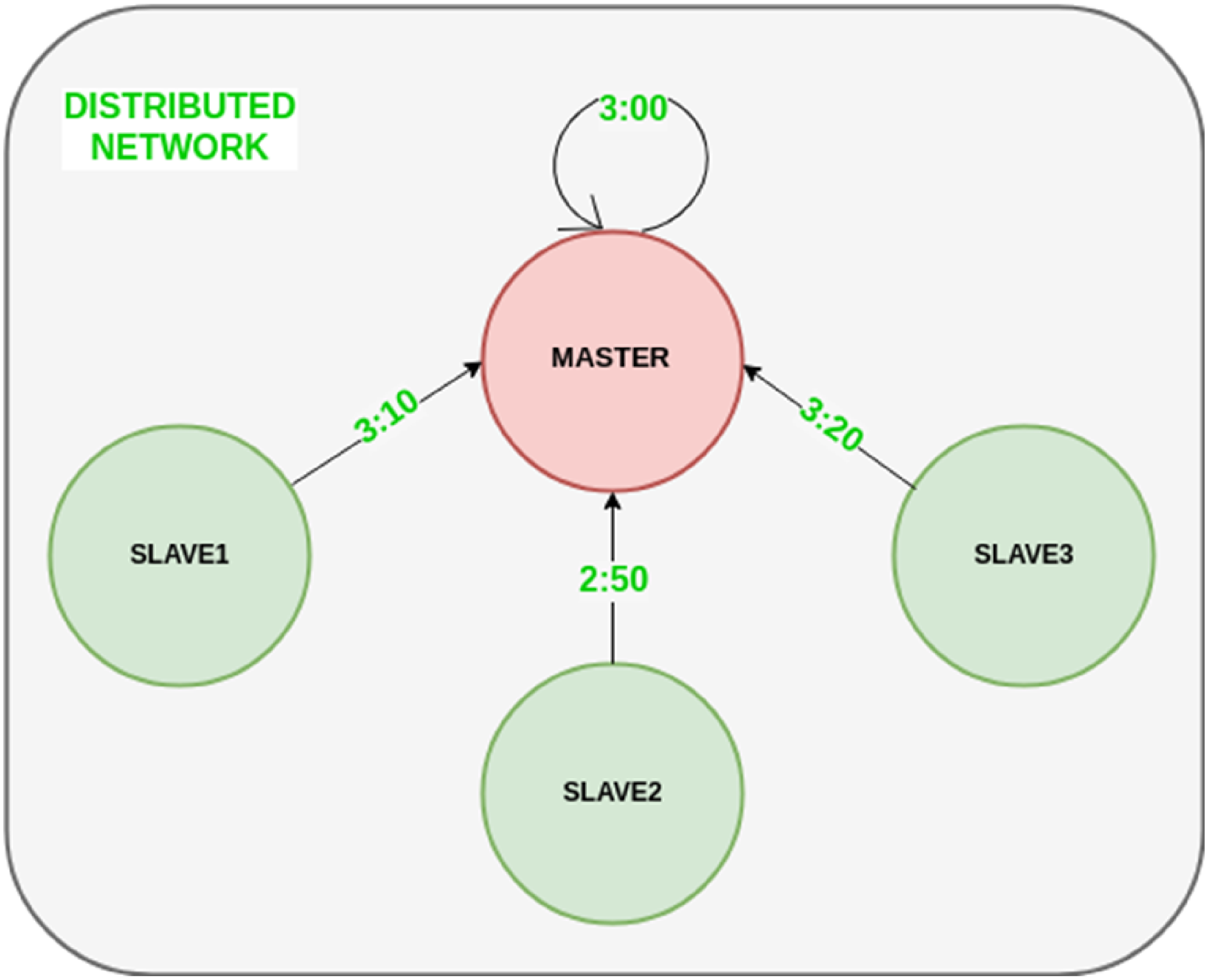
1) An individual node is chosen as the master node from a pool node in the network. This node is the main node in the network which acts as a master and the rest of the nodes act as slaves. The master node is chosen using an election process/leader election algorithm.

2) Master node periodically pings slaves nodes and fetches clock time at them using Cristian’s algorithm.

The diagram below illustrates how the master sends requests to slave nodes.

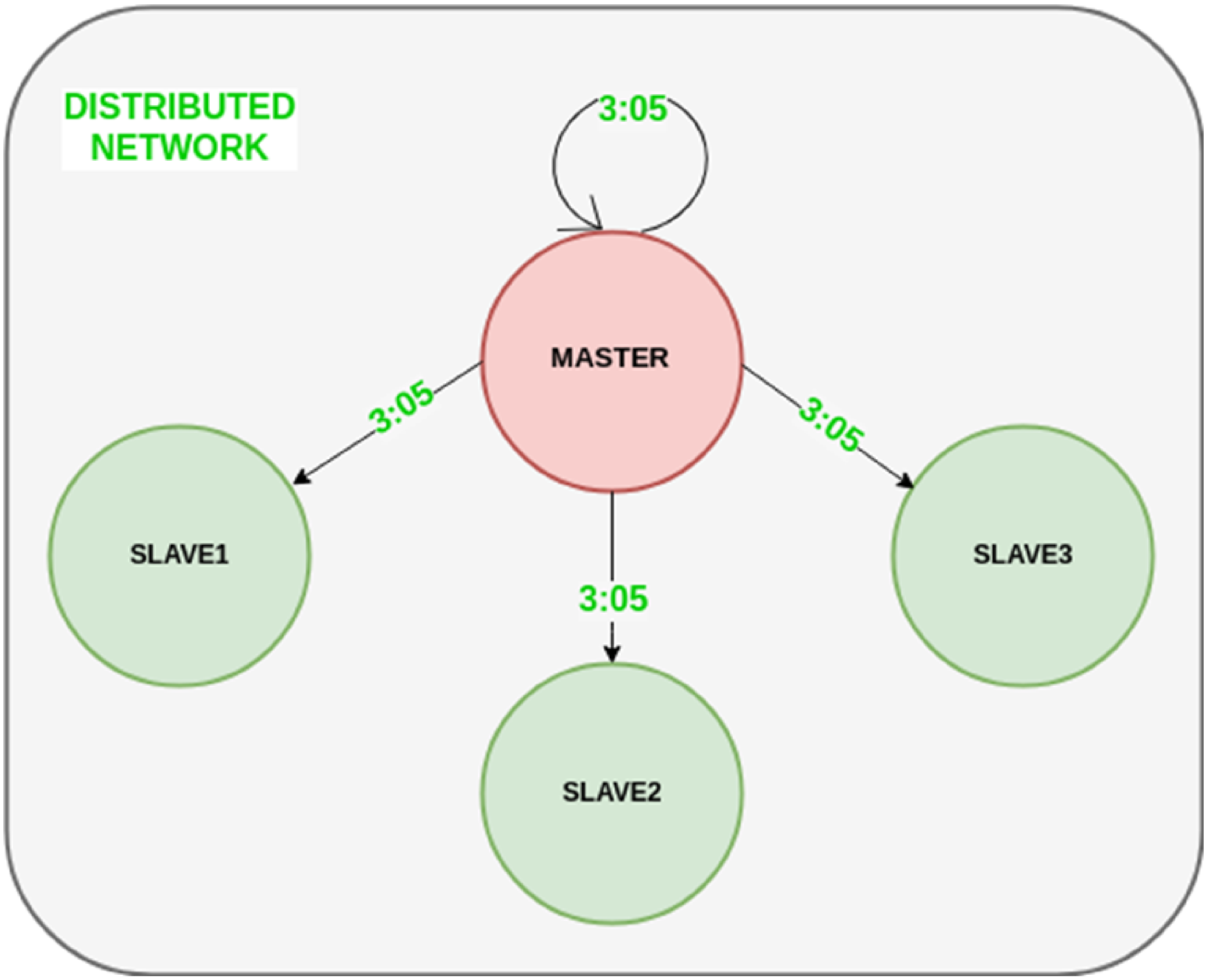


The diagram below illustrates how slave nodes send back time given by their system clock.



3) Master node calculates the average time difference between all the clock times received and the clock time given by the master's system clock itself. This average time difference is added to the current time at the master's system clock and broadcasted over the network.

The diagram below illustrates the last step of Berkeley's algorithm.



Scope of Improvement

- Improvision inaccuracy of Cristian's algorithm.
- Ignoring significant outliers in the calculation of average time difference

- In case the master node fails/corrupts, a secondary leader must be ready/pre-chosen to take the place of the master node to reduce downtime caused due to the master's unavailability.
- Instead of sending the synchronized time, master broadcasts relative inverse time difference, which leads to a decrease in latency induced by traversal time in the network while the time of calculation at slave node.

Code

Client side

```
from multiprocessing.sharedctypes import Synchronized
from timeit import default_timer as timer
from dateutil import parser
import threading
import datetime
import socket
import json
import time

from server import print_messages
name = ''

# client thread function used to send time at client side


def startSendingTime(slave_client):

    while True:
        # provide server with clock time at the client
        data = {
            'type': 'time',
            'time': str(
                datetime.datetime.now())
        }
        payload = json.dumps(data)
```

```python
        slave_client.send(payload.encode())

        print("Recent time sent successfully",
              end="\n\n")

        time.sleep(5)


# client thread function used to receive synchronized time
def startReceivingTime(slave_client):

    while True:
        # receive data from the server
        # Synchronized_time = parser.parse(
        #     slave_client.recv(1024).decode())
        payload = slave_client.recv(1024).decode()
        payload = json.loads(payload)  # data loaded
        if payload.get('type') != 'time':
            print_messages(payload)
            continue
        Synchronized_time = parser.parse(payload['time'])

        print("Synchronized time at the client is: " +
              str(Synchronized_time),
              end="\n\n")


def sendMessage(slave_client):
    while True:
        # print("Send Message", end='\n')
        message = input()
        data = {
            'type': 'message',
            'message': message,
            'name': name
        }
        payload = json.dumps(data)
        slave_client.send(payload.encode())
```

```python
def print_messages(data):
    global name
    if data['type'] == 'connect':
        name = data['name']
        print(f"{name} connected")
    if data['type'] == 'message':
        _name = data['name']
        message = data['message']
        print(f"{_name}: {message}")


# function used to Synchronize client process time


def initiateSlaveClient(port=8080):

    global name
    slave_client = socket.socket()
    name = input("Enter name => ")

    # connect to the clock server on local computer
    slave_client.connect(('127.0.0.1', port))
    data = {
        'type': 'connect',
        'name': name
    }
    data = json.dumps(data)
    slave_client.send(data.encode())
    print("Send message =>", end='\r')

    # start sending time to server
    print("Starting to receive time from server\n")
    send_time_thread = threading.Thread(
        target=startSendingTime,
        args=(slave_client, ))
    send_time_thread.start()

    # start receiving synchronized from server
    print("Starting to receiving " +
          "synchronized time from server\n")
```

```
    receive_time_thread = threading.Thread(
        target=startReceivingTime,
        args=(slave_client, ))

    send_messages = threading.Thread(
        target=sendMessage,
        args=(slave_client, ))
    receive_time_thread.start()
    send_messages.start()


# Driver function
if __name__ == '__main__':

    # initialize the Slave / Client
    initiateSlaveClient(port=8080)


from multiprocessing.sharedctypes import Synchronized
from timeit import default_timer as timer
from dateutil import parser
import threading
import datetime
import socket
import json
import time

from server import print_messages
name = ''

# client thread function used to send time at client side


def startSendingTime(slave_client):

    while True:
        # provide server with clock time at the client
        data = {
            'type': 'time',
            'time': str(
```

```python
                datetime.datetime.now())
        }
        payload = json.dumps(data)
        slave_client.send(payload.encode())

        print("Recent time sent successfully",
              end="\n\n")

        time.sleep(5)


# client thread function used to receive synchronized time
def startReceivingTime(slave_client):

    while True:
        # receive data from the server
        # Synchronized_time = parser.parse(
        #     slave_client.recv(1024).decode())
        payload = slave_client.recv(1024).decode()
        payload = json.loads(payload)  # data loaded
        if payload.get('type') != 'time':
            print_messages(payload)
            continue
        Synchronized_time = parser.parse(payload['time'])

        print("Synchronized time at the client is: " +
              str(Synchronized_time),
              end="\n\n")


def sendMessage(slave_client):
    while True:
        # print("Send Message", end='\n')
        message = input()
        data = {
            'type': 'message',
            'message': message,
            'name': name
        }
        payload = json.dumps(data)
```

```python
        slave_client.send(payload.encode())


def print_messages(data):
    global name
    if data['type'] == 'connect':
        name = data['name']
        print(f"{name} connected")
    if data['type'] == 'message':
        _name = data['name']
        message = data['message']
        print(f"{_name}: {message}")


# function used to Synchronize client process time


def initiateSlaveClient(port=8080):

    global name
    slave_client = socket.socket()
    name = input("Enter name => ")

    # connect to the clock server on local computer
    slave_client.connect(('127.0.0.1', port))
    data = {
        'type': 'connect',
        'name': name
    }
    data = json.dumps(data)
    slave_client.send(data.encode())
    print("Send message =>", end='\r')

    # start sending time to server
    print("Starting to receive time from server\n")
    send_time_thread = threading.Thread(
        target=startSendingTime,
        args=(slave_client, ))
    send_time_thread.start()
```

```
    # start receiving synchronized from server
    print("Starting to receiving " +
          "synchronized time from server\n")
    receive_time_thread = threading.Thread(
        target=startReceivingTime,
        args=(slave_client, ))

    send_messages = threading.Thread(
        target=sendMessage,
        args=(slave_client, ))
    receive_time_thread.start()
    send_messages.start()


# Driver function
if __name__ == '__main__':

    # initialize the Slave / Client
    initiateSlaveClient(port=8080)
```

Server Side

```
from concurrent.futures import thread
from email import message
from dateutil import parser
import threading
import datetime
import socket
import json
import time


client_data = {}


''' nested thread function used to receive
    clock time from a connected client '''
```

```python
def startReceivingClockTime(connector, address, data):

    # while True:
    # receive clock time
    # data = connector.recv(1024).decode()
    # data = json.loads(data)
    # print("type=", data)

    if data['type'] != 'time':
        print_messages(data)
        return

    print("time recieved")
    clock_time = parser.parse(data['time'])
    clock_time_diff = datetime.datetime.now() - \
        clock_time

    client_data[address] = {
        "clock_time": clock_time,
        "time_difference": clock_time_diff,
        "connector": connector
    }

    print("Client Data updated with: " + str(address),
          end="\n\n")
    # time.sleep(5)


''' master thread function used to open portal for
    accepting clients over given port '''


def startConnecting(master_server):

    # fetch clock time at slaves / clients
    while True:
        # accepting a client / slave clock client
        master_slave_connector, addr = master_server.accept()
```

```
        slave_address = str(addr[0]) + ":" + str(addr[1])

        print(slave_address + " got connected successfully")

        # current_thread = threading.Thread(
        #     target=startReceivingClockTime,
        #     args=(master_slave_connector,
        #            slave_address, ))

        receive_messages = threading.Thread(target=receiveData,
                                            args=(master_slave_connector,
                                                  slave_address, ))
        receive_messages.start()
        data = {
            'type': 'time',
            'time': str(
                datetime.datetime.now())
        }
        startReceivingClockTime(master_slave_connector, slave_address,
data)

        # current_thread.start()

# subroutine function used to fetch average clock difference


def getAverageClockDiff():

    current_client_data = client_data.copy()

    time_difference_list = list(client['time_difference']
                                for client_addr, client
                                in client_data.items())

    sum_of_clock_difference = sum(time_difference_list,
                                  datetime.timedelta(0, 0))

    average_clock_difference = sum_of_clock_difference \
        / len(client_data)
```

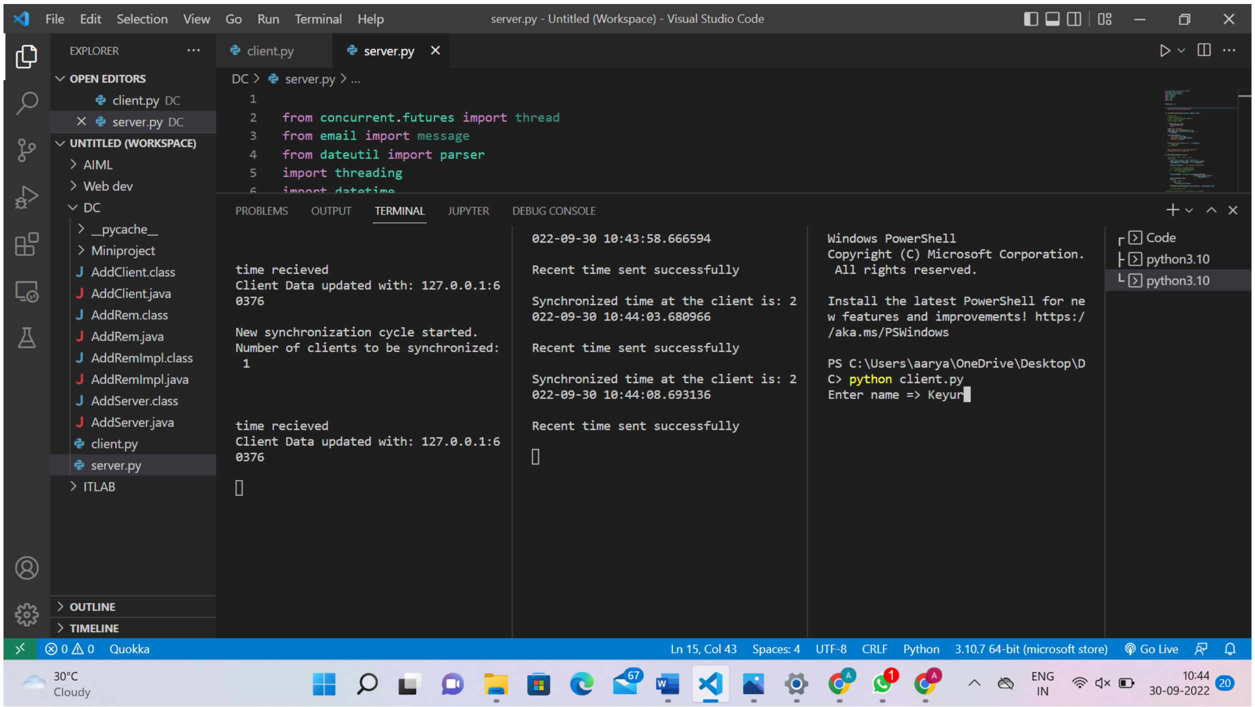
```
    return average_clock_difference


''' master sync thread function used to generate
    cycles of clock synchronization in the network '''


def synchronizeAllClocks():

    while True:

        print("New synchronization cycle started.")
        print("Number of clients to be synchronized: " +
              str(len(client_data)))

        if len(client_data) > 0:

            average_clock_difference = getAverageClockDiff()

            for client_addr, client in client_data.items():
                try:
                    synchronized_time = \
                        datetime.datetime.now() + \
                        average_clock_difference

                    data = {
                        'type': 'time',
                        'time': str(synchronized_time)
                    }
                    payload = json.dumps(data)

                    client['connector'].send(payload.encode())
                    # client['connector'].send(str(
                    #     synchronized_time).encode())

                except Exception as e:
                    print(e)
                    print("Something went wrong while " +
                          "sending synchronized time " +
                          "through " + str(client_addr))
```

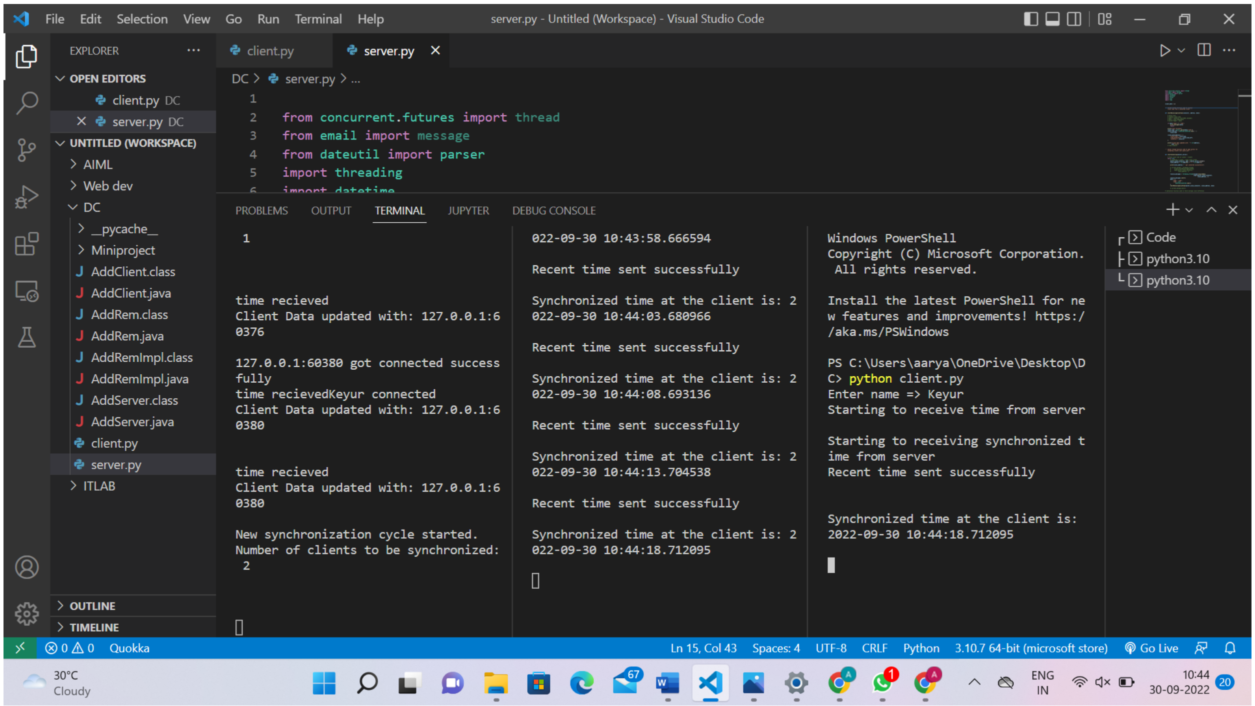
```python
        else:
            print("No client data." +
                  " Synchronization not applicable.")

        print("\n\n")

        time.sleep(5)


def print_messages(data):
    if data['type'] == 'connect':
        name = data['name']
        print(f"{name} connected")
    if data['type'] == 'message':
        _name = data['name']
        message = data['message']

        # print(f"{_name}: {message}")
        payload = {
            'type': 'message',
            'message': message,
            'name': _name
        }
        for client_addr, client in client_data.items():
            data = json.dumps(payload)
            client['connector'].send(data.encode())


# function used to initiate the Clock Server / Master Node

def receiveData(connector, slave_address):
    while True:
        data = connector.recv(1024).decode()
        data = json.loads(data)
        if data['type'] == 'time':
            startReceivingClockTime(connector, slave_address, data)
        else:
            print_messages(data)
```

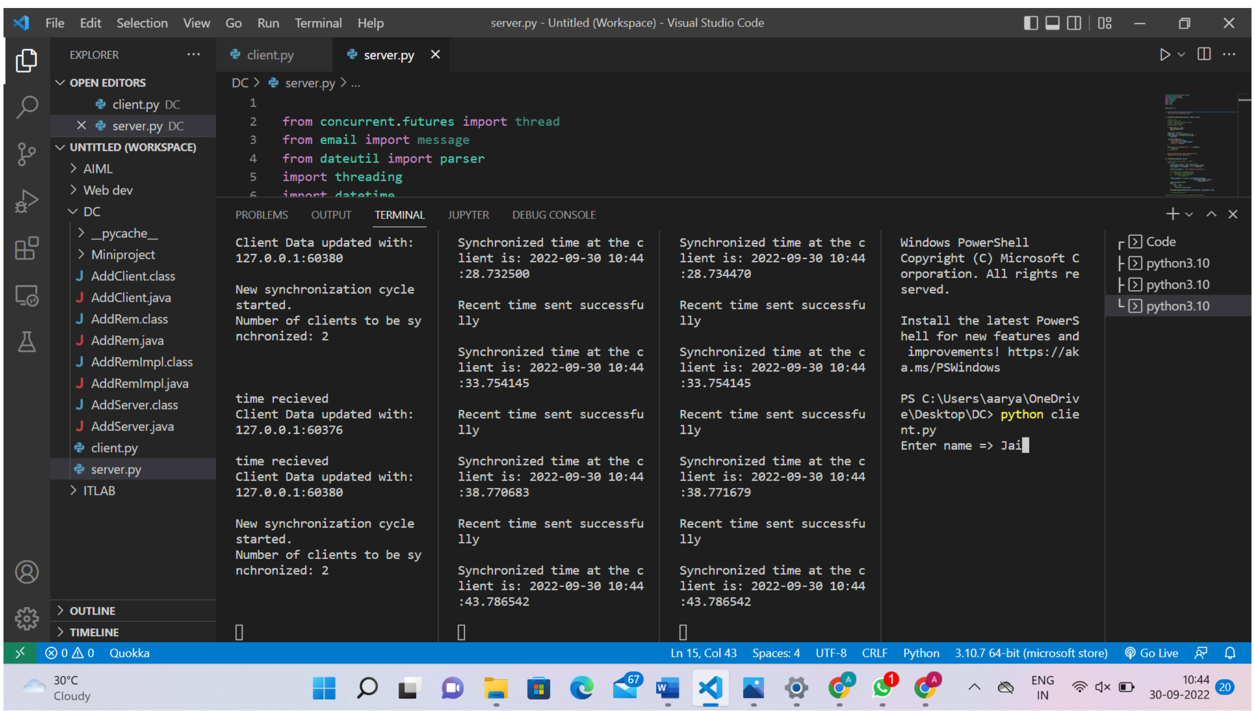
```python
def initiateClockServer(port=8080):

    master_server = socket.socket()
    master_server.setsockopt(socket.SOL_SOCKET,
                             socket.SO_REUSEADDR, 1)

    print("Socket at master node created successfully\n")

    master_server.bind(('', port))

    # Start listening to requests
    master_server.listen(10)
    print("Clock server started...\n")

    # start making connections
    print("Starting to make connections...\n")
    master_thread = threading.Thread(
        target=startConnecting,
        args=(master_server, ))
    master_thread.start()

    # start synchronization
    print("Starting synchronization parallelly...\n")
    sync_thread = threading.Thread(
        target=synchronizeAllClocks,
        args=())
    sync_thread.daemon = True
    sync_thread.start()

    receive_messages = threading.Thread(
        target=receiveData,
        args=(master_server))
    receive_messages.start()


# Driver function
if __name__ == '__main__':

    # Trigger the Clock Server
```

```
initiateClockServer(port=8080)
```

Output




server.py - Untitled (Workspace) - Visual Studio Code
from concurrent.futures import thread
from email import message
from dateutil import parser
import threading
PROBLEMS OUTPUT TERMINAL JUPYTER DEBUG CONSOLE
time recieved
Client Data updated with: 127.0.0.1:60376
127.0.0.1:60380 got connected successfully
time recievedKeyur connected
Client Data updated with: 127.0.0.1:60380
time recieved
Client Data updated with: 127.0.0.1:60380
New synchronization cycle started.
Number of clients to be synchronized: 2
022-09-30 10:43:58.666594
Recent time sent successfully
Synchronized time at the client is: 2022-09-30 10:44:03.680966
Recent time sent successfully
Synchronized time at the client is: 2022-09-30 10:44:08.693136
Recent time sent successfully
Synchronized time at the client is: 2022-09-30 10:44:13.704538
Recent time sent successfully
Synchronized time at the client is: 2022-09-30 10:44:18.712095
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS C:\Users\aarya\OneDrive\Desktop\DC> python client.py
Enter name => Keyur
Starting to receive time from server
Starting to receiving synchronized time from server
Recent time sent successfully
Synchronized time at the client is: 2022-09-30 10:44:18.712095


server.py - Untitled (Workspace) - Visual Studio Code
from concurrent.futures import thread
from email import message
from dateutil import parser
import threading
PROBLEMS OUTPUT TERMINAL JUPYTER DEBUG CONSOLE
Client Data updated with: 127.0.0.1:60380
New synchronization cycle started.
Number of clients to be synchronized: 2
time recieved
Client Data updated with: 127.0.0.1:60376
time recieved
Client Data updated with: 127.0.0.1:60380
New synchronization cycle started.
Number of clients to be synchronized: 2
Synchronized time at the client is: 2022-09-30 10:44:28.732500
Recent time sent successfully
Synchronized time at the client is: 2022-09-30 10:44:33.754145
Recent time sent successfully
Synchronized time at the client is: 2022-09-30 10:44:38.770683
Recent time sent successfully
Synchronized time at the client is: 2022-09-30 10:44:43.786542
Synchronized time at the client is: 2022-09-30 10:44:28.734470
Synchronized time at the client is: 2022-09-30 10:44:33.754145
Synchronized time at the client is: 2022-09-30 10:44:38.771679
Synchronized time at the client is: 2022-09-30 10:44:43.786542
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS C:\Users\aarya\OneDrive\Desktop\DC> python client.py
Enter name => Jai