```python
#!/usr/bin/python
import time
import asyncio
import aiohttp
import json
import sys
import re
import logging
import decimal

def get_port_num(server_name):
    return {
        'Goloman' : 11989,
        'Hands' : 11990,
        'Holiday' : 11991,
        'Wilkes' : 11992,
        'Welsh' : 11993
    } [server_name]

def talks_with(server_name):
    return {
        'Goloman' : ['Hands', 'Holiday', 'Wilkes'],
        'Hands' : ['Goloman', 'Wilkes'],
        'Holiday' : ['Goloman', 'Welsh', 'Wilkes'],
        'Wilkes': ['Goloman', 'Hands', 'Holiday'],
        'Welsh' : ['Holiday']
    } [server_name]

def handle_latlon(lat_lon_str):
    if(lat_lon_str[0] == "+"):
        lat_lon = lat_lon_str.strip("+")
        pos_split = lat_lon.split("+")
        neg_split = lat_lon.split("-")
        if(len(pos_split) == 2):
            lat = decimal.Decimal(pos_split[0])
            lon = decimal.Decimal(pos_split[1])
        elif(len(neg_split) == 2):
            lat = decimal.Decimal(neg_split[0])
            lon = decimal.Decimal("-{0}".format(neg_split[1]))
        else:
            raise ValueError("invalid input")
    elif(lat_lon_str[0] == "-"):
        lat_lon = lat_lon_str.strip("-")
        pos_split = lat_lon.split("+")
        neg_split = lat_lon.split("-")
        if(len(pos_split) == 2):
            lat = decimal.Decimal("-{0}".format(pos_split[0]))
            lon = decimal.Decimal(pos_split[1])
        elif(len(neg_split) == 2):
            lat = decimal.Decimal("-{0}".format(neg_split[0]))
            lon = decimal.Decimal("-{0}".format(neg_split[1]))
        else:
            raise ValueError("invalid input")
    else:
        raise ValueError("invalid input")
    return lat, lon

class server_class:
    def __init__(self, server_name):
        self.name = server_name
        self.loop = asyncio.get_event_loop()
        self.connected_servers = []
        self.user_data = {} # user_name : [server,skew,lat,lon,time]
        logging.basicConfig(filename='{0}.log'.format(self.name), level=logging.INFO, format='%(levelname)s - %(asctime)s
- %(message)s')
        logging.info("Started {0}".format(self.name))

        server_port = get_port_num(self.name)
        routine = asyncio.start_server(self.server_routine,'127.0.0.1', server_port, loop=self.loop)
        self.server = self.loop.run_until_complete(routine)
        try:
            self.loop.run_forever()
        except KeyboardInterrupt:
            pass

    def stop_server(self):
        self.server.close()
        self.loop.run_until_complete(self.server.wait_closed())
        self.loop.close()

    async def propagate_message(self, message):
        composition = message.split(" ")
        message_to_friends = "{0} {1}\n".format(message, self.name)
        propagation_tasks = []
        if len(composition) > 6:
            up_stream = composition[6:]
            propagation_tasks.append(asyncio.ensure_future(self.maintain_connections(up_stream)))
        else:
            up_stream = []
        for friend in talks_with(self.name):
            friend_port = get_port_num(friend)
            if friend not in up_stream:
```

```
                propagation_tasks.append(asyncio.ensure_future(self.cant_stop_the_signal(friend, friend_port, 
message_to_friends)))
        await asyncio.gather(*propagation_tasks)
        logging.info("Message propagated to available servers.")

    async def cant_stop_the_signal(self, friend, friend_port, message_to_friends):
        persistent_connection = False
        for connected in self.connected_servers:
            if friend_port == connected[0]:
                try:
                    connected[2].write(message_to_friends.encode())
                    await connected[2].drain()
                    logging.info("Recipient: %s Output: %r" % (friend, message_to_friends))
                    persistent_connection = True
                except:
                    self.connected_servers.remove(connected)
                    logging.info("Lost Connection To {0}".format(friend))
                    persistent_connection = False
        if not persistent_connection:
            try:
                self.connected_servers.append(await self.connection_routine(friend_port, message_to_friends))
                logging.info("Connected To {0}".format(friend))
                logging.info("Recipient: %s Output: %r" % (friend, message_to_friends))
            except:
                pass

    async def attempt_connection(self, ports, port_num):
        try:
            reader, writer = await asyncio.open_connection('127.0.0.1', port_num, loop=self.loop)
            self.connected_servers.append(port_num, reader, writer)
            logging.info("Connected To {0}".format(ports[port_num]))
        except:
            pass

    async def maintain_connections(self, up_stream_friends):
        ports = {}
        attempt_connections = []
        for i in up_stream_friends:
            ports[get_port_num(i)] = i
        for connected in self.connected_servers:
            if connected[0] in ports:
                del ports[connected[0]]
        for maintain in ports:
            attempt_connections.append(asyncio.ensure_future(self.attempt_connection(ports, maintain), loop=self.loop))
        await asyncio.gather(*attempt_connections)
    async def connection_routine(self, server_port, message):
        reader, writer = await asyncio.open_connection('127.0.0.1', server_port, loop=self.loop)
        writer.write(message.encode())
        await writer.drain()
        return server_port, reader, writer

    async def handle_iamat(self, message, writer, received_time):
        try:
            if len(message) == 4:
                lat, lon = handle_latlon(message[2])
                skew = decimal.Decimal(received_time) - decimal.Decimal(message[3])
                skew_str = "+{0}".format(str(skew)) if skew >= 0 else "{0}".format(str(skew))
                response = "AT {0} {1} {2}".format(self.name,skew_str, " ".join(message[1:]))
                self.user_data[message[1]] = [self.name, skew_str, lat, lon, message[3]]
                writer.write("{0}\n".format(response).encode())
                logging.info("To Client %s: %r" % (writer.get_extra_info("peername"),response))
                tasks = [asyncio.ensure_future(writer.drain()), asyncio.ensure_future(self.propagate_message(response))]
                await asyncio.gather(*tasks)
                return 0
            else:
                return 1
        except:
            return 1

    async def handle_whatsat(self, message, writer):
        try:
            if len(message) == 4:
                items = int(message[3])
                radius = int(message[2])
                if (items > 20 or radius > 50):
                    return 1
                curr_user = self.user_data[message[1]]
                lat = "+{0}".format(curr_user[2]) if curr_user[2] >= 0 else "{0}".format(curr_user[2])
                lon = "+{0}".format(curr_user[3]) if curr_user[3] >= 0 else "{0}".format(curr_user[3])
                lat_lon = "{0}{1}".format(lat,lon)
                google_params = {
                    "location" : "{0},{1}".format(curr_user[2],curr_user[3]),
                    "radius" : message[2],
                    "key" : "AIzaSyDeiY9zr5FB8cpKie7aNRfQWoMQ0Kbf3Es"
                }
                async with aiohttp.ClientSession() as session:
                    async with session.get('https://maps.googleapis.com/maps/api/place/nearbysearch/json?',
params=google_params) as result:
                        google_data = await result.json()
                        logging.info("From Google: {0}".format(google_data))
                        google_data["results"] = google_data["results"][:items]
                nearby_locations = json.dumps(google_data, indent = 3)

                at_response = "AT {0} {1} {2} {3} {4}\n".format(curr_user[0],curr_user[1], message[1], lat_lon,
curr_user[4])
                response = "{0}{1}\n\n".format(at_response,re.sub(r'\n\n+','\n',nearby_locations))
                writer.write(response.encode())
                logging.info("To Client %s: %r" % (writer.get_extra_info("peername"),response))
                await writer.drain()
                return 0
            else:
                return 1
        except:
            return 1

    async def handle_at(self, message):
        lat, lon = handle_latlon(message[4])
        #if unknown user or time + skew (absolute time) of message is greater than time + skew of recorded data
        if(message[3] not in self.user_data or \
                (message[3] in self.user_data and \
                (decimal.Decimal(message[5]) + decimal.Decimal(message[2])) >\
                (decimal.Decimal(self.user_data[message[3]][1]) + \
                decimal.Decimal(self.user_data[message[3]][4])))):
            self.user_data[message[3]] = [message[1],message[2], lat, lon, message[5]]

    async def server_routine(self, reader, writer):
        peer = writer.get_extra_info("peername")
        logging.info("New Connection: {0}".format(peer))
        while True:
            try:
                received = await reader.readline()
                received_time = time.time()
                received_decoded = received.decode().strip("\n").strip("\r")
                if(len(received_decoded) == 0):
                    writer.write("\0".encode())
                    await writer.drain()
                    continue
                received_decomp = received_decoded.split()
                logging.info('Received From {0}: \"{1}\"'.format(peer, received_decoded))
                error = 0
                if (len(received_decomp) > 0):
                    if (received_decomp[0] == "IAMAT"):
                        error = await self.handle_iamat(received_decomp, writer, received_time)
                    elif (received_decomp[0] == "WHATSAT"):
                        error = await self.handle_whatsat(received_decomp, writer)
                    elif (received_decomp[0] == "AT"):
                        if (len(received_decomp) >= 6):
                            error = await self.handle_at(received_decomp)
                            await self.propagate_message(received_decoded)
                        else:
                            error = 1
                    else:
                        error = 1
                else:
                    error = 1
                if error:
                    error_response = "? {0}".format(received.decode())
                    writer.write(error_response.encode())
                    logging.info("To Client %s: %r" % (peer,error_response))
                    await writer.drain()
            except:
                logging.info("Lost Connection: {0}".format(peer))
                return

def main():
    server = server_class(sys.argv[1])
    server.stop_server()


if __name__ == '__main__':
    if len(sys.argv) == 2:
        if sys.argv[1] in ['Goloman','Hands','Holiday','Wilkes','Welsh']:
            main()
        else:
            print("Invalid Server Name : {0}".format(sys.argv[1]))
    else:
        print("Invalid arguments")
```