```python
import asyncio
import async_timeout
import aiohttp
import json
import time
import logging
import sys

Clients = {}
Server = sys.argv[1]
Server_port = {'Goloman':19455, 'Hands':19456, 'Holiday':19457, 'Welsh':19458, 'Wilkes':19459}
Server_relation = {'Goloman':['Hands','Holiday','Wilkes'],'Hands':['Wilkes', 'Goloman'],'Holiday':['Welsh','Wilkes', 'Goloman'], 'Wilkes':['Goloman','Hands','Holiday'],
Host = '127.0.0.1'
Key = 'AIzaSyDlrl2AypJA4EhtEYmyi6ZXvx8nOM78c70'
Google_place_api = 'https://maps.googleapis.com/maps/api/place/nearbysearch/json'

logging.basicConfig(filename=Server+'.log',format='%(asctime)s:  %(message)s', datefmt='%Y-%m-%d,%H:%M:%S', level=logging.DEBUG)


class ServerProtocol(asyncio.Protocol):

    def __init__(self, loop):
        print('Server {} protocol instance created'.format(Server))
        logging.info('Server {} protocol instance created'.format(Server))
        self.loop = loop

    def connection_made(self, transport):
        peername = transport.get_extra_info('peername')
        self.transport = transport
        logging.info('Connected from {}'.format(peername))
        print('Connected from {}'.format(peername))

    def cal_time(self, server_time, user_time):
        diff_time = server_time - user_time
        if diff_time > 0:
            return '+' + str(diff_time)
        else:
            return '-' + str(diff_time)

    def is_number(self, str):
        if str.isdigit():
            return True
        str_list = str.split('.')
        if len(str_list) != 2:
            return False
        if (str_list[0].isdigit() or str_list[0] == '') and str_list[1].isdigit():
            return True
        return False

    def basic_check(self, args):
        if len(args) < 1:
            return False
        return True
    #FLOOD=============================================
    async def flood(self, message, sender):
        for server in Server_relation[Server]:
            if sender != server:
                try:
                    logging.info('Flood info to {}'.format(server))
                    print('Flood info to {}'.format(server))
                    _, w = await asyncio.open_connection(host=Host, port=Server_port[server], loop=self.loop)
                    w.write((message+' '+sender).encode())
                    await w.drain()
                    logging.info('Success flood info to {}'.format(server))
                    print('Success flood info to {}'.format(server))
                    w.close()
                except:
                    logging.info('Fail connect to {}'.format(server))
                    print('Fail connect to {}'.format(server))
    #==================================================

    #IAMAT=============================================
    def check_IAMAT(self,args):
        if len(args) != 4:
            return False
        if (args[2].count('+') + args[2].count('-')) != 2:
            return False
        if args[2][0] != '+' and args[2][0] != '-':
            return False
        coordinate = args[2].replace('+',' ').replace('-',' ').split()
        if len(coordinate) != 2:
            return False
        if not (self.is_number(coordinate[0]) and self.is_number(coordinate[1])):
            return False
        if not self.is_number(args[3]):
            return False
        return True

    async def handle_IAMAT(self, args, time, sender):
        time_diff = self.cal_time(time, float(args[3]))
        f_message = "AT {0} {1} {2} {3} {4}".format(Server, time_diff, args[1], args[2], args[3])
        message = f_message+'\n'

        if not (args[1] in Clients) or float(Clients[args[1]].split()[5]) < float(args[3]):
            logging.info('Server {} update client info'.format(Server))
            print('Server {} update client info'.format(Server))
            Clients[args[1]] = message
            print("=====================================================")
            print("Server {0}'s current Client: {1}".format(Server,Clients))
            print("=====================================================")
            logging.info('Server {} flood client info'.format(Server))
            print('Server {} flood client info'.format(Server))
            asyncio.ensure_future(self.flood(f_message, sender),loop=self.loop)
        else:
```

```
            logging.info('Server {} received but not goting to update client info'.format(Server))
            print('Server {} received but not goting to update client info'.format(Server))
        self.transport.write(message.encode())
    #IAMAT=========================================

    #WHATSAT=======================================
    def formate_location(self, location):
        res = []
        for c in location:
            if c == '+':
                res.append(c)
            if c == '-':
                res.append(c)
        coordinate = location.replace('+',' ').replace('-',' ').split()
        return res[0]+coordinate[0]+','+res[1]+coordinate[1]

    def check_WHATSAT(self, args):
        if not (len(args) == 4 and args[2].isdigit() and args[3].isdigit()):
            return False
        if float(args[2])>=50 or float(args[2])<=0 or float(args[3])>=20 or float(args[3])<=0:
            return False
        if not args[1] in Clients:
            return False
        return True

    async def fetch(self, session, url):
        async with session.get(url) as response:
            return await response.text()

    async def handle_WHATSAT(self, args):
        url = Google_place_api + '?location={0}&radius={1}&key={2}'.format(self.formate_location(Clients[args[1]].split()[4]), str(float(args[2])*1000), Key)
        async with aiohttp.ClientSession() as session:
            response = await self.fetch(session, url)
            json_res = json.loads(response)
            json_res['results'] = json_res['results'][:int(args[3])]
            final_res = json.dumps(json_res, indent=4)
            logging.info('Server {} query Google Place API'.format(Server))
            print('Server {} query Google Place API'.format(Server))
            self.transport.write((Clients[args[1]]+final_res+'\n').encode())
    #WHATSAT=======================================

    #AT============================================
    def check_sender(self, args):
        if len(args) != 7:
            return False
        if not (args[6] in Server_port):
            return False
        return True

    async def handle_AT(self, args, message, sender):
        if not (args[3] in Clients) or float(Clients[args[3]].split()[5]) < float(args[5]):
            logging.info('Server {} update client'.format(Server))
            print('Server {} update client'.format(Server))
            Clients[args[3]]=message
            print("=====================================================")
            print("Server {0}'s current Client: {1}".format(Server,Clients))
            print("=====================================================")
            asyncio.ensure_future(self.flood(message, sender), loop=self.loop)
            self.transport.close()
        else:
            logging.info('Server {} already updated'.format(Server))
            print('Server {} already updated'.format(Server))
            self.transport.close()
    #==============================================

    #SERVER========================================
    def data_received(self, data):
        recieve_time = time.time()
        data = data.decode()
        args = data.split()
        logging.info('Server {0} recieved message: {1}'.format(Server, data))
        print('Server {0} recieved message: {1}'.format(Server, data[:-2]))
        if self.basic_check(args):
            if args[0] == 'IAMAT':
                if self.check_IAMAT(args):
                    logging.info('Server {} recieved valid IAMAT message'.format(Server))
                    print('Server {} recieved valid IAMAT message'.format(Server))
                    asyncio.ensure_future(self.handle_IAMAT(args, recieve_time, Server), loop=self.loop)
                else:
                    logging.info('Server {} recieved invalid IAMAT'.format(Server))
                    print('Server {} recieved invalid IAMAT'.format(Server))
                    self.transport.write("? {}".format(data).encode())
            elif args[0] == 'WHATSAT':
                if self.check_WHATSAT(args):
                    logging.info('Server {} recieved valid WHATSAT message'.format(Server))
                    print('Server {} recieved valid WHATSAT message'.format(Server))
                    asyncio.ensure_future(self.handle_WHATSAT(args), loop=self.loop)
                else:
                    logging.info('Server {} recieved invalid WHATSAT'.format(Server))
                    print('Server {} recieved invalid WHATSAT'.format(Server))
                    self.transport.write("? {}".format(data).encode())
            elif args[0] == 'AT':
                if not self.check_sender(args):
                    logging.info('Server {} recieved AT from invalid sender'.format(Server))
                    print('Server {} recieved AT from invalid sender'.format(Server))
                    self.transport.write("? {}".format(data).encode())
                else:
                    logging.info('Server {} recieved valid AT message'.format(Server))
                    print('Server {} recieved valid AT message'.format(Server))
                    asyncio.ensure_future(self.handle_AT(args, ' '.join(args[:-1]), args[-1]), loop=self.loop)
            else:
                logging.info('Server {} recieved invalid message'.format(Server))
                print('Server {} recieved invalid message'.format(Server))
                self.transport.write("? {}".format(data).encode())
        else:
```

```
210             logging.info('Server {} recieved invalid message'.format(Server))
211             print('Server {} recieved invalid message'.format(Server))
212             self.transport.write("? {}".format(data).encode())
213     #=============================================
214
215
216 def main():
217
218     loop = asyncio.get_event_loop()
219     coro = loop.create_server(lambda:ServerProtocol(loop), Host, Server_port[Server])
220     server = loop.run_until_complete(coro)
221     logging.info("Server {0} up at port {1}".format(Server, Server_port[Server]))
222     print("Server {0} up at port {1}".format(Server, Server_port[Server]))
223
224     try:
225         loop.run_forever()
226     except KeyboardInterrupt:
227         pass
228
229     logging.info("Server {} down".format(Server))
230     print("Server {} down".format(Server))
231     server.close()
232
233 if __name__ == "__main__":
234     main()
```