DBE - Distributed Systems

Group 07: Kevin Ross, Simon Haupt, Zacharias Sitter

1 Project Proposal

1.1 Introduction

We propose the design and implementation of a chat application using central and advanced principles of distributed systems development. The application should allow users to communicate via text messages. It should also be possible to communicate in group chats. The idea is that the users register themselves. The main server transmits the registered users as active. Afterwards the communication runs over the main server, which receives the messages and transmits them to the addressee. For the protection of the data further backup servers are installed, which get their information over the main server.

1.2 Project requirements analysis

1.2.1 Dynamic discovery

- Client registers itself by sending username and ip to server
- Server receives various incoming client's requests
- It stores the information of the clients Ip address and name in a list
- This List is **broadcasted** to all servers
- It also keeps track of various chat rooms, providing a group Ip to each multicast group

1.2.2 Crash-Fault-Tolerance

- The failure of a participant must not cause a crash of the entire system
- In the failure case a new leader election should be triggered
- Heartbeat messages sent by the leader as a failure detector when a participant crash
- Backup server which replicates the data

1.2.3 Voting

- Based on the LCR (LeLann-Chang-Roberts) Algorithms
- Every server must have a unique ID
- If the leader server goes down a new leader will be elected

- Clients can trigger election if the server does not respond
- After trigger neighbor will be searched and pinged
- Voting message will go around until a server get its UID back
- After election leader announces itself as a leader

1.3 Architecture design

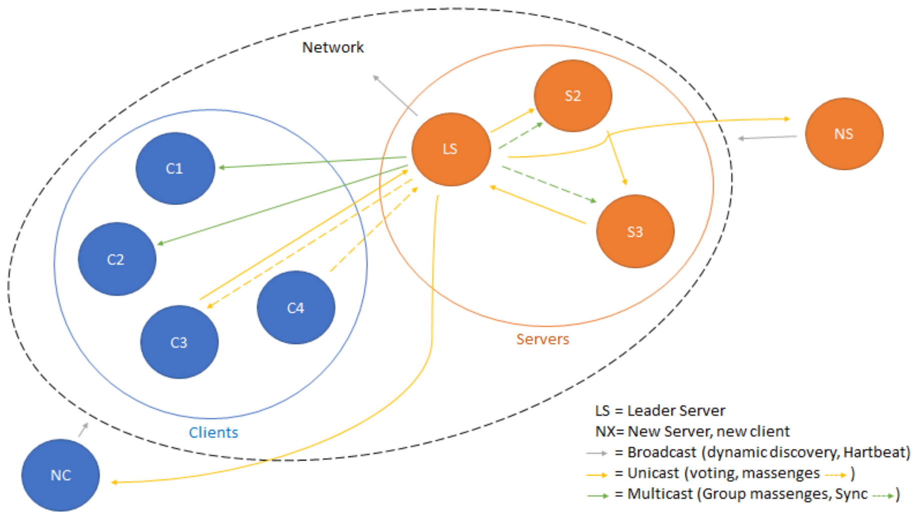

Network
S2
LS
NS
C1
C2
C3
C4
S3
Servers
Clients
NC
LS = Leader Server
NX= New Server, new client
= Broadcast (dynamic discovery, Hartbeat)
= Unicast (voting, massenges)
= Multicast (Group massenges, Sync)

Fig. 1. Architecture design draft.

1.4 Implementation overview

We plan to write the whole project in Python 3 programming language. For the remotely collaborative work we will create a public repository in GitHub. Through this opportunity, each group member can contribute on their own project topic as they prefer, and we will have a version documentation. Therefore, any programming environment can be used. Until now, PyCharm and Visual Studio are used and connected to GitHub.

2 Progress

2.1 Python socket module

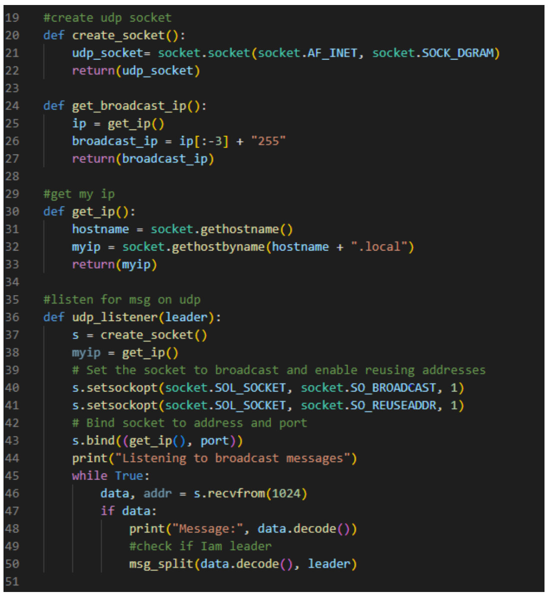
The socket library is available for the Python programming language and is used to send messages and data in a network between endpoints. There are different network structures in which sockets can be used. For example, a small implementation can take place within a local network between two computers, but also a very large implementation such as functions within the Internet.
Socket is a low-level networking interface. It is available for all modern operating systems, such as Unix, Windows and MacOS. The socket library can be used for different structures. For instance, client-server architectures or multicast applications can be created.
For our distributed systems project we will use this module and build a client-server architecture.

2.2 Current State

The whole project is embedded in GitHub for collaboration. The repository can be found under this url: https://github.com/hackschnitzel09/Github_DS_Project As of today, the system is able to send messages from one client via the leader-server to another client via udp sockets based on human readable names like Simon. To implement this a users.json and a servers.json file provides the needed information and are maintained by the leader server. Now most of the code can be found in the utility.py which is included in the server and client for accessing the needed code. In the server.py and client.py are only variables found which the processes need to be specified before running. The voting process can identify the neighbor server according to the id which should be given when joining the network and send a voting message. In the following will be some explanations of the code by functions (see screenshots):

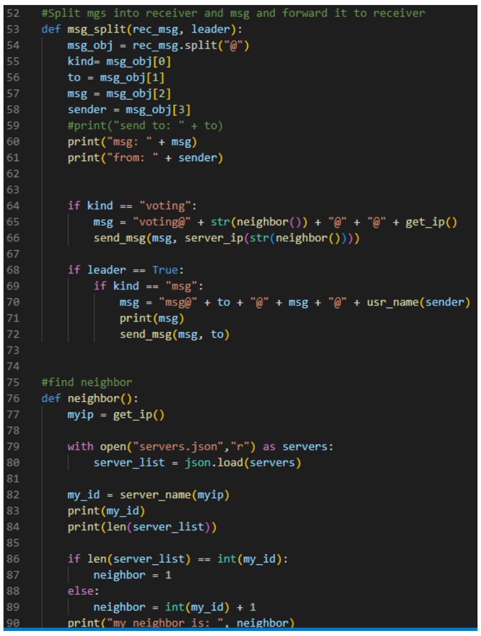
- Get_ip(): reads the ip based on the system hostname
- Get_brodcast_ip: extracts the ip of the local machine and changes the last thre digits to 255
- Udp_listener(): opens a socket for receiving udp messages and calles msg_split() for further steps
- Msg_split(): divides the received message to extract information. The syntax of messages is which kind @ to whom @ the message @ from whom. The kind is important for the server to know what to do with it. If kind is msg the leader will send the msg to the defined receiver. Therefore he uses usr_ip() to get the latest ip to the name and sends it via send_msg(). If the kind is voting the msg will for now only be forwarded to the neighbor.
- The neighbor(): gives the server with the next higher id back
- Send_msg() and send_brodcast(): sending messages either to a fixed ip or the broadcastip

- Usr_ip(), usr_name(), sever_ip() and server_name(): are translating ether names to id or ips or the otherway around based on the information given in the .json files



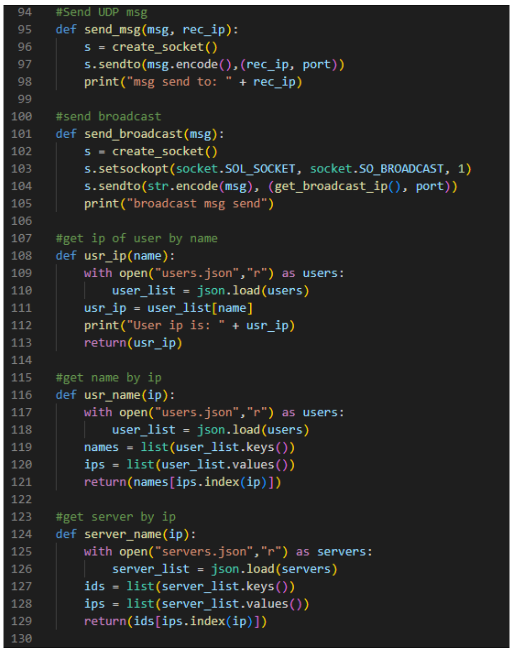
```python
#create udp socket
def create_socket():
    udp_socket= socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    return(udp_socket)

def get_broadcast_ip():
    ip = get_ip()
    broadcast_ip = ip[:-3] + "255"
    return(broadcast_ip)

#get my ip
def get_ip():
    hostname = socket.gethostname()
    myip = socket.gethostbyname(hostname + ".local")
    return(myip)

#listen for msg on udp
def udp_listener(leader):
    s = create_socket()
    myip = get_ip()
    # Set the socket to broadcast and enable reusing addresses
    s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    # Bind socket to address and port
    s.bind((get_ip(), port))
    print("Listening to broadcast messages")
    while True:
        data, addr = s.recvfrom(1024)
        if data:
            print("Message:", data.decode())
            #check if Iam leader
            msg_split(data.decode(), leader)

```

Figure 2: Code Screenshot 1



```python
#Split mgs into receiver and msg and forward it to receiver
def msg_split(rec_msg, leader):
    msg_obj = rec_msg.split("@")
    kind= msg_obj[0]
    to = msg_obj[1]
    msg = msg_obj[2]
    sender = msg_obj[3]
    #print("send to: " + to)
    print("msg: " + msg)
    print("from: " + sender)


    if kind == "voting":
        msg = "voting@" + str(neighbor()) + "@" + "@" + get_ip()
        send_msg(msg, server_ip(str(neighbor())))

    if leader == True:
        if kind == "msg":
            msg = "msg@" + to + "@" + msg + "@" + usr_name(sender)
            print(msg)
            send_msg(msg, to)


#find neighbor
def neighbor():
    myip = get_ip()

    with open("servers.json","r") as servers:
        server_list = json.load(servers)

    my_id = server_name(myip)
    print(my_id)
    print(len(server_list))

    if len(server_list) == int(my_id):
        neighbor = 1
    else:
        neighbor = int(my_id) + 1
    print("my neighbor is: ", neighbor)
```

Figure 3: Code Screenshot 2



```python
#Send UDP msg
def send_msg(msg, rec_ip):
    s = create_socket()
    s.sendto(msg.encode(),(rec_ip, port))
    print("msg send to: " + rec_ip)

#send broadcast
def send_broadcast(msg):
    s = create_socket()
    s.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
    s.sendto(str.encode(msg), (get_broadcast_ip(), port))
    print("broadcast msg send")

#get ip of user by name
def usr_ip(name):
    with open("users.json","r") as users:
        user_list = json.load(users)
    usr_ip = user_list[name]
    print("User ip is: " + usr_ip)
    return(usr_ip)

#get name by ip
def usr_name(ip):
    with open("users.json","r") as users:
        user_list = json.load(users)
    names = list(user_list.keys())
    ips = list(user_list.values())
    return(names[ips.index(ip)])

#get server by ip
def server_name(ip):
    with open("servers.json","r") as servers:
        server_list = json.load(servers)
    ids = list(server_list.keys())
    ips = list(server_list.values())
    return(ids[ips.index(ip)])

```

Figure 4: Code Screenshot 3

2.3 Challenges

The biggest challenge accruing is the ability to receive broadcast messages because the messages send to the broadcast ip are not transmitted further to the udp listening sockets. There were several trys in different environments like hosting the machines in VirtualBox with several different network configurations as well as hosted machines on a NAS system. That's why the decision was made to put the focus on the other given tasks and lower the priority of dynamic discovery.

2.4 Prospect

The next steps in this project will be the implementation of the voting system. Now messages are forwarded to the neighbor, but he is also forwarding the msg without interaction. The finding of the neighbor needs to be optimized for the case of no direct follower id. Also, now only one port is in place the idea is to put voting msgs for instance on another port.

Afterwards new servers and clients should be able to be recorded by the leader server in the .json files if no leader is available a voting will be triggered. Syncing this list to other servers would be the next step. Also, a heartbeat between the leader and the other servers and maybe clients as well, needs to be implemented along with the voting trigger if a beat is not received.

For getting the system running multithreading is needed to be implemented for clients to send and receive messages at the same time as well as servers for forwarding, listening heartbeat etc.

Lastly the broadcast issue will be taken into elaboration.

References

1. Aiello M..: Lecture notes Distributed Systems (Summer term 2022)
2. https://docs.python.org/3/library/socket.html (Last checked, 22.06.2022)