Gnutella Client

Report – Term Project

COMP 3010
April 25, 2022

Juan A. (armijosj@myumanitoba.ca)

1. How it works.

Joining the Network

To begin the Gnutella Protocol, we first need to connect the peers in the network. The code for the peers is in "*gnutella-peer.py".* To create the first peer in the Network we will use the following command:

- *python3 gnutella-peer.py [PORT].*
- **Example**: *python3 gnutella-peer.py 8025*

For a peer to join the Network, it has to know about the existence of a peer that it is already on the network and running. To create a new peer and join the network the following command need to be executed:

- *python3 gnutella-peer.py [PORT] [peerHost:peerPort].*
- **Example**:
    - *python3 gnutella-peer.py 8027 DESKTOP-3T:8025*
    - *python3 gnutella-peer.py 8026 DESKTOP-3T:8025*

These two commands will join the two recently created peers to the peer that is already running on port 8025.

Assigning a directory

The peer will ask for a directory so it can look or the files that it contains. The directory has to be placed in the same workspace as where the code is running. To provide a directory user will have to type in the name of it. This is an important step because that means that two devices could share their information no matter where they are connected from.

Requesting a file

For a peer to request a file, user will have to send a SIGTSTP signal by pressing CTRL+Z on the terminal, this will ask for the filename that the peer wants to look for, after entering the filename a request will be made to all the neighboring peers.

Receiving the file

The peer will receive the information that it is contained in that file and print it out to the terminal. If there is a case in which no peer has that file, the peer that requested the file will then timeout.

Quitting the Network

To terminate a peer, close the connections, and exit the network the user will have to press CTRL+C. This will send a BYE message to all connected peers so they know it is closing its connection.

2. Implementation

The **messages** that are being passed are JSON objects or dictionaries in python. The are the following:

```
ping = {"type": "PING", "host": HOST, "port": PORT, "id": None}
pong = {"type": "PONG", "host": HOST, "port": PORT, "id": None}
query ={"type": "QUERY","host": HOST, "port": PORT, "file": None, "id": None}
queryHit = {"type": "QUERY-HIT", "host": HOST, "port": PORT, "hasFile": None, "id": None}
bye = {"type": "BYE", "host": HOST, "port": PORT}
```

The **Domain Specific Objects** that I'm using are timeoutQueue and message:
timeoutQueue: Tracks all messages which have a timeout time and check if there are any timeouts in it so they could be handled.

```
class timeoutQueue:
class message:
```

The **communication** is being done via TCP. And the file transferring is being done via UDP.

```
# UDP socket - File Transfering
udpSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
udpSocket.bind( (HOST, PORT) )

# TCP socket - Peer communication
mainSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mainSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
mainSocket.bind((HOST, PORT))
mainSocket.listen()
mainSocket.setblocking(0)
```

TCP: works with a ***select*** that either accepts new connections or receives messages from peers. And then manages each type of message accordingly.

```
(readable, writable, error) = select.select([mainSocket, ] + myPeers, [], [], timeout)
```

UDP: is active only for a timeout time or until it receives a file whenever a request is made. This is being handled in a new thread.

```
udpThread = threading.Thread(target=listenData)
```

```
def listenData():
    try:
        udpSocket.settimeout(5)
        data11, addr11 = udpSocket.recvfrom(2048)
        print("\nContent of requested file:")
        print("{}\n".format(data11.decode('utf-8')))
    except socket.timeout:
        print("TIMEOUT! It seems that the file does not exist")
    except Exception as e:
        print(e)
```

3. Protocol Implementation

- **PING**

  When a peer joins the Network, it will send a PING message to the peer that it I trying to connect. If the peer does not receive a PONG message back it will timeout.

```
print("Sending PING")
ping["id"] = str(uuid.uuid3)
s.sendall(json.dumps(ping).encode())
messages.addMessage( message(ping) )
myPeers.append(s)
```

- **PONG**

  When a peer receives a PONG then it removes the PING from the timeoutQueue and the peer knows the connection is established.

```
elif msg["type"] == "PONG":
    print("PONG received, connection established successfully with peer on port: {}".format(msg["port"]))
    messages.removeByID(msg)
```

- **QUERY**

  When we want to send a QUERY, we ask for the filename and send it to all peers.

```
def sendQuery(signum, frame):
    print("\n - Asking for file...")
    file = input("Filename: ")
    query["file"] = file
    query["id"] = str(uuid.uuid1())
    #add to queries so if when I recieve do not process again
    myQueries.append(query)

    for p in myPeers:
        #send
        p.sendall(json.dumps(query).encode())
        messages.addMessage( message( query ))
```

  When we receive a QUERY, we check if we have that file, if we do, we send it via UDP, if we don't then we pass the query to other peers:

```
elif msg["type"] == "QUERY":
    if msg not in myQueries:
        print("\nGot a query for {}" .format(msg["file"]))
        myQueries.append(msg)

        #Look if I have the file
        queryHit["hasFile"] = retireveData(msg)
        if not queryHit["hasFile"]:
            for peer in myPeers:
                #do not send to the peer that sent it
                if s is not peer:
                    #replicate query to all peers
                    print("Passing query to another peer.")
                    peer.sendall(json.dumps(msg).encode())
                    messages.addMessage( message (msg) )
```

- **QUERY-HIT**
  We send a QUERY-HIT after we checked for the requested file, and send that information to the peer that requested it. When we receive a QUERY-HIT, we remove the QUERY message from the timeoutQueue.

```
elif msg["type"] == "QUERY-HIT":
    messages.removeByID(msg)
```

- **BYE**
  When we exit the program by CTRL+C we send a BYE message to all of our peers so the connection can be closed accordingly.

```
def terminatePeer(signum, frame):
    print("\n\nOk exiting\n")
    for p in myPeers:
        p.sendall(json.dumps(bye).encode())
    exit()
```

  When we receive a BYE message, we close the connection and remove that peer from our list of Peers

```
elif msg["type"] == "BYE":
    print("A peer has closed the connection")
    s.close()
    myPeers.remove(s)
```

4. Course material used

- **Multithreading**
  I am creating a new thread of execution each time I request a file. This new thread of executing will be listening to a UDP socket for the file or timeout after some time.

- **Signal Handling**
  I am adding signal handling to interrupt the program whenever I need input from the user or when the user wants to terminate a peer.

- **TCP - Sockets**
  Main communication is being done via TCP sockets.

- **UDP – Sockets**
  File transferring is being done via UDP sockets.

- **JSON**
  I communicate by sending JSON objects through the sockets.

5. Graphical representation

Slides are shown to represent the PING and PONG messages.