QuicFunc.py

import libraries and announce constants

```python
from QuicPackage import *  # import the QuicPackage file for creating packets and using its functions
import socket  # import the socket library for creating a socket and sending packets
from functools import cmp_to_key  #tchelet
import re  #tchelet
import time  # import the time library to get the time of the packet creation, for timeouts and more

send_sleep = 0.002  # if we want faster result but a greater chance to loss packet than we can do 0.0002
                    #depends on the packet loss %

```

recreate && send packets

```python
# function for recreating packets, we get encoded packet and we decode it, each variable acording to its kind
11 usages
def recreate_package(str_package: bytes):
    list_package = str_package.decode("utf-8").split( sep: "&",  maxsplit: 4)  # the packet encoded using & between each field
    new_package = QuicPackage(int(list_package[1]), list_package[3], list_package[4])  # create the new packet
    new_package.recreate_from_str(float(list_package[2]),
                                  list_package[0])  # setting the time anf sequence number fot the packet
    return new_package  # return the new packet


# function for sending the packet using the "sendto" function from the socket library
14 usages
def send_package(package: QuicPackage, sock: socket, ADDR):  # event is passed by binders.
    """Handles sending of messages."""
    sock.sendto(package.encode_package(), ADDR)


```

send packaged from file

function to get information from file, store it in packets and send the packets

```python
# function to read information from the file, store it in packet objects and sending the packets
1 usage
def send_packages_from_file(file1, sock: socket, ADDR, package_list, no_ack, connect):
    """Handles sending of files (attachments)."""
    global send_sleep
    pos = 0
    print("Sending file...")

    while True:  # while we can still read from the file
        slice = file1.read(268)  # read 268 bytes from the file
        # print("hello\n",slice)
        if not slice:  # if we are done reading from the file
            time.sleep(0.5)  # wait for 0.5 seconds
            # new_package = QuicPackage(pos, "DONE",connect)
            # send_package(new_package, sock, ADDR)
            send_last_ack(connect, sock, ADDR)  # send the last packet and wait for ack
            return package_list  # return all the packets we sent from the file

        time.sleep(send_sleep)  # wait
        new_package = QuicPackage(pos, slice, connect)  # create the new packet
        package_list[new_package.seq] = new_package  # add the new packet to the list
        send_package(new_package, sock, ADDR)  # send the packet
        pos += 1  # increase the position of the packets by one

```

send last ack

intended to send the last packet and wait for acknoledgment
to verify that we ended the file sending stream

```python
# function to send the last packet and wait for response from the server that we it was received successfully
2 usages
def send_last_ack(con, client_socket, SERVER_ADDR):
    print("Send last")
    pack = QuicPackage( pos: 0,  payload: "DONE", con)  # create the last packet
    send_package(pack, client_socket, SERVER_ADDR)  # send the last packet
    wait_for_ack(con, client_socket, SERVER_ADDR)  # wait for the server to receive it, send again if it didn't


# function to wait for the server to receive the last packet, and send again if it didn't
2 usages
def wait_for_ack(con, client_socket, SERVER_ADDR):
    print("Enter ack")
    start = time.time()
    timeout = 1  # Timeout for retransmission
    while True:  # while the server don't get the packet
        try:
            client_socket.settimeout(timeout)  # wait
            data, n = client_socket.recvfrom(1024)  # receive the acknowledged packet from the server
            new_pack = recreate_package(data)  # decoded the packet

            if new_pack.payload == "DONE":  # if the packet we received is the acknowledgment packet from the server
                print("sent last packet")
                return  # quit the function

        except socket.timeout:
            if time.time() - start > timeout:  # if we didn't receive the acknowledgment packet from the server in time
                print("Didn't receive last ack in time, resending...")
                send_last_ack(con, client_socket, SERVER_ADDR)  # send again the last packet
                return
```

wait for last ack
server function wo wait to receive the last packet and to send acknowledgement about receiving it

```python
# function for the server to receive the last ack and send an acknowledgment packet back to the client
1 usage
def wait_for_last_ack(server_socket, new_package):
    start = time.time()
    timeout = 1  # Timeout for retransmission
    while True:  # while we didn't get the last packet
        try:
            server_socket.settimeout(timeout)  # wait
            pack, address = server_socket.recvfrom(1024)  # receive the last packet
            new_pack = recreate_package(pack)  # decode the packet

            if new_pack.payload == "DONE":  # if the packet received is the last one, send acknowledgment packet
                # print("The connection ID is: " + CONNECTION_ID)
                send_package(new_pack, server_socket, address)  # send the acknowledgment packet
                return

        except socket.timeout:
            if time.time() - start > timeout:  # if we didn't receive the last packet from the server in time, wait
                print("Didn't receive last pack in time, waiting again...")
                start = time.time()  # reset the start time for the next timeout

        except Exception as e:
            print(f"An error occurred: {e}")
```

recv packaged list from file && compare (simple compare)
functions to receive packets from the file sending by the client and to compare packets by their position

```python
# function to receive the packets from the client
1 usage
def recv_package_list_from_file(sock: socket):
    package_list = []  # initialize a list of packets
    while True:  # while we get packets
        package, addr = sock.recvfrom(1024)  # receive the packets
        if not package:  # if the packet we received is None, there are no longer packets to receive
            print("done recv")
            return package_list  # return the list of packets
        new_package = recreate_package(package)  # decode the received packet
        if new_package.payload == "DONE":  # if its the last one
            # print("done recv")
            wait_for_last_ack(sock, new_package)  # wait to receive it and send ack
            return package_list
        # time.sleep(0.00002)
        print("ack for ", new_package.getpos())  # send ack for receiving packets
        new_package.send_ack(sock, addr)  # send ack
        package_list.append(new_package)  # add the new packet to the list


# function to compare packets by position
2 usages
def compare(x, y):
    return x.getpos() - y.getpos()# return True if the x packet is bigger that the y packet by its position

```

remove duplicates && write to file
functions to remove duplicated packets
so we won't write duplicated to the file

```python
# function to remove duplicates packets, we know its duplicate by position
1 usage
def remove_duplicates(list):
    seen_x = set()
    unique_packets = []

    for pac in list:
        if pac.getpos() not in seen_x:
            unique_packets.append(pac)
            seen_x.add(pac.getpos())

    return unique_packets


# function to write the packets received from the file sent by the client into a new file
1 usage
def write_to_file(package_list: [], file1):
    unique_sorted_package_list = remove_duplicates(package_list)# remove duplicated packets
    package_list = sorted(unique_sorted_package_list, key=cmp_to_key(compare))# sort the packets by position

    f = open("packetPos.txt", "w")# open the packetPos file to write their the packets position
    for payload in package_list:
        f.write(str(payload.getpos()) + "\n")  #print packet pos
        # f.write("\n")
        file1.write(payload.payload)  #print the packet itself
    f.close()

```

resend lost packages
function to resend lost packets
called by the client

```python
# function to resend lost packets
1 usage
def resend_lost_packages(package_list, no_ack, sock, ADDR):
    print("resend lost package")
    pack = no_ack[0] # send the first packet in the no_acks list, its the first packet that need to resend
    print("I lost this :", pack.seq, "and this is the pos somehow", pack.getpos())
    pack.update_for_resend() # update the packet for resend
    send_package(pack, sock, ADDR) # send the packet again
    package_list[pack.seq] = pack # add the packet to the list of sending packets

```