SEQ AND ACK GAME - TERM PROJECT

Computer Networks

3 January 2022

Rabia Kodal

TABLE OF CONTENTS

INTRODUCTION

The project is based on TCP and UDP, which form the basis of the Computer Networks course. The project consists of a UDP based client and server communication and it has been made into a game.UDP does not guarantee that the delivery phase will go smoothly. If a packet is lost due to a network problem, it may be lost forever. Data transmitted over the UDP service may be detected as abnormal, or the transmitted data may be lost. This project aims to show lost and unusual packets in server and client relationship in the context of UDP socket programming.The game revolves around a client (Player 1) that sends a message with SEQ, ACK, and packet length values, and a server (Player 2 for manual mode) that responds with a response, along with its own SEQ, ACK, and packet length values. The program consists of an automatic and a manual mode.

MOTIVATION

The aim of this project is to explain the client-server communication in a more tangible way. This is done by creating a game out of the relationship. By playing this game, the user is able to understand socket programming and see the differences between client and server.

RULES OF ACK & SECK GAME

Auto Mode

At the start of the game, the program asks the player if they want to play in auto mode or manual mode. If auto mode is chosen, the player (client) plays with a computer. The client sends a message with seq, ack and packet length values. The server side automatically calculates the new numbers and immediately sends back the reply. The client can detect duplicate and corrupted packets.

Manual Mode

In manual mode, there are two players. One is a client and the other is a server, however in this case the server acts similarly to a client. The client starts the game by sending a message and the seq, ack and packet length numbers. When this message is received by the server, it means it is the Player 2's turn to answer. Player 2 answers the message first, and then calculates and sends the numbers. The client and the server can detect duplicate and corrupted packets.

PSEUDOCODES OF SOCKET PROGRAMMING

UDP Client Pseudocode

1. Create a socket
2. Bind this socket with an IP address and a port number
3. Loop
    a. Send message and seq, ack, packet length numbers to the server
    b. Receive answer from the server
4. Close socket

UDP Server Pseudocode

1. Create a socket
2. Bind this socket with the same IP address as the client part, specify a port number
3. Loop
    a. Receive message from client
    b. Send a reply to the client
4. Close socket

CODE DESCRIPTION

UDP Client

Starting from the client side, we first created a socket called clientSock. We used the bind() method from the 'socket' library to assign an IP address and a port number to the socket instance. Then, we defined a function called send_msg.

Inside the send_msg function, the program first asks the player if they want to play in auto mode or manual mode.

1. Auto Mode

If auto mode is chosen, the player will play with a computer, which means that the replies from the server will be given automatically. Firstly, we get a message from the player, and then we ask the player to enter their seq, ack and packet length numbers. We split the seq, ack and pl values so that they can be sent to the server side individually. We then use the socket that we have created at the beginning of our code, and send that to the server side. Client score adds up to be 1 at this point. We use the recvfrom() function from the socket library to receive the reply from the server side, and decode this data and display it to the player.

After this initialization of the game, the program starts to search for possible errors that the client input may create while the game is continuing. The program keeps adding one point to the client, and decreases one point as errors happen. The error detections that are implemented in this part are duplicate and corrupted packet errors.

a. Duplicate message error

   There is an empty list at the beginning for both auto and manual modes, and with each packet being sent to the other side, every packet that is sent becomes an element in the list. This is done to detect duplicate packets. If the last element and the element before it is the same, a duplicate packet is detected. The score is reduced because of the duplicate message and a message is printed to make a warning about the duplicate packet.

b. Corrupted message error

   For the client side, the correct packet that should be sent should have a seq number that is equal to the server's ack number and a ack number that is the sum of the server's seq and packet length numbers. If these conditions are not met, there is a corrupted message error and the player is notified.

2. Manual Mode

The manual part for the client side has the same logic as the auto mode. The only difference is that there will be another player on the server side, which we will get to in the UDP Server part.

UDP Server

As for the server part, we create a socket called serverSock. We assign an IP address and specify a port number to it. We receive the client's (i.e. the player's) answer for the mode selection.

1. Auto Mode

In the auto mode case, there is a built-in timeout function that puts an error message on display in case the client does not send any data in 10 seconds to the server. After receiving the data from the client side, we split this data. In case of no errors, the new sequence number becomes the received ack number. The new ack number becomes the summation of the received seq and packet length values. The new packet length is randomized and determined from the range of 0-100. After these calculations, the new numbers are sent to the client side.

If the calculations explained above are incorrect, the program displays a "Corrupted packet or duplicated packet" message.

2. Manual Mode

In the manual mode, there are two players. The server side behaves more like a client in this part. Player 2 enters a reply, calculates and sends back the expected numbers to the client side. If the correct numbers are sent, the server receives one point.

If the player sends the same seq, ack and pl numbers that they have sent before, the player is told that they have tried to send a duplicate message. In another scenario, if the calculations are wrong, the player is told that they have tried to send a corrupted packet, and that they should try again. In both cases, the server's score is decreased by one.

SOURCE CODE

Client source code:

```
import socket
from numpy import random

clientSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
clientSock.bind(('127.0.0.2', 3997))


def send_msg():
    print('Do you want to play in auto mode or manual mode?')
```

```
    mode_selection = input()

    if mode_selection == 'auto':
        # score count declaration
        client_count = 0

        i = 0

        clientSock.sendto(bytes(mode_selection, 'utf-8'), ('127.0.0.2', 2897))

        list = []

        while i == 0:

            print('Enter your message')
            msg = input()
            print('Enter your sequence, acknowledgement and packet length values:')
            seq, ack, pl = input().split()
            print('Ready to send data')

            clientSock.sendto(bytes(msg + ' ' + seq + ' ' + ack + ' ' + pl, 'utf-8'),  # data to
be sent
                              ('127.0.0.2', 2897)  # details of destination
                              )

            # code to receive data
            client_count += 1


            print('Ready to receive' + ' client score = ' + str(client_count))

            # data => received
            data, addressInfo = clientSock.recvfrom(100)
            print(data.decode('utf-8') + ' from ' + str(addressInfo))
            i += 1

            packet = seq + " " + ack + " " + pl
            list.append(packet)


        while i>=1 and i< 10:

            print('Enter your message')
            msg = input()
            print('Enter your sequence, acknowledgement and packet length values:')
            seq, ack, pl = input().split()


            seqS, ackS, plS = data.split()


            packet = seq + " " + ack + " " + pl
            list.append(packet)


            if (seq == str(int(ackS))) and (str((int(plS)+int(seqS))) == ack):
                print('Ready to send data')

                clientSock.sendto(bytes(msg + ' ' + seq + ' ' + ack + ' ' + pl, 'utf-8'),  # data
to be sent
                                  ('127.0.0.2', 2897)  # details of destination
                                  )
```

```
                # code to receive data
                client_count += 1

                print('Ready to receive' + ' client score = ' + str(client_count))

                # data => received
                data, addressInfo = clientSock.recvfrom(100)
                print(data.decode('utf-8') + ' from ' + str(addressInfo))
                i += 1

            elif list[-1] == list[-2]:
                client_count -= 1
                i += 1
                print("client score = " + str(client_count))
                print("This is a duplicate packet!")

            else:
                client_count -= 1
                i += 1
                print("client score = " + str(client_count))
                msg1 = '!!Corrupted packet!! try again, please more careful!!'
                print(msg1)
        #print("server score:   "+UDPServer.server_count)




    if mode_selection == 'manual':

        # score count declaration

        client_count = 0

        i = 0

        clientSock.sendto(bytes(mode_selection, 'utf-8'), ('127.0.0.2', 2897))

        list = []

        while i == 0:
            print('Enter your message')

            msg = input()

            print('Enter your sequence, acknowledgement and packet length values:')

            seq, ack, pl = input().split()

            print('Ready to send data')

            clientSock.sendto(bytes(msg + ' ' + seq + ' ' + ack + ' ' + pl, 'utf-8'),  # data to
be sent

                              ('127.0.0.2', 2897)  # details of destination

                              )

            # code to receive data

            client_count += 1
```

```
            print('Ready to receive' + ' client score = ' + str(client_count))

            # data => received

            data, addressInfo = clientSock.recvfrom(100)

            print(data.decode('utf-8') + ' from ' + str(addressInfo))

            i += 1

            packet = seq + " " + ack + " " + pl
            list.append(packet)

        while i >= 1 and i < 10:

            print('Enter your message')

            msg = input()

            print('Enter your sequence, acknowledgement and packet length values:')

            seq, ack, pl = input().split()

            reply, seqS, ackS, plS = data.split()

            packet = seq + " " + ack + " " + pl
            list.append(packet)

            if (seq == str(int(ackS))) and (str((int(plS) + int(seqS))) == ack):

                print('Ready to send data')

                clientSock.sendto(bytes(msg + ' ' + seq + ' ' + ack + ' ' + pl, 'utf-8'),  # data
to be sent

                                  ('127.0.0.2', 2897)  # details of destination

                                  )

                # code to receive data

                client_count += 1

                print('Ready to receive' + ' client score = ' + str(client_count))

                # data => received

                data, addressInfo = clientSock.recvfrom(100)

                print(data.decode('utf-8') + ' from ' + str(addressInfo))

                i += 1

            elif list[-1] == list[-2]:
                client_count -= 1
                i += 1
                print("client score = " + str(client_count))
                print("This is a duplicate packet!")

            else:
```

```
                client_count -= 1

                i += 1

                print("client score = " + str(client_count))

                msg1 = '!!Corrupted packet!! try again, please more careful!!'

                print(msg1)
        #print("server score:   "+UDPServer.server_count)
    else:
        print('Invalid answer. Try again')


send_msg()
```

Server source code:

```
import socket
from random import randrange

serverSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
serverSock.bind(('127.0.0.2', 2897))
server_count = 0
total_pl = 0

# receive player's answer for mode selection
mode, addressInfo = serverSock.recvfrom(100)

while True:

   print('Ready to receive')

   # if player chooses auto, automatically calculate and send seq, ack and pl numbers
   if mode.decode('utf-8') == 'auto':

       # code to receive data
       data, addressInfo = serverSock.recvfrom(100)
       server_count += 1
       print(data.decode('utf-8') + ' from ' + str(addressInfo))
       print(' Ready to send' + ' server score: ' + str(server_count))

       # split received data
       msg, seq, ack, pl = data.split()

       # calculate new seq, ack numbers
       # received ack number is now server side's seq number
       new_seq = str(int(ack))

       # add total pl, add 1 to it and make it the new ack number  (expected value from client
side)
       old_seq = str(int(seq))
       old_ack = str(int(ack))
       old_pl = str(int(pl))


       total_pl = int(pl)
       seq = int(seq)
       new_ack = str(total_pl + seq)
```

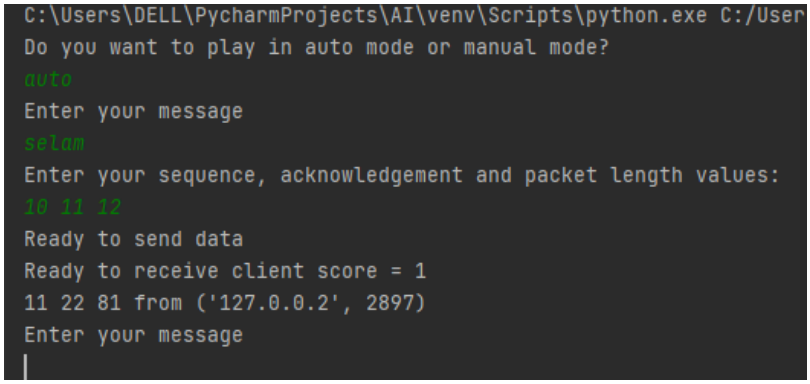
```
        # packet length is random, between 0 and 100
        new_pl = str(randrange(100))



        # code to send calculated data
        #serverSock.sendto(bytes(new_seq + ' ' + new_ack + ' ' + new_pl, 'utf-8'),
('127.0.0.2', 3997))
        ################################################
        ack = str((int(old_seq)+int(old_pl)))
        if new_seq != old_ack or new_ack != ack:
            print("False")
            Negative_response = "Corrupted packet or duplicate packet"
            serverSock.sendto(bytes(Negative_response + ' ' + new_seq + ' ' + new_ack + ' ' +
new_pl, 'utf-8'),
                              ('127.0.0.2', 3997))
        else:

            print("Correct")
            serverSock.sendto(bytes( new_seq + ' ' + new_ack + ' ' + new_pl, 'utf-8'),
('127.0.0.2', 3997))



    # if player chooses manual, let player 2 (server) calculate and type their own reply
    elif mode.decode('utf-8') == 'manual':
        # receive data
        data, addressInfo = serverSock.recvfrom(100)

        print(data.decode('utf-8') + ' from ' + str(addressInfo))
        print(' Ready to send' + ' server score: ' + str(server_count))

        i = 0


        list = []

        while i>=0 and i<10:
            print('Enter your reply')
            reply = input()
            print('Enter your calculated seq, ack, packet length numbers: ')
            seq1, ack1, pl1 = input().split()
            # split received data
            msg2, seq2, ack2, pl2 = data.split()
            # print(seq)
            # print(ack)
            #  print(pl)
            # print('*')

            packet = seq1 + " " + ack1 + " " + pl1
            list.append(packet)


            if (seq1 == str(int(ack2))) and (str((int(pl2) + int(seq2))) == ack1):
                print('Ready to send data')

                serverSock.sendto(bytes(reply + ' ' + seq1 + ' ' + ack1 + ' ' + pl1, 'utf-8'),
# data to be sent
                                  ('127.0.0.2', 3997)  # details of destination

                                  )
```

```
        # code to receive data

        server_count += 1

        print('Ready to receive' + ' server score = ' + str(server_count))

        # data => received

        data, addressInfo = serverSock.recvfrom(100)

        print(data.decode('utf-8') + ' from ' + str(addressInfo))

        i += 1



    elif len(list)>1 and list[-1] == list[-2]:
        server_count -= 1
        i += 1
        print("server score = " + str(server_count))
        print("This is a duplicate packet!")

    else:

        server_count -= 1

        i += 1

        print("server score = " + str(server_count))

        msg1 = '!!Corrupted packet!! try again, please more careful!!'

        print(msg1)
```

OUTPUTS

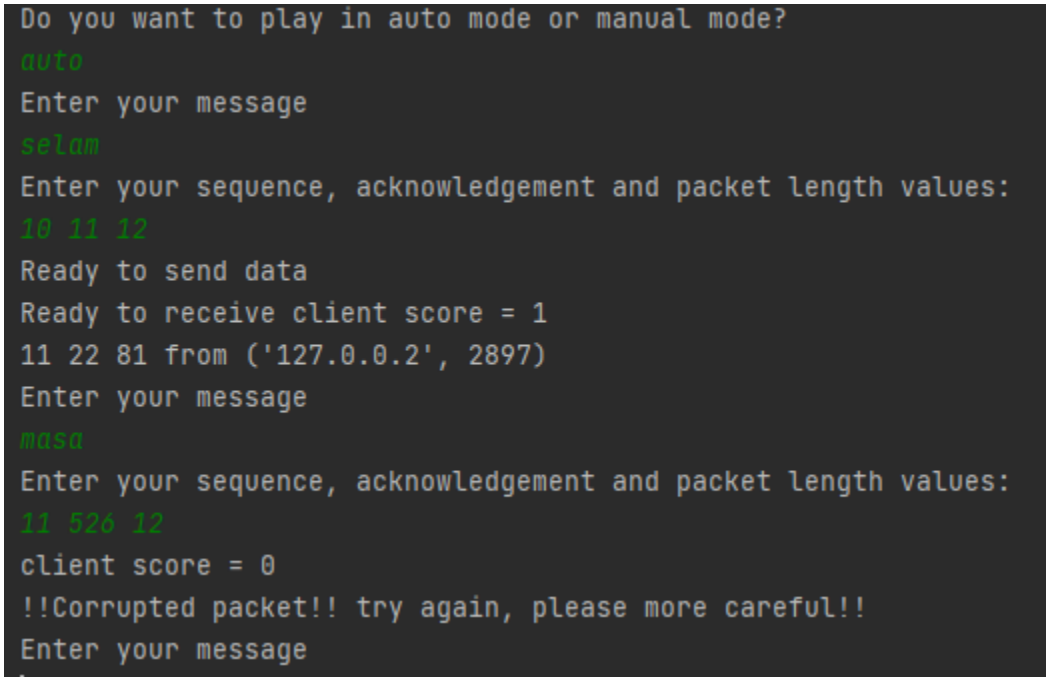
Auto

Client side:



Correct Packet

> (hello, 10, 11, 12) packet sent by client (user)
>The part after the "Ready to receive" line has been sent by the server. This means that the server has successfully received the client's message.



Wrong packet

>The client sent the second message (masa, 11,256,12). The sequence number of this package should have been 22 and the acknowledgment number should have been 92. This indicates that the package is corrupted and the client is prompted to resend the message.

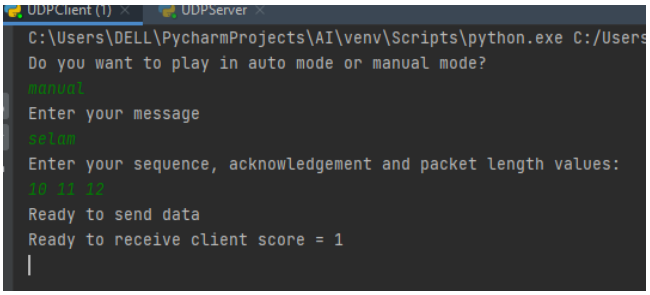
Server side:

```
C:\Users\DELL\PycharmProjects\AI\venv\Scripts
Ready to receive
selam 10 11 12 from ('127.0.0.2', 3997)
 Ready to send server score: 1
Correct
Ready to receive
```

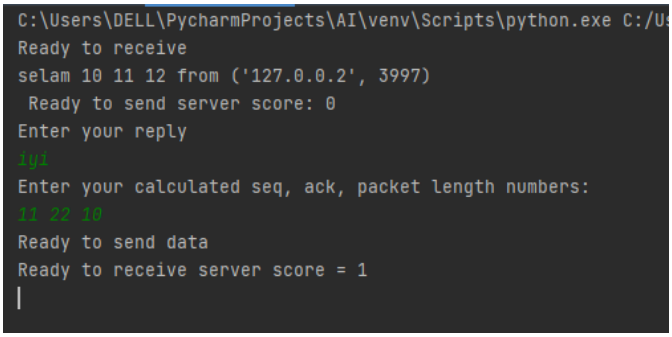
>The server received and replied to the first packet sent by the client, but did not receive this message because the seq and ack numbers of the second message sent by the client were not correct.

Manual

Client: packet = (selam, 10, 11, 12)



Server: packet = (iyi, 11, 22, 10)



Client: packet = (noldu, 22, 21, 11)

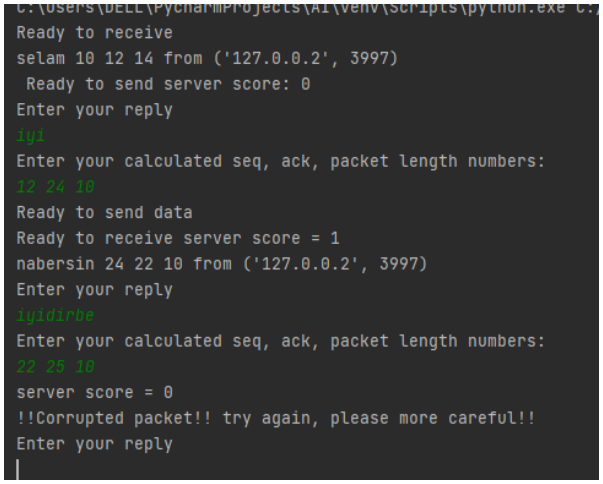
```
C:\Users\DELL\PycharmProjects\AI\venv\Scripts\python.exe C:/Users/DEL
Do you want to play in auto mode or manual mode?
manual
Enter your message
selam
Enter your sequence, acknowledgement and packet length values:
10 11 12
Ready to send data
Ready to receive client score = 1
iyi 11 22 10 from ('127.0.0.2', 2897)
Enter your message
noldu
Enter your sequence, acknowledgement and packet length values:
22 21 11
Ready to send data
Ready to receive client score = 2
```

What if he or she writes wrong packet seq, ack, length numbers.

Client side: wrong packet = true is (24 22 10)

```
C:\Users\DELL\PycharmProjects\AI\venv\Scripts\python.exe C:/Use
Do you want to play in auto mode or manual mode?
manual
Enter your message
selam
Enter your sequence, acknowledgement and packet length values:
10 12 14
Ready to send data
Ready to receive client score = 1
iyi 12 24 10 from ('127.0.0.2', 2897)
Enter your message
nabersin
Enter your sequence, acknowledgement and packet length values:
24 21 10
client score = 0
!!Corrupted packet!! try again, please more careful!!
Enter your message
```

Server side: wrong packet = true is (22 34 10)



Duplicate message error

1. Auto

Client side: Sending a message and a packet (10 14 15)

```
Do you want to play in auto mode or manual mode?
auto
Enter your message
hi
Enter your sequence, acknowledgement and packet length values:
10 14 15
Ready to send data
Ready to receive client score = 1
14 25 69 from ('127.0.0.2', 2897)
Enter your message
```

Server side: Received the message and the packet

```
hi 10 14 15 from ('127.0.0.2', 3997)
 Ready to send server score: 1
Correct
Ready to receive
```

Client side: Tried to send a packet as the player sent before. Therefore, a duplicate packet warning appeared.

```
Do you want to play in auto mode or manual mode?
auto
Enter your message
hi
Enter your sequence, acknowledgement and packet length values:
10 14 15
Ready to send data
Ready to receive client score = 1
14 25 69 from ('127.0.0.2', 2897)
Enter your message
hello
Enter your sequence, acknowledgement and packet length values:
10 14 15
client score = 0
This is a duplicate packet!
Enter your message
```

2. Manual

Client side: Sent a packet with values 13 14 15

```
Do you want to play in auto mode or manual mode?
manual
Enter your message
hi
Enter your sequence, acknowledgement and packet length values:
13 14 15
Ready to send data
Ready to receive client score = 1
```

Server side: Received the message and the packet from the client and sent a reply with a packet

```
Ready to receive
hi 13 14 15 from ('127.0.0.2', 3997)
 Ready to send server score: 0
Enter your reply
hello
Enter your calculated seq, ack, packet length numbers:
14 28 10
Ready to send data
Ready to receive server score = 1
```

Client side: Received the message and the packet from the server and sent a packet that was sent before (13 14 15). Duplicate packet warning message appeared. Client was asked to enter a message and a packet again.

```
Do you want to play in auto mode or manual mode?
manual
Enter your message
hi
Enter your sequence, acknowledgement and packet length values:
13 14 15
Ready to send data
Ready to receive client score = 1
hello 14 28 10 from ('127.0.0.2', 2897)
Enter your message
yes
Enter your sequence, acknowledgement and packet length values:
13 14 15
client score = 0
This is a duplicate packet!
Enter your message
```

Duplicate packet warning message

```
Do you want to play in auto mode or manual mode?
manual
Enter your message
hi
Enter your sequence, acknowledgement and packet length values:
13 14 15
Ready to send data
Ready to receive client score = 1
hello 14 28 10 from ('127.0.0.2', 2897)
Enter your message
yes
Enter your sequence, acknowledgement and packet length values:
13 14 15
client score = 0
This is a duplicate packet!
Enter your message
yes
Enter your sequence, acknowledgement and packet length values:
28 24 20
Ready to send data
Ready to receive client score = 1
```

Client sent a new message and a packet after the warning message

Server side: This time, server tries to send a packet that was sent before and again the duplicate packet message appeared.

```
Ready to receive
hi 13 14 15 from ('127.0.0.2', 3997)
 Ready to send server score: 0
Enter your reply
hello
Enter your calculated seq, ack, packet length numbers:
14 28 10
Ready to send data
Ready to receive server score = 1
yes 28 24 20 from ('127.0.0.2', 3997)
Enter your reply
no
Enter your calculated seq, ack, packet length numbers:
14 28 10
server score = 0
This is a duplicate packet!
Enter your reply
```

As it can be seen in each case when a duplicate message happens the score is decreased.

Timeout

The client sends a message and a packet (5 7 8). After the data is sent, the server waits for the other packet. However, if the client does not send any message or packet the server side gives a timeout warning and waits for the message or the packet again.

```
Do you want to play in auto mode or manual mode?
auto
Enter your message
hi
Enter your sequence, acknowledgement and packet length values:
5 7 8
Ready to send data
Ready to receive client score = 1
7 13 14 from ('127.0.0.2', 2897)
Enter your message
```

Client side

```
Ready to receive
hi 5 7 8 from ('127.0.0.2', 3997)
 Ready to send server score: 1
Correct
Ready to receive
Timeout!
Ready to receive
```

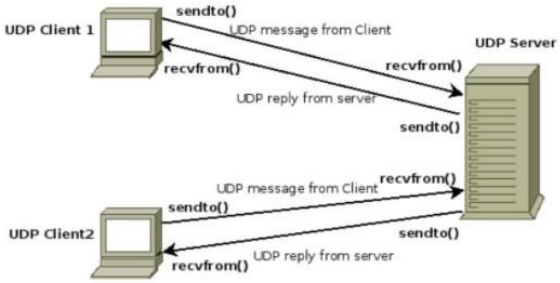
Server side

SLIDES

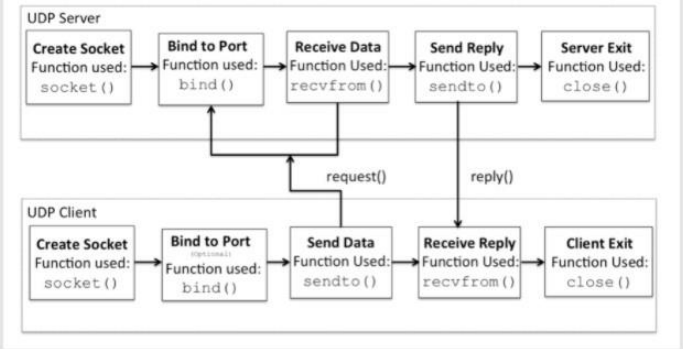
WHAT IS UDP?

- UDP is the abbreviation of User Datagram Protocol.
- In communications using UDP, a client program sends a message packet to a destination server.

- The UDP does not provide guaranteed delivery of message packets.
- If for some issue in a network if a packet is lost it could be lost forever.



SOCKET PROGRAMMING WITH UDP



UDP: no "connection" between client & server

- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- rcvr extracts sender IP address and port# from received packet

PSEUDO CODE SOCKET PROGRAMMING

Pseudocode UDP client

- Create Socket
- Loop
  - (Send Message To Well-known port of server) +
  - (Receive Message From Server)
- Close Socket

Pseudocode UDP server

- Create Socket
- Bind socket to a specific port where clients can contact you
- Loop
  - (Receive UDP Message from client x) +
  - (Send UDP Reply to client)
- Close Socket

AIM OF THE GAME

- The project is a gamification of a UDP-based client-server communication using sockets.
- The main purpose here is to examine the behavior of the UDP service.
- UDP does not provide guaranteed delivery of message packets.
- Transmitted data may be lost and received out of order.

RULES OF THE GAME

Auto Mode

- At the start of the game, the program asks the player if they want to play in auto mode or manual mode.
- If auto mode is chosen, the player (client) plays with a computer.
- The client sends a message with seq, ack and packet length values.
- The server side automatically calculates the new numbers and immediately sends back the reply.
- The client can detect duplicate and corrupted packets.

Manual Mode

- In manual mode, there are two players.
- One is a client and the other is a server, however in this case the server acts similarly to a client.
- The client starts the game by sending a message and the seq, ack and packet length numbers.
- When this message is received by the server, it means it is the Player 2's turn to answer.
- Player 2 answers the message first, and then calculates and sends the numbers.
- The client and the server can detect duplicate and corrupted packets.