Assignment 05

- Title: UDP Socket Program

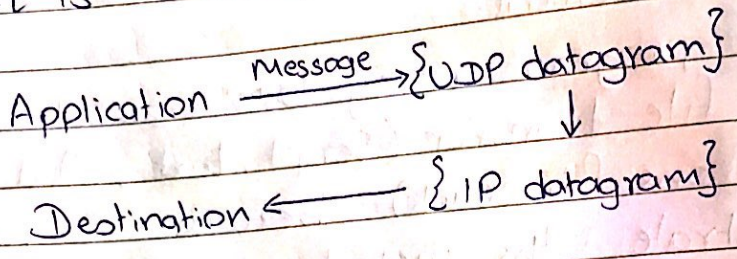
- Problem Statement:
Write a program using UDP sockets to enable file transfer (Script, text, Audio, Video one file each) between 2 machines. Demonstrate the packets captured traces using wireshark Packet analyzer Tool for peer to peer mode.

- Objectives:
- Getting familiar with client-server communication model.
- Designing simple client or server application for datagram.
- Learn important libraries and method classes (the UNIX and internet sockets) used for network programming.

- Software & Hardware Requirements:
- Java SE11, IDE
- Ubuntu/Windows
- intel i5 64bit
- i/o devices

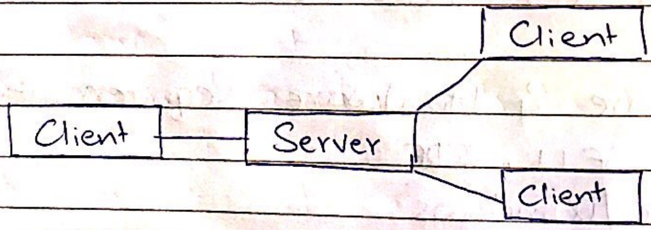
- Theory:

User Datagram Protocol:
It is a transport layer protocol



The problem of UDP is the lack of reliability if the message will reach it's destination, but it's good at checking errors of checksum & retransmitting if need be.
UDP provides a connection - less service because no connection is there between client & server.

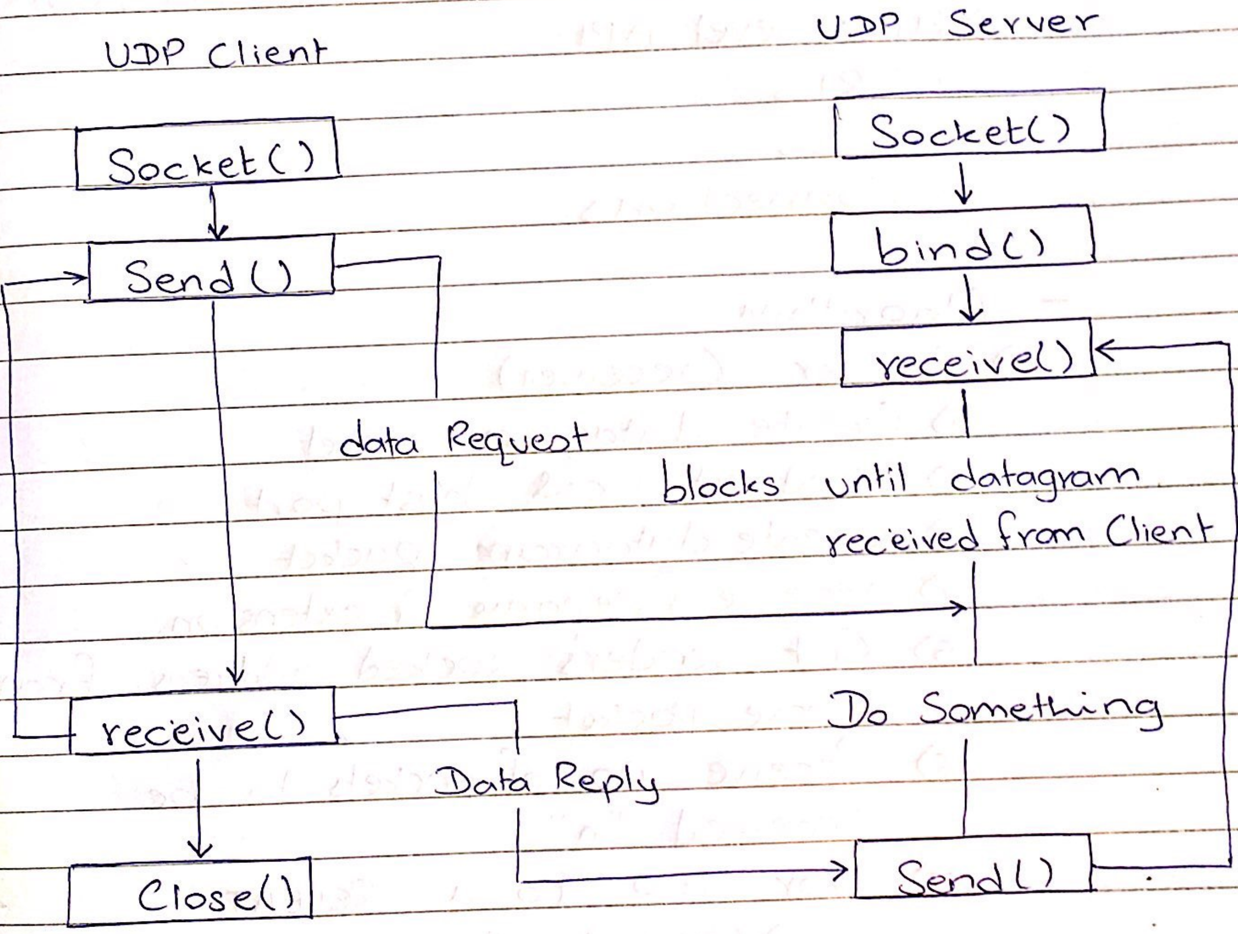
Client Server Model & UDP



ARCHITECTURE OF
CLIENT - SERVER

UDP client just sends a datagram to server using sendto() function Which requires destination address as parameter.

UDP server just calls receive() function it returns the IP address & port of Client & datagram.



- Java. net. Package()

Java.net package provides classes for implementing networking applications It can be divided into two sections.

1) Low level API
 - Address - IPV4 , IPV6
 - Sockets - TCP Client/Server API , UDP

2) High level API
 - URI's
 - URL's
 - Connections.

- Algorithm:

```
A) Server (receiver)
    1) Create datagram Socket
    2) Bind at local host port
    3) Create datagram packet
    4) receive filename & extension.
    5) Get sender's socked address from
       above packet
    6) Receive no. of packets to be
       received "n"
    7) for i=0 to n REPEAT
        1) Request ith packet
        2) Receive packet
        3) Write it to a file in sequence
    8) Send Completion acknowledgement
       to server.
```

B) Client (Sender)

```
1) Create a new datagram
2) Get InetAddress & Port & file path
   from User
3) Send filename to receiver
4) Open file in read mode
5) Calculate no. of packets to be sent (n)
6) Send "n"
7) Break file into packets & store
   in a sequenced data structure
8) While completion ack != received REPEAT
       1) Receive Packet sequence
       2) Send ith packet
9) Close socket
10) Print no. of bytes sent.
```

- Conclusion

In this assignment we studied client Server model & UDP & implemented it for file ~~d~~ transfer using fast Ethernet.

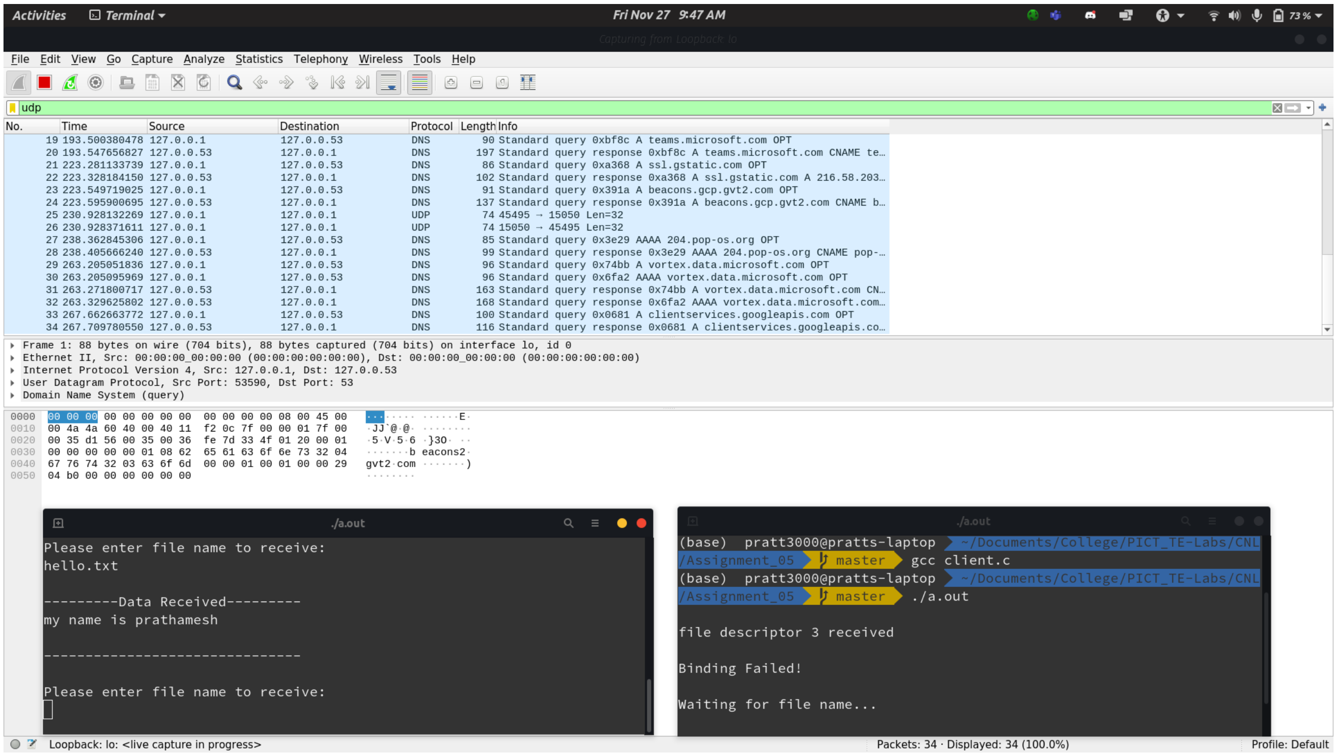
```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define IP_PROTOCOL 0
#define IP_ADDRESS "127.0.0.1" // localhost
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0

// funtion to clear buffer
void clearBuf(char* b)
{
    int i;
    for (i = 0; i < NET_BUF_SIZE; i++)
        b[i] = '\0';
}

// function for decryption
char Cipher(char ch)
{
    return ch ^ cipherKey;
}

// function to receive file
int recvFile(char* buf, int s)
{
    int i;
    char ch;
    for (i = 0; i < s; i++) {
        ch = buf[i];
        ch = Cipher(ch);
        if (ch == EOF)
            return 1;
        else
            printf("%c", ch);
    }
    return 0;
}

// driver code
int main()
```

```
{
    int sockfd, nBytes;
    struct sockaddr_in addr_con;
    int addrlen = sizeof(addr_con);
    addr_con.sin_family = AF_INET;
    addr_con.sin_port = htons(PORT_NO);
    addr_con.sin_addr.s_addr = inet_addr(IP_ADDRESS);
    char net_buf[NET_BUF_SIZE];
    FILE* fp;

    // socket()
    sockfd = socket(AF_INET, SOCK_DGRAM,
                    IP_PROTOCOL);

    if (sockfd < 0)
        printf("\nfile descriptor not received!!\n");
    else
        printf("\nfile descriptor %d received\n", sockfd);

    while (1) {
        printf("\nPlease enter file name to receive:\n");
        scanf("%s", net_buf);
        sendto(sockfd, net_buf, NET_BUF_SIZE,
               sendrecvflag, (struct sockaddr*)&addr_con,
               addrlen);

        printf("\n---------Data Received---------\n");

        while (1) {
            // receive
            clearBuf(net_buf);
            nBytes = recvfrom(sockfd, net_buf, NET_BUF_SIZE,
                              sendrecvflag, (struct sockaddr*)&addr_con,
                              &addrlen);

            // process
            if (recvFile(net_buf, NET_BUF_SIZE)) {
                break;
            }
        }
        printf("\n-------------------------------\n");
    }
    return 0;
}
```

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define IP_PROTOCOL 0
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0
#define nofile "File Not Found!"

// funtion to clear buffer
void clearBuf(char* b)
{
    int i;
    for (i = 0; i < NET_BUF_SIZE; i++)
        b[i] = '\0';
}

// funtion to encrypt
char Cipher(char ch)
{
    return ch ^ cipherKey;
}

// funtion sending file
int sendFile(FILE* fp, char* buf, int s)
{
    int i, len;
    if (fp == NULL) {
        strcpy(buf, nofile);
        len = strlen(nofile);
        buf[len] = EOF;
        for (i = 0; i <= len; i++)
            buf[i] = Cipher(buf[i]);
        return 1;
    }

    char ch, ch2;
    for (i = 0; i < s; i++) {
        ch = fgetc(fp);
        ch2 = Cipher(ch);
        buf[i] = ch2;
        if (ch == EOF)
```

```
            return 1;
        }
        return 0;
    }

    // driver code
    int main()
    {
        int sockfd, nBytes;
        struct sockaddr_in addr_con;
        int addrlen = sizeof(addr_con);
        addr_con.sin_family = AF_INET;
        addr_con.sin_port = htons(PORT_NO);
        addr_con.sin_addr.s_addr = INADDR_ANY;
        char net_buf[NET_BUF_SIZE];
        FILE* fp;

        // socket()
        sockfd = socket(AF_INET, SOCK_DGRAM, IP_PROTOCOL);

        if (sockfd < 0)
            printf("\nfile descriptor not received!!\n");
        else
            printf("\nfile descriptor %d received\n", sockfd);

        // bind()
        if (bind(sockfd, (struct sockaddr*)&addr_con, sizeof(addr_con)) == 0)
            printf("\nSuccessfully binded!\n");
        else
            printf("\nBinding Failed!\n");

        while (1) {
            printf("\nWaiting for file name...\n");

            // receive file name
            clearBuf(net_buf);

            nBytes = recvfrom(sockfd, net_buf,
                              NET_BUF_SIZE, sendrecvflag,
                              (struct sockaddr*)&addr_con, &addrlen);

            fp = fopen(net_buf, "r");
            printf("\nFile Name Received: %s\n", net_buf);
            if (fp == NULL)
                printf("\nFile open failed!\n");
            else
                printf("\nFile Successfully opened!\n");

            while (1) {
```

```
            // process
            if (sendFile(fp, net_buf, NET_BUF_SIZE)) {
                sendto(sockfd, net_buf, NET_BUF_SIZE,
                       sendrecvflag,
                    (struct sockaddr*)&addr_con, addrlen);
                break;
            }

            // send
            sendto(sockfd, net_buf, NET_BUF_SIZE,
                   sendrecvflag,
                (struct sockaddr*)&addr_con, addrlen);
            clearBuf(net_buf);
        }
        if (fp != NULL)
            fclose(fp);
    }
    return 0;
}
```



udp

```
No.  Time         Source      Destination  Protocol Length Info
  19 193.500380478 127.0.0.1  127.0.0.53   DNS       90 Standard query 0xbf8c A teams.microsoft.com OPT
  20 193.547656827 127.0.0.53 127.0.0.1    DNS      197 Standard query response 0xbf8c A teams.microsoft.com CNAME te…
  21 223.281133739 127.0.0.1  127.0.0.53   DNS       86 Standard query 0xa368 A ssl.gstatic.com OPT
  22 223.328184150 127.0.0.53 127.0.0.1    DNS      102 Standard query response 0xa368 A ssl.gstatic.com A 216.58.203…
  23 223.549719025 127.0.0.1  127.0.0.53   DNS       91 Standard query 0x391a A beacons.gcp.gvt2.com OPT
  24 223.595900695 127.0.0.53 127.0.0.1    DNS      137 Standard query response 0x391a A beacons.gcp.gvt2.com CNAME b…
  25 230.928132269 127.0.0.1  127.0.0.1    UDP       74 45495 → 15050 Len=32
  26 230.928371611 127.0.0.1  127.0.0.1    UDP       74 15050 → 45495 Len=32
  27 238.362845306 127.0.0.1  127.0.0.53   DNS       85 Standard query 0x3e29 AAAA 204.pop-os.org OPT
  28 238.405666240 127.0.0.53 127.0.0.1    DNS       99 Standard query response 0x3e29 AAAA 204.pop-os.org CNAME pop-…
  29 263.205051836 127.0.0.1  127.0.0.53   DNS       96 Standard query 0x74bb A vortex.data.microsoft.com OPT
  30 263.205095969 127.0.0.1  127.0.0.53   DNS       96 Standard query 0x6fa2 AAAA vortex.data.microsoft.com OPT
  31 263.271800717 127.0.0.53 127.0.0.1    DNS      163 Standard query response 0x74bb A vortex.data.microsoft.com CN…
  32 263.329625802 127.0.0.53 127.0.0.1    DNS      168 Standard query response 0x6fa2 AAAA vortex.data.microsoft.com…
  33 267.662663772 127.0.0.1  127.0.0.53   DNS      100 Standard query 0x0681 A clientservices.googleapis.com OPT
  34 267.709780550 127.0.0.53 127.0.0.1    DNS      116 Standard query response 0x0681 A clientservices.googleapis.co…
```

```
▸ Frame 1: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.53
▸ User Datagram Protocol, Src Port: 53590, Dst Port: 53
▸ Domain Name System (query)
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 4a 4a 60 40 00 40 11  f2 0c 7f 00 00 01 7f 00   ·JJ`@·@· ········
0020  00 35 d1 56 00 35 00 36  fe 7d 33 4f 01 20 00 01   ·5·V·5·6 ·}3O· ··
0030  00 00 00 00 00 01 08 62  65 61 63 6f 6e 73 32 04   ·······b eacons2·
0040  67 76 74 32 03 63 6f 6d  00 00 01 00 01 00 00 29   gvt2·com ·······)
0050  04 b0 00 00 00 00 00 00                            ········
```

./a.out

```
Please enter file name to receive:
hello.txt

---------Data Received---------
my name is prathamesh

-------------------------------

Please enter file name to receive:
```

./a.out

```
(base)  pratt3000@pratts-laptop  ~/Documents/College/PICT_TE-Labs/CNL
/Assignment_05  master  gcc client.c
(base)  pratt3000@pratts-laptop  ~/Documents/College/PICT_TE-Labs/CNL
/Assignment_05  master  ./a.out

file descriptor 3 received

Binding Failed!

Waiting for file name...
```

Loopback: lo: <live capture in progress>    Packets: 34 · Displayed: 34 (100.0%)    Profile: Default