Faculty of Engineering and Architectural Science

Department of Electrical and Computer Engineering

Course Number	COE 768
Course Title	Computer Networks
Semester/Year	F2020
Lab No	05
Instructor Name	Bobby Ma
Section No	05

Submission Date	10/29/2020
Due Date	10/29/2020

Name	Student ID	Signature*
Vatsal Shreekant	500771363	
Dimple Gamnani	500726015	

**By signing above, you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at:*

www.ryerson.ca/senate/current/pol60.pdf

Introduction

UDP is a transport protocol commonly used by network applications. Unlike TCP, applications based on UDP can send data without the establishment of transport-layer connection.

The socket system call indicates that the transport service is UDP (SOCK_DGRAM). It does not need to call *accept* for it does not need to deal with connection request. UDP server calls *recvfrom* to wait for data from the client.

Recvfrom is same as read with the exception that recvfrom also returns the address of the sender (the client) stored in the argument *fsin*. When the server needs to send data back to the client, it will call *sendto*.

The UDP client makes essentially the same system calls as TCP client to prepare the sock. However, in this case, connect does not trigger a TCP connection. Instead, it associates the socket to the destination address (stored in the argument sin). Because of this association, the client does not need to use *sendto* when it sends data to the server; instead it can use write.

Procedure

1) Load time_server.c and time_client.c example project and complete the instructions in the lab manual.

2) Modify the 'time_client.c' file to allow the user download multiple files. Consequently, the client program should provide a user interface such that the user can select to download file or quit the program.

3) The PDU should be defined as a PDU structure and the data should be sent in the Final PDU. To predetermine the file size, the *lstate* system call can be used.

4) The following figures implement the time_client.c and time_server.c files.

C:\Users\Owner\OneDrive - Ryerson University\4th Year\COE 718\Labs_new\Lab3\Demo_768.c

```
1   /* time_client.c - main */
2
3   #include <sys/types.h>
4
5   #include <unistd.h>
6   #include <stdlib.h>
7   #include <string.h>
8   #include <strings.h>
9   #include <stdio.h>
10  #include
    <sys/socket.h>
11  #include <netinet/in.h>
12  #include <arpa/inet.h>
13
14  #include <netdb.h>
15  #include <math.h>
16
17  #include <sys/stat.h>
18
19  #define BUFSIZE 64
20
21  #define MSG    "Any Message \n"
22
23
24  /*------------------------------------------------------------------------
25   * main - UDP client for TIME service that prints the resulting time
26   *------------------------------------------------------------------------
27   */
28   double filesize(FILE *fp){
29    int prev = ftell(fp);
30    fseek(fp, 0L, SEEK_END);
31    int sz = ftell(fp);
32    fseek(fp, prev, SEEK_SET);
33    return sz;
34    }
35
36  int main(int argc, char **argv)
37  {
38    char  *host = "localhost", *str1;
39    int port = 3000;
40    char  now[100];    /* 32-bit integer to hold time  */
41    char buffer[5];
42    FILE *file;
43    char *loc, str[25];
44    struct hostent  *phe; /* pointer to host information entry  */
45    struct sockaddr_in sin; /* an Internet endpoint address   */
46    struct pdu {
47      char type;
48      char data[100];
49      };
50    int s, n, type, final;  /* socket descriptor and socket type  */
51
52    switch (argc) {
53    case 1:
54      break;
55    case 2:
56      host = argv[1];
57    case 3:
58      host = argv[1];
59      port = atoi(argv[2]);
60      break;
61    default:
62      fprintf(stderr, "usage: UDPtime [host [port]]\n");
63      exit(1);
64    }
65
66    memset(&sin, 0, sizeof(sin));
67          sin.sin_family = AF_INET;
```

Page 1

Figure 1: Page 1 of time_client.c

```
68          sin.sin_port = htons(port);
69
70      /* Map host name to IP address, allowing for dotted decimal */
71          if ( phe = gethostbyname(host) ){
72                  memcpy(&sin.sin_addr, phe->h_addr, phe->h_length);
73          }
74          else if ( (sin.sin_addr.s_addr = inet_addr(host)) == INADDR_NONE )
75      fprintf(stderr, "Can't get host entry \n");
76
77      /* Allocate a socket */
78          s = socket(AF_INET, SOCK_DGRAM, 0);
79          if (s < 0)
80      fprintf(stderr, "Can't create socket \n");
81
82
83      /* Connect the socket */
84          if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
85      fprintf(stderr, "Can't connect to %s %s \n", host, "Time");
86
87
88    //(void) write(s, MSG, strlen(MSG));
89
90    /* Read the time */
91
92    struct pdu spud;
93    spud.type = 'C';
94    while(1){
95    printf("Enter a file name then click enter or enter 'Exit' to exit the program. \n");
96    scanf("%s", spud.data);
97    if(strstr(spud.data, "Exit") != NULL)exit(0);
98    write(s, &spud, strlen(spud.data)+1);
99    n = read(s, buffer, 26);
100   if ((n < 0)||(strstr(buffer, "Error") != NULL)){
101   if (n < 0){ fprintf(stderr, "Read failed\n");
102   printf("-------------------------------------------------------- \n");}
103   if(strstr(buffer, "Error") != NULL){ printf("Error file does not exist \n");
104   printf("-------------------------------------------------------- \n");}
105   }
106   else{
107
108   int size = atoi(buffer);
109   double group = size / 100.000;
110   int value = ceil(group);
111   if(value > 0) final = group*100 - (value-1)*100;
112   int j = 0;
113   loc = (char *) malloc(1000);
114   for(int i = 0; i < value;i++){
115     //printf("%d \n", j);
116     char temp[100];
117     int index = 0;
118     index = read(s, temp, 101);
119     if(j = 0) memmove(temp, temp+5, strlen(temp));
120     memmove(temp, temp+1, strlen(temp));
121     strcat(loc, temp);
122     j = j + index;
123     //bzero(temp, 100);
124   }
125   //strcpy(str1, loc);
126   file = fopen(spud.data, "w");
127   fwrite(loc, 1, size, file);
128   free(loc);
129   fclose(file);
130   printf("File transfer of '%s' has completed successfully \n", spud.data);
131   printf("-------------------------------------------------------- \n");
132     }
133   //write(1, now, n);
134   //exit(0);
135
```


Figure 2: Page 2 of time_client.c

C:\Users\Owner\OneDrive - Ryerson University\4th Year\COE 718\Labs_new\Lab3\Demo_768.c

```c
/* time_server.c - main */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>
#include <sys/stat.h>
#define length 100
/*------------------------------------------------------------------------
 * main - Iterative UDP server for TIME service
 *------------------------------------------------------------------------
 */

double filesize(FILE *fp){
  int prev = ftell(fp);
  fseek(fp, 0L, SEEK_END);
  int sz = ftell(fp);
  fseek(fp, prev, SEEK_SET);
  return sz;
  }
struct pdu{
  char type;
  char data[100];
};
int main(int argc, char *argv[])
{
  struct  sockaddr_in fsin; /* the from address of a client */
  char  buf[100], rbuf[5];    /* "input" buffer; any size > 0 */
  char    *pts,  *loc;
  int sock;     /* server socket    */
  time_t  now;      /* current time     */
  int alen;     /* from-address length    */
  struct  sockaddr_in sin; /* an Internet endpoint address         */
        int     s, type;        /* socket descriptor and socket type    */
        FILE *file;
  int   port=3000;
        int final;

  switch(argc){
    case 1:
      break;
    case 2:
      port = atoi(argv[1]);
      break;
    default:
      fprintf(stderr, "Usage: %s [port]\n", argv[0]);
      exit(1);
  }

        memset(&sin, 0, sizeof(sin));
        sin.sin_family = AF_INET;
        sin.sin_addr.s_addr = INADDR_ANY;
        sin.sin_port = htons(port);

    /* Allocate a socket */
        s = socket(AF_INET, SOCK_DGRAM, 0);
        if (s < 0)
    fprintf(stderr, "can't creat socket\n");

    /* Bind the socket */
```


Figure 3: Page 1 of time_server.c

C:\Users\Owner\OneDrive - Ryerson University\4th Year\COE 718\Labs_new\Lab3\Demo_768.c

```
69          if (bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
70        fprintf(stderr, "can't bind to %d port\n",port);
71          listen(s, 5);
72      alen = sizeof(fsin);
73
74      while (1) {
75        char str1[200];
76        memset(&buf, 0, sizeof(buf));
77        if (recvfrom(s, buf, sizeof(buf), 0,
78            (struct sockaddr *)&fsin, &alen) < 0)
79          fprintf(stderr, "recvfrom error\n");
80
81        strncpy(str1, buf+1, strlen(buf));
82        //printf("%s\n", buf);
83        file = fopen(str1, "r");
84        if(file != NULL){
85          struct pdu spudf;
86          spudf.type = 'F';
87          struct pdu spud;
88          spud.type = 'D';
89
90          int size = filesize(file);
91          sprintf(rbuf, "%d", size);
92
93          (void) sendto(s, rbuf, 5, 0, (struct sockaddr *)&fsin, sizeof(fsin));
94          double group = size / 100.00;
95          int value = ceil(group);
96          if(value > 0)final = group*100 - (value-1)*100;
97          int index = 0;
98          loc = (char *) malloc(size);
99          fread(loc, 1, size, file);
100         for(int i = 0;i < value; i++){
101           char temp[100];
102         //  spud.data[100];
103         if(i != value-1){
104           strncpy(spud.data, loc + index, size - index);
105
106           index += 100;
107
108             (void) sendto(s, &spud, 101, 0, (struct sockaddr *)&fsin, sizeof(fsin));
109         }
110         else{
111         strncpy(spudf.data, loc + index, size - index);
112           (void) sendto(s, &spudf, final+1, 0, (struct sockaddr *)&fsin, sizeof(fsin));
113           memset(spud.data, 0, sizeof(spud));
114           memset(spudf.data, 0, sizeof(spudf));
115
116           fclose(file);
117         }
118         }
119
120
121       }
122       else{
123       //(void) time(&now);
124             //pts = ctime(&now);
125             struct pdu spude;
126         spude.type = 'E';
127         strcpy(spude.data, "Error file does not exist");
128       (void) sendto(s, &spude, 26, 0,
129         (struct sockaddr *)&fsin, sizeof(fsin));
130       }
131       //(void) time(&now);
132             //pts = ctime(&now);
133
134       //(void) sendto(s, pts, strlen(pts), 0,
135       //  (struct sockaddr *)&fsin, sizeof(fsin));
136     }
```


Figure 4: Page 2 of time_server.c

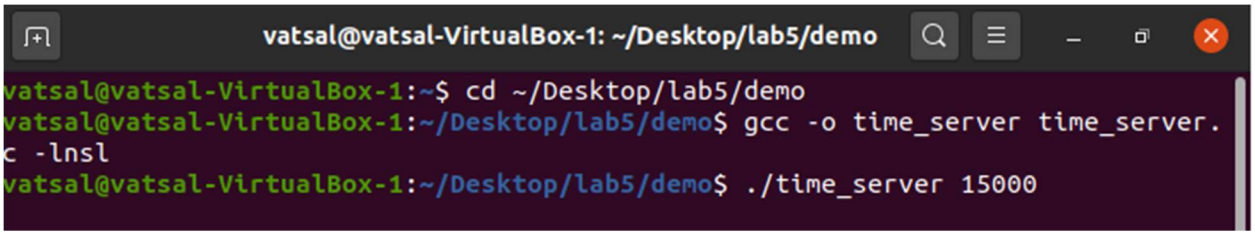

Figure 5: Server-side commands

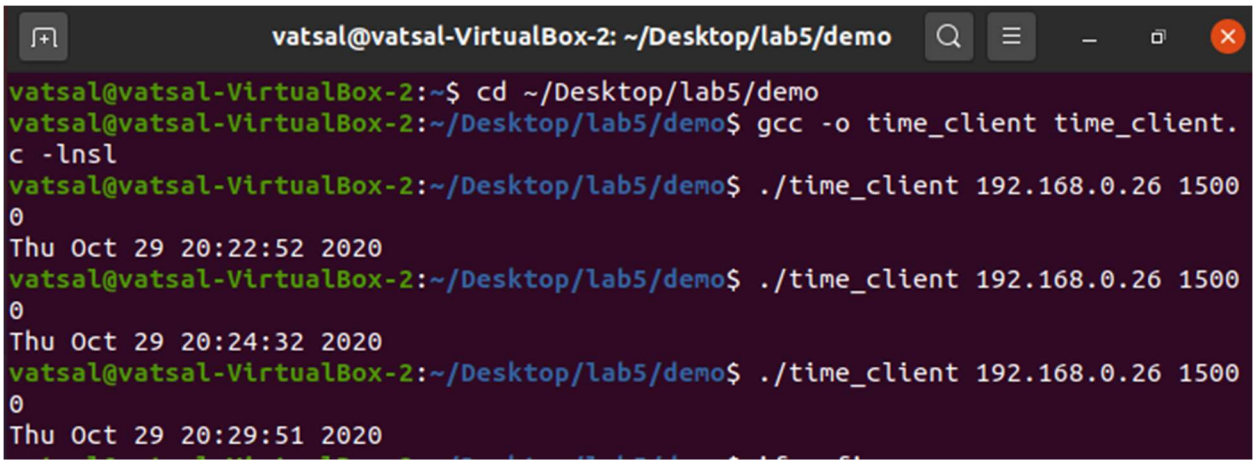

Figure 6: Client-side commands

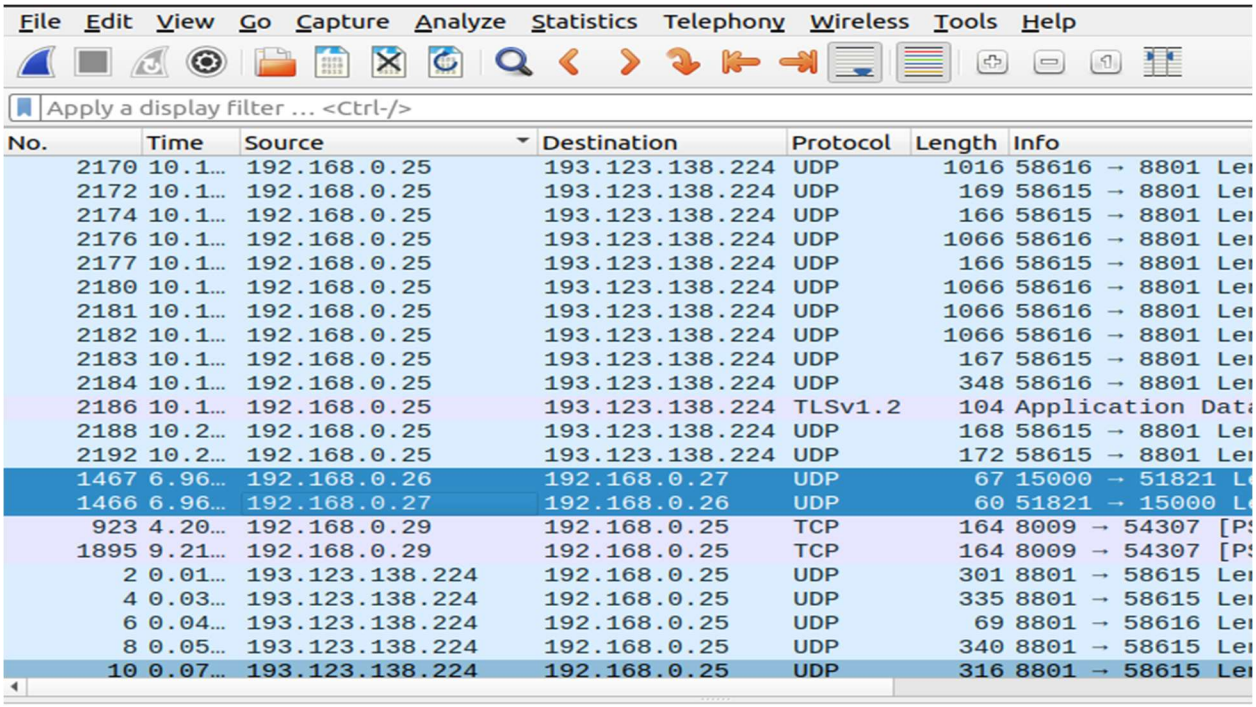

Figure 7: Wirehshark capture of the 2 UDP packets

Conclusion

The answers to the lab questions are listed below:

1. To demonstrate the completion of steps 1-6 of the lab manual, please refer to figures 5, 6 and 7. These figures show the Server/Client-side commands as well as the Wireshark capture of the 2 UDP packets.

2. The server uses the recvfrom()method to extract the client-side (source) port number from the segment it receives from the client, which is the message; it then sends a new segment to the client, with the extracted source port number serving as the destination port number in this new segment.

3. UDP protocol is more suitable for Time service because it provides fast and efficient transmission. UDP also allows to run multiple clients on one server and it is lightweight. There is no ordering of messages, no tracking connections. No handshaking.

4. Concurrency is more appropriate because concurrent server can serve multiple clients with at the same time (faster connection). Concurrency also allows to run multiple clients in parallel.

5. The programs have been demonstrated for both time_server as well as time_client as per figures 1, 2, 3 and 4.

6. The logic used to enable the server to transfer the complete file to the client is listed between lines 74-130, figure 4.

7. The logic used to enable the client to receive the complete file from the server is listed between lines 126-131, figure 2.

8. The client handles the error message through a function strstr(). It is a function that finds the first occurrence of the substring in the string. This is evident from line 103, figure 2.

References

1) Coonjah, I., Catherine, P. C., & Soyjaudah, M. S. (2015). Experimental performance comparison between TCP vs UDP tunnel using OpenVPN. Paper presented at the 1-5.

2) Gu, Y., & Grossman, R. L. (2007). UDT: UDP-based data transfer for high-speed wide area networks. Computer Networks (Amsterdam, Netherlands : 1999), 51(7), 1777-1799.