컴퓨터 네트워크 Term Project

201524527 이석준

201524582 정희석

TCP vs UDP 성능 비교

---

1. 목적: 컴퓨터 네트워크 시간에 배운 Transport Layer에서의 통신 프로토콜인 TCP와 UDP의 차이를 이해하고 실제 소켓프로그래밍을 통해 차이를 알아보는데 그 목적이 있다.

2. 배경 지식:

2.1 Transport Layer: OSI 7 layer model에서 4번째 계층, 네트워크 구성요소와 프로토콜 내에서 송/수신자를 연결하는 서비스를 제공하는 계층이다. 주로 사용되는 프로토콜은 TCP이며 멀티미디어와 같이 빠른 전송을 요구할 때는 UDP를 사용한다.

2.2. TCP: Transmission Control Protocol(전송 제어 프로토콜)

특징) Connection-Oriented Service를 기반으로 서버와 클라이언트 사이에 연결을 수립하고 패킷의 오류 및 네트워크 혼잡 제어, 서버-클라이언트 동기화를 통해 안정적인 데이터 교환이 가능하나 헤더 overhead가 크고 느리다는 단점이 있다.

2.3. UDP: User Datatgram Protocol(사용자 데이터그램 프로토콜)

특징) Connectionless Service를 기반으로 TCP와 다르게 서버와 클라이언트 사이에 사전 연결을 수립하지 않고 바로 데이터를 전달하는 방식이며 checksum 이외의 다른 제어 정보는 header에 포함하지 않으므로 헤더 overhead가 작고 빠른 데이터 교환이 가능하나 연결 수립 및 에러에 대해 약하고, 데이터가 손실되어도 feedback이 없다는 단점이 있다.

3. 서버 설정: Oracle VM VirtualBox, Ubuntu 16.04LTS버전

포워딩 목록

IP 주소	프로토콜	포트 범위	설명	선택
192.168.10.103	TCP+UDP	3389	Laptop remote cont	☐
192.168.10.101	TCP+UDP	25565	Minecraft Server	☐
192.168.10.101	TCP+UDP	25555	TCP_UDP_TEST	☐

삭제 모두 삭제 취소

서버 측 컴퓨터에서 공유기를 사용하고 있었으므로 공유기 설정으로 들어가서 위와 같이 Port Forwarding을 통해 공용 IP와 사설 IP를 연결하였다. 우리는 25555번 포트를 사용하였다.

포트 포워딩 규칙

이름	프로토콜	호스트 IP	호스트 포트	게스트 IP	게스트 포트
TCP_SERVER_TEST	TCP	192.168.10.101	25555	10.0.2.15	25555
UDP_SERVER_TEST	UDP	192.168.10.101	25555	10.0.2.15	25555

서버 측 컴퓨터에서 또한 Oracle VM VirtualBox를 사용하여 Ubuntu 16.04LTS 운영체제를 가상머신으로 작동, 서버를 열었기 때문에 가상머신에도 위와 같이 Port Forwarding이 필요했다.

DMZ 호스트

DMZ는 공유기 사용시 막혀 있는 포트들을 별도의 설정없이 손쉽게 개방해 주는 서비스 입니다.

☑ 사용

DMZ 호스트 IP주소: 192.168.10.101 자동 채우기 지우기

설정 적용 취소

위의 Port forwarding을 했으나 접근이 거부될 가능성을 없애기 위해 DMZ기능을 사용하여 모든 공용 IP를 데스크탑 컴퓨터로 연결시켰다.

그리고 방화벽에서 가상머신과 25555포트의 인바운드 규칙을 허용하여 연결할 수 있도록 하였다.

4. 소스코드:

>inet설정 파일<

<Server>

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

#define SERV_UDP_PORT 25555
#define SERV_TCP_PORT 25555
#define SERV_HOST_ADDR "127.0.0.1"
char *pname;
~
"inet.h" 12 lines, 258 characters
```

<Client>

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


#define SERV_UDP_PORT 25555
#define SERV_TCP_PORT 25555
#define SERV_HOST_ADDR "36.38.143.54"
char *pname;


void err_sys(const char *x);
void err_dump(const char *x);
"inet.h" 19 lines, 361 characters
```

서버에서는 HOST_ADDR은 사용하지 않으므로 localhost, UDP와 TCP포트 모드 25555포트를 사용, 클라이언트에서는 HOST_ADDR이 서버의 IP 36.38.143.54이므로 설정. 그리고 main에서 필요한 헤더파일을 선언해 놓았다.

4.1 TCP

4.1.1 TCP Server Code

```
#include "inet.h"
#include "echo.h"

void main(argc, argv)
        int argc;
        char *argv[];
{
        int socketFD, newsocketFD, clilen, childpid;
        struct sockaddr_in cli_addr, serv_addr;
        pname = argv[0];
        if((socketFD = socket(AF_INET, SOCK_STREAM, 0)) <0){
                printf("server : can't open stream socket\n");
                exit(-1);
        }
        bzero((char*) &serv_addr, sizeof(serv_addr));
        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        serv_addr.sin_port = htons (SERV_TCP_PORT);
        printf("Socket Opened\n");
        if(bind (socketFD, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) <0){
```

```
                printf("server : can't bind local address\n");
                exit(-1);
        }
        printf("waiting...\n");
        listen(socketFD, 5);
        for(;;){
                clilen = sizeof(cli_addr);
                printf("open client socket\n");
                newsocketFD = accept(socketFD, (struct sockaddr *)&cli_addr, &clilen);
                if(newsocketFD <0){
                        printf("server : accept error\n");
                        exit(-1);
                }
                if((childpid=fork())<0){
                        printf("server : fork error\n");
                        exit(-1);
                }
                else if(childpid == 0){
                        close (socketFD);
                        str_echo(newsocketFD);
                        printf("finish successfully\n");
                        exit(0);
                }
                close(newsocketFD);
        }
}

~
"TCP_Server.c" 47 lines, 1154 characters
```

위의 코드는 BSD Programing ppt의 예제 코드를 사용하였다.

1. 먼저 소켓을 열고(socketFD = socket(AF_INET, SOCK_STREAM,0))

2. 소켓과 서버 주소와 포트번호를 bind한다. (bind (SocketFD, (struct sockaddr *) &serv_addr, sizeof(serv_addr)))

3. listen을 통해 클라이언트를 5개까지 받을 수 있게 하고 소켓 설정을 저장.

4. for문(무한 loop)를 통해 클라이언트들의 접근을 기다리며 받을 때 마다 fork하여 echo함수를 동작, 작업이 완료되고 클라이언트가 접속을 끊으면 다시 대기상태로 돌아간다.

<Echo 함수>

```
#include <time.h>
#include "echo.h"
#include "readline.h"
void str_echo(sockfd)
        int sockfd;
{
        clock_t receive_timing, send_timing;
        int n;
        int count=0;
        char line[MAXLINE];

        for(;;){
                n = readline(sockfd, line, MAXLINE);
                receive_timing = clock();
                if(n == 0)
                        return;
                else if(n<0)
                        printf("str_echo : readline error");
                send_timing = clock();
                if(write(sockfd, line, n) != n)
                        printf("str_echo: writen error");
                printf("%d) Processing Time : %f \n",++count,(double) (send_timi
ng - receive_timing));
                //printf("RECEIVE : %d bytes \n %ld \n %s \n",n,CLOCKS_PER_SEC,l
ine);
        }
}
~
"echo.c" 25 lines, 598 characters
```

1. socket으로부터 MAXLINE(=4096)만큼의 데이터를 읽어들이고 그 길이를 n에 저장.

2. 수신 지점의 시간을 체크하고 발신 직전의 시간을 check하여 이를 processing time으로 한다.

3. socket으로 받은 데이터를 그대로 전송한다.

4. 처리 시간을 확인하여 출력한다.

4.1.2 TCP Client Code

```
#include "inet.h"
#include "strcli.h"
void err_sys(const char *x){
        perror(x);
        exit(0);
}
void err_dump(const char *x){
        printf("%s",x);
        exit(-1);
}
int main(argc,argv)
int argc;
char *argv[];
{
        int sockfd;
        struct sockaddr_in serv_addr;
        pname = argv[0];
        bzero((char *) &serv_addr, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
        serv_addr.sin_port = htons(SERV_TCP_PORT);
        printf("open stream socket\n");
        if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0)
                err_sys("client : can't open stream socket");
        printf("stream socket opened\n");
        printf("connect to server\n");
        if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
                err_sys("client : can't connect to server");
        printf("connected\n");
        str_cli(stdin,sockfd);
        close(sockfd);
        exit(0);
}
"TCPclient.c" 33 lines, 839 characters
```

위의 코드는 BSD Programing ppt의 예제 코드를 사용하였다.

1. 먼저 소켓을 열고(socketFD = socket(AF_INET, SOCK_STREAM,0))

2. 소켓과 접근할 서버 주소와 포트번호를 bind한다. (bind (SocketFD, (struct sockaddr *) &serv_addr, sizeof(serv_addr)))

3. connect를 통해 서버와 소켓을 연결한다.

4. str_cli를 통해서 stdin(standard input)으로 데이터를 보내게 된다.

Str_cli(echo함수)

```
#include "strcli.h"
#include "writen.h"
#include "readline.h"
void str_cli(fp, sockfd)
register FILE *fp;
register int sockfd;
{
        int n, sendCount = 0;
        clock_t sendTime = 0, receiveTime = 0;
        char sendline[MAXLINE], recvline[MAXLINE + 1];
        clock_t send_start, send_end,receive_end ;
        int sendThousand = 0;
        fgets(sendline, MAXLINE, fp);
        if (ferror(fp))
                err_sys("str_cli : error reading file");
        n = strlen(sendline);
        for (sendThousand; sendThousand < 1000 ; sendThousand++){
                send_start = clock();//
                if (writen(sockfd, sendline, n) != n )
                        err_sys("str_cli : written error on socket");
                send_end = clock();//
                n = readline(sockfd, recvline, MAXLINE);
                if (n < 0)
                        err_dump("str_cli : reading error");
                receive_end = clock();//
                recvline[n] = 0;
                sendTime += (send_end - send_start);
                receiveTime += (receive_end - send_end);
                //fputs(recvline,stdout);
                sendCount++;
        }
        sendTime = sendTime/1000.0;
        receiveTime = receiveTime/1000.0;
        printf("Average Send time is : %fms\n",(double)(sendTime/1000.0));
        printf("Average receive time is : %fms\n",(double)(receiveTime/1000.0));
        printf("sendCount : %d \n",sendCount);
}
"strcli.c" 37 lines, 1139 characters
```

1. fget으로 파일로부터 보낼 데이터를 받아서 저장한다.

2. 1000번 보내는 것으로 테스트하기 위해서 for문을 1000번 돈다.

3. 보낼 데이터를 writen을 통해 전송

4. 서버로부터 받은 데이터를 읽어들임

4.2 UDP

4.2.1 UDP Server Code

```
#include "inet.h"
#include "echo.h"

int main(argc,argv)
        int argc;
        char *argv[];
{
        int socketFD;
        struct sockaddr_in serv_addr, cli_addr;

        pname = argv[0];
        if((socketFD = socket(AF_INET, SOCK_DGRAM, 0)) < 0){
                printf("server : can't open datagram socket\n");
                exit(-1);
                }
        bzero((char *) &serv_addr, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        serv_addr.sin_port = htons(SERV_UDP_PORT);
        printf("Server opened\n");
        if(bind(socketFD, (struct sockaddr *) &serv_addr,sizeof(serv_addr))<0){
                printf("server : can't bind local address\n");
                exit(-1);
                }
        printf("Server Bind address successfully\n");
        dg_echo(socketFD, (struct sockaddr *) &cli_addr, sizeof(cli_addr));
}


"UDP_Server.c" 30 lines, 746 characters
```

위의 코드 역시 BSD Programming ppt의 예제 코드를 사용하였다.

1. 먼저 소켓을 열고(socketFD = socket(AF_INET, SOCK_STREAM,0))

2. 소켓과 서버 주소와 포트번호를 bind한다. (bind (SocketFD, (struct sockaddr *) &serv_addr, sizeof(serv_addr)))

3. 다른 연결 수립 과정 없이 echo함수를 바로 호출한다.

<echo 함수>

```
#include "echo.h"

void dg_echo(sockfd, pcli_addr, maxclilen)
        int sockfd;
        struct sockaddr * pcli_addr;
        int maxclilen;
{
        int n, clilen;
```

```
        int count = 0;
        char mesg[MAXMESG];
        clock_t receive_time, send_time;
        printf("echo function\n");
        for(;;){
                clilen = maxclilen;
                printf("waiting...\n");
                n = recvfrom(sockfd, mesg, MAXMESG, 0, pcli_addr, &clilen);
                receive_time = clock();
                if(n<0){printf("dg_echo : recvfrom error\n");exit(-1);}
                send_time = clock();
                if(sendto(sockfd, mesg, n, 0, pcli_addr,clilen) != n){
                        printf("dg_echo : sendto error\n");
                        exit(-1);
                }
                //printf("Receive %d Bytes \n > %s \n",n,mesg);
                printf("%d)Processing Time : %f \n",++count,(double)(send_time-r
eceive_time));
        }
}
```

위의 코드 역시 BSD Programming ppt의 예제 코드를 사용하였다.

1. 소켓으로부터 메시지를 대기한다. 이는 메시지가 들어올 때까지 대기한다.

2. 메시지가 들어오면 들어온 여부만 확인하고 다시 되돌려 보낸다.

4.2.2 UDP Client Code

```
#include "inet.h"
#include "dgcli.h"
void err_sys(const char *x){
        perror(x);
        exit(0);
}
void err_dump(const char *x){
        printf("%s",x);
        exit(-1);
}
int main(argc,argv)
int argc;
char *argv[];
{
        int sockfd;
        struct sockaddr_in serv_addr, cli_addr;
        pname = argv[0];
        bzero((char *) &serv_addr, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
        serv_addr.sin_port = htons(SERV_TCP_PORT);
        printf("open datagram socket\n");
        if ((sockfd = socket(AF_INET,SOCK_DGRAM,0)) < 0)
                err_sys("client : can't open datagram socket");
        printf("datagram socket opened\n");
        dg_cli(stdin, sockfd, (struct sockaddr *) &serv_addr,sizeof(serv_addr));
```

위의 코드 역시 BSD Programming ppt의 예제 코드를 사용하였다.

<echo 함수>

```
#include "dgcli.h"

void dg_cli(fp, sockfd,pserv_addr,servlen)
FILE *fp;
int sockfd;
struct sockaddr *pserv_addr;
int servlen;
{
        struct timeval timeout;
        int n,sendThousand = 0,sendCount = 0 ,timeouts, lost = 0;
        char sendline[MAXLINE], recvline[MAXLINE + 1];
        clock_t send_start, send_end, receive_end;
        clock_t sendTime, receiveTime;
        fgets(sendline, MAXLINE, fp);
        if (ferror(fp))
                err_sys("dg_cli : error reading file");
        n = strlen(sendline);
        fd_set rfds;
        for (sendThousand; sendThousand < 1000; sendThousand++){
                FD_ZERO(&rfds);
                FD_SET(sockfd,&rfds);
                timeout.tv_sec = 0;
                timeout.tv_usec = 76500;
                send_start = clock();
                if (sendto(sockfd, sendline, n,0, pserv_addr,servlen) != n )
                        err_dump("dg_cli : sendto error on socket");
                send_end = clock();
                if (timeouts = select(sockfd+1,&rfds,NULL,NULL,&timeout) == 0){
                        lost++;
                        printf("%dlost!\n",sendThousand);
                }
                else {
                        n = recvfrom(sockfd, recvline, MAXLINE, 0,
                                (struct sockaddr *) 0,(int *)0);
                        if (n < 0)
                                err_dump("dg_cli : recvfrom error");
                        receive_end = clock();
                        recvline[n] = 0;
                        //fputs(recvline,stdout);
                        sendTime += send_end - send_start;
                        receiveTime += receive_end - send_end;
                        sendCount++;
                }
        }
        sendTime = sendTime/(double)1000;
        receiveTime = receiveTime/(double)sendCount;
        printf("Average Send time is : %fms\n",(double)(sendTime/1000.0));
        printf("Average receive time is : %fms\n",(double)(receiveTime/1000.0));
        printf("sendCount : %d \n",sendCount);
        printf("lost : %d \n",lost);
}
```

해당 코드도 기본적인 틀은 BSD의 코드를 활용하였다.

UDP의 경우에는 timeout을 설정해 줄 필요가 있었다. 그래서 struct timeval이라는 구조체를 활용할 필요가 있다. 우리는 1000번의 메시지를 보내고, timeout을 판단하거나 잃어버린 메시지에 대한 변수 송 수신 시간을 측정하기 위한 변수 등

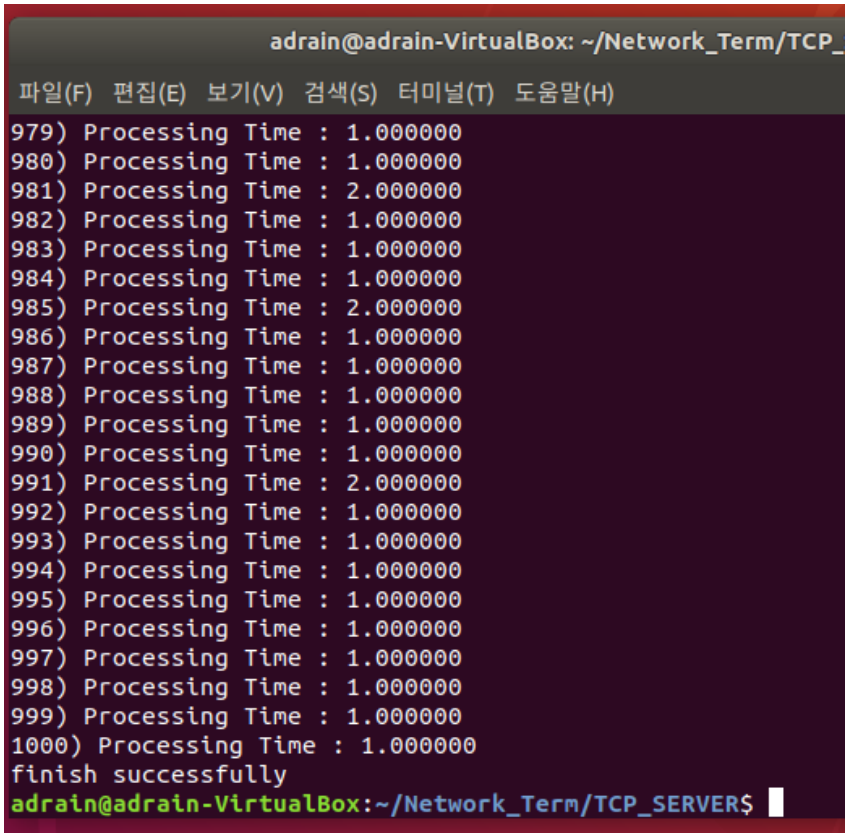
기본적인 여러 변수들을 설정하고, input file을 읽어와서 저장한다. 우리는 timeout을 판단해 줄 필요가 있기 때문에 select를 활용하기 위해서 fd_set을 사용한다. fd_set을 만들어주고 1000회의 메시지를 보내기 위해서 for문으로 들어간다. 해당 for문 안에서FD_ZERO를 통해 매번 아까 설정한 fr_set을 0으로 만든 다음에 우리가 읽을 file description을 FD_SET을 통해서 set 상태로 만들어준다. 또한 timeout도 설정해준다. 이 세 문장들이 반복 문 안에 선언된 이유는 select함수가 끝나면 수신 버퍼에 특정 값이 들어온 fd 이외에는 fd_set을 모두 0으로 만들어 버리기 때문이고 select함수가 호출되고 timeout의 발생까지 남아있던 시간을 다시 timeval 구조체에 담아버리기 때문이다. 이에 우리는 매 select함수 호출 전에 다시 설정을 해 줄 필요가 있다. 원래 기존 select는 여러 개의 통신에 대한 control이 목적인 함수이지만 이번에 우리는 select함수를 timeout을 위해서 사용하였다.

다음엔 sendto를 수행하기 전에 시간을 한번 측정하고 sendto함수 다음에 다시 한번 시간을 측정해서 보내는데 걸리는 시간을 계산하기 위한 변수를 담는다. 또 recvfrom함수를 수행하기 전에 수신 버퍼에 데이터가 timeout안에 수신되었는지 확인하기 위해서 select함수를 if문 안에 넣었다. 만약 select의 결과가 0이라면 timeout으로 lost된것이다. 그 경우 lost에 +1을 해줘서 총 잃어버린 데이터의 수를 파악한 후 몇 번째 자료가 lost되었는지를 출력한다. 만약 timeout이 발생하지 않았다면, recvfrom을 통해서 함수를 받아낸다. 함수의 다음부분에서 다시 시간을 측정해서 수신하는데 까지의 시간을 측정한다. 이후 아까 측정한 시간들을 통해서 송신, 수신하는데 걸린 시간을 측정해서 송신, 수신에 걸린 시간의 총 합을 계산한다. 이까지 실행이 되었다는 것은 통신이 잘 되었다는 뜻으로 sendcount에 +1을 해서 실제 통신이 성공한 횟수를 파악한다. 평균의 시간을 구하기 위해서 통신에 실패한 경우를 제외하고 실제 송수신에 성공한 수로 전체 총합을 나누면 평균 수신 시간을 파악할 수 있다. 송신은 수신과 상관없이 무조건적으로 보내기 때문에 총 보낸 횟수 1000으로 나눠준다. 출력 부분에서 1000으로 나누는 이유는 ms단위로출력하기 위해서이다. clock함수는 단위가 us단위이기 때문에 ms단위로 표시하기 위해서는 1000으로 나눠주어야 한다.

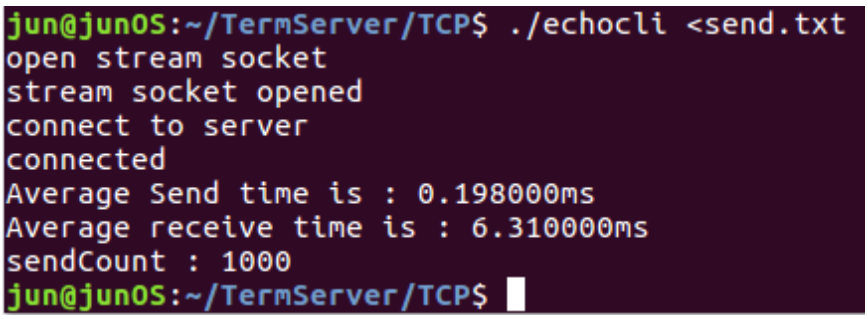
5. 실행 결과:

<2019/12/22 1:10AM TCP/UDP 실행>

서버(TCP):



클라이언트(TCP):



=> 클라이언트(아파트 네트워크)와 서버(빌라 네트워크)사이의 TCP통신 결과.

TCP의 안정성 보장을 통해 손실률이 0%인 것을 확인할 수 있었으며 평균 전송시간은 6.508ms였으며 서버측 처리시간은 1us로 거의 없는 수준으로 간주해도 될 정도였다. (4KB 파일, 1000회 기준)

서버(UDP):

```
990)Processing Time : 1.000000
waiting...
991)Processing Time : 1.000000
waiting...
992)Processing Time : 1.000000
waiting...
993)Processing Time : 1.000000
waiting...
994)Processing Time : 1.000000
waiting...
995)Processing Time : 1.000000
waiting...
996)Processing Time : 1.000000
waiting...
997)Processing Time : 1.000000
waiting...
998)Processing Time : 1.000000
waiting...
999)Processing Time : 1.000000
waiting...
1000)Processing Time : 0.000000
waiting...
```

클라이언트(UDP):

```
jun@junOS:~/TermServer/UDP$ ./echocli <send.txt
Average Send time is : 0.249000ms
Average receive time is : 0.058000ms
sendCount : 997
lost : 3
jun@junOS:~/TermServer/UDP$ ./echocli <send.txt
Average Send time is : 0.235000ms
Average receive time is : 0.056000ms
sendCount : 998
lost : 2
jun@junOS:~/TermServer/UDP$
```

=> 클라이언트(아파트 네트워크)와 서버(빌라 네트워크)사이의 TCP통신 결과.

UDP는 TCP와 다르게 사전 연결 수립하지 않고 안정성 보장이 떨어지는 대신 속도가 빠르다는 장점이 있었다. 전송 속도를 보면 307us, 291us의 빠른 속도와 일정한 속도를 보여주고 있다. 그러나 새벽시간에 데이터 이동이 적음에도 불구하고 2개에서 3개의 패킷이 손실되는 것을 확인할 수 있었다. 위의 결과를 보면 서버 측에서는 모두 수신하였으나 ACK(Echo)가 클라이언트 측으로 전달되지 않고 손실된 것을 확인할 수 있었다.

<2019/12/22 10:30AM TCP/UDP 실행>

서버(TCP):

```
 990) Processing Time : 1.000000
RECEIVE : 4000 bytes
 991) Processing Time : 1.000000
RECEIVE : 4000 bytes
 992) Processing Time : 1.000000
RECEIVE : 4000 bytes
 993) Processing Time : 1.000000
RECEIVE : 4000 bytes
 994) Processing Time : 0.000000
RECEIVE : 4000 bytes
 995) Processing Time : 1.000000
RECEIVE : 4000 bytes
 996) Processing Time : 1.000000
RECEIVE : 4000 bytes
 997) Processing Time : 1.000000
RECEIVE : 4000 bytes
 998) Processing Time : 1.000000
RECEIVE : 4000 bytes
 999) Processing Time : 1.000000
RECEIVE : 4000 bytes
 1000) Processing Time : 1.000000
RECEIVE : 4000 bytes
 finish successfully
```

클라이언트(TCP):

```
jun@ubuntu:~/TermServer/TCP$ ./echocli <send.txt
open stream socket
stream socket opened
connect to server
connected
Average Send time is : 0.218000ms
Average receive time is : 5.808000ms
sendCount : 1000
jun@ubuntu:~/TermServer/TCP$
```

=> 클라이언트(아파트 네트워크)와 서버(빌라 네트워크)사이의 TCP통신 결과.

새벽의 네트워크 전달과 비슷하게 TCP의 안정성 보장을 통해 손실률이 0%인 것을 확인할 수 있었으며 평균 전송시간은 6.026ms였으며 서버측 처리시간은 1us로 거의 없는 수준으로 간주해도 될 정도였다. (4KB 파일, 1000회 기준)

서버(UDP):

```
997)Processing Time : 1.000000
4000 Byte received
waiting...
998)Processing Time : 2.000000
4000 Byte received
waiting...
999)Processing Time : 1.000000
4000 Byte received
waiting...
1000)Processing Time : 1.000000
4000 Byte received
waiting...
```

클라이언트(UDP):

```
jun@ubuntu:~/TermServer/UDP$ ./echocli <send.txt
open datagram socket
datagram socket opened
Average Send time is : 0.202000ms
Average receive time is : 0.043000ms
sendCount : 1000
lost : 0
jun@ubuntu:~/TermServer/UDP$
```

=> 클라이언트(아파트 네트워크)와 서버(빌라 네트워크)사이의 TCP통신 결과.

위의 결과는 진행한 테스트 중 손실이 없었던 경우를 캡쳐 한 결과이며 대부분 2~3 많게는 10개이상 까지도 손실이 발생했다. 10개 이상 손실이 발생한 경우 확인해보니 서버 측에서 3~4개 미 수신, 클라이언트 측에서 ACK를 5~6개 미 수신과 같은 결과가 나타났다. 속도는 TCP에 비해 상당히 빠른 속도를 확인할 수 있었다. 그리고 위와 같은 결과를 위해서 여러 테스트를 한 결과 최적의 time out값이 7.6ms로 했을 때 위의 무손실 결과가 나왔었다. 물론 이후 여러 차례 데이터 교환을 하였으나 무손실의 결과는 나오지 않았었다는 한계점이 있다.

<2019/12/22 15:30PM TCP/UDP 실행>

서버(TCP):

```
RECEIVE : 4000 bytes
 995) Processing Time : 2.000000
RECEIVE : 4000 bytes
 996) Processing Time : 1.000000
RECEIVE : 4000 bytes
 997) Processing Time : 2.000000
RECEIVE : 4000 bytes
 998) Processing Time : 1.000000
RECEIVE : 4000 bytes
 999) Processing Time : 2.000000
RECEIVE : 4000 bytes
 1000) Processing Time : 2.000000
RECEIVE : 4000 bytes
 finish successfully
```

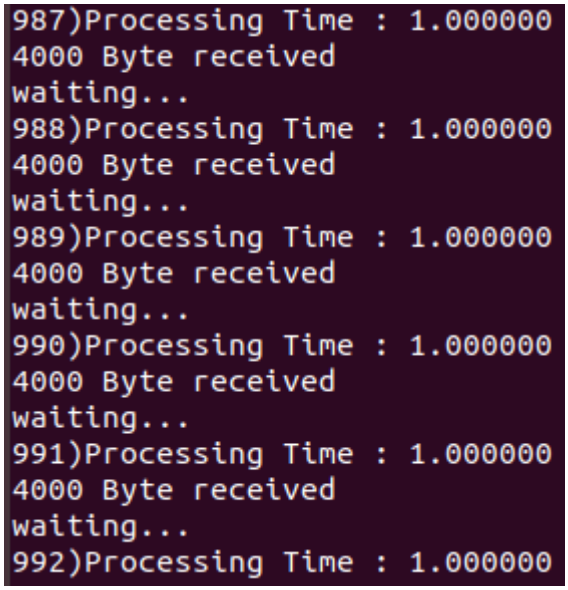
클라이언트(TCP):

```
jun@ubuntu:~/TermServer/TCP$ ./echocli <send.txt
open stream socket
stream socket opened
connect to server
connected
Average Send time is : 0.076000ms
Average receive time is : 2.360000ms
sendCount : 1000
jun@ubuntu:~/TermServer/TCP$
```

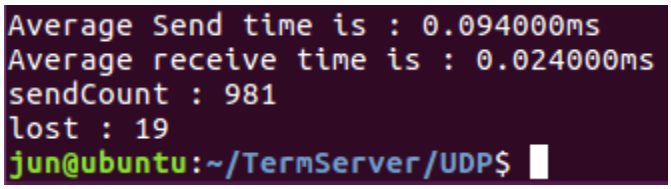
=> 클라이언트(모바일 핫스팟)와 서버(빌라 네트워크)사이의 TCP통신 결과.

앞의 결과들과 비슷하게 무손실, 느린 통신속도를 보여줬다. 하지만 이전의 아파트 네트워크보다 더 빠르게 데이터를 전송할 수 있었다. 계속 TCP와 UDP를 테스트했을 때 TCP는 항상 전송 & 수신율이 100%로 안정적인 경향을 보여주었으며 대신 속도는 UDP에 비해 많이 느린 것 역시 확인 가능했다.

서버(UDP):



클라이언트(UDP)(Time out 7.65ms):



=> 클라이언트(모바일 핫스팟)와 서버(빌라 네트워크)사이의 TCP통신 결과.

위의 결과는 모바일 네트워크와 유선 네트워크의 차이를 볼 수 있는 실험 결과였다. 기존의 아파트(집)에서의 UDP통신은 손실이 간헐적으로 발생하긴 했으나 1~2개정도의 적은 양의 데이터가 손실되었지만 모바일 네트워크를 통한 UDP통신에서는 데이터 손실률이 급격하게 높아지는 것을 알 수 있다. 위의 결과는 전송 당시 8개의 데이터가 서버에 도착하지 않은 것이고 11개의 데이터가 서버로부터 클라이언트에 도착하지 않은 것이다. 위의 결과가 가장 나았던 결과로 Time out을 기존의 76000(7.6ms)에서 21개의 lost, 90000(9ms)에서 25개, 100000(10ms)에서 16개 등 뒤죽박죽의 결과를 확인할 수 있었다(너무 많아서 이 결과는 보고서에 추가하지 않았다). 이를 통해 UDP는 통신환경에 영향을 많이 받고 속도는 빠르나 안정성 보장의 측면에서 약간 떨어지는 것을 확인할 수 있었고, 반면에 TCP는 안정성이 보장되지만 UDP의 10배 가까이(혹은 이상)의 속도차이가 나는 것을 확인할 수 있었다.

TCP 평균 전송시간: 6.508ms(오전 1시)+6.0188ms(오전 10시)+2.436ms(오후 3시)

UDP 평균 전송시간: 0.348ms(오전 1시)+0.245ms(오전 10시)+0.118ms(오후 3시)

TCP 손실률: 0%(전부 송/수신)

UDP 손실률: 3/1000(오전 1시)+0/1000(오전 10시)+91/1000(오후 3시) = 94/3000 = 3.13% 손실

위의 통계 결과는 3번의 대표적인 결과를 토대로 작성하였으며 실제는 이보다 전송시간은 느리고 손실률은 크지는 않지만 작지도 않다고 할 수 있다. 그리고 밤 시간대 보다 낮 시간대에 속도가 빨랐다는 것은 모바일 핫 스팟이 빠른 것인지, 아파트 네트워크가 안 좋은 것인지는 더 많은 상황의 시뮬레이션을 통해 알 수 있을 것 같다. 그리고 학교 네트워크에서 서버로 접근할 때 TCP는 100% 송/수신 완료하였으나 UDP는 1000회 실행 시 6203-1의 창의마루 네트워크에서는 100% 손실, PNU-WIFI를 통해서는 30%이상의 손실률을 확인할 수 있었다.

6. 어려웠던 점:

서버(정희석): 처음에 서버를 위해 공유기 포트 포워딩과 방화벽 인바운드 규칙, 가상머신 설정까지 다 하고 실행해서 서버를 올렸을 때 동작하지 않은 것으로 4~6시간 소요하여 힘들었다. 이는 간단하게 해결이 되어서 당황스러웠는데 가상머신인 VMBox의 개인 네트워크 접근을 비허용한 것이 문제였다.

클라이언트(이석준): TCP환경에서의 통신은 비교적 어렵지 않았다. PPT에 대부분 내용이 있었으므로 코드를 이해하고 작성하면 되었는데 UDP의 timeout을 설정하는 부분에서 많이 어려움을 느꼈다. select함수의 사용법부터, 해당 함수를 어느 위치에서 실행해야 하는지를 파악하거나 실행한 후 어떻게 해야 하는지에 대한 판단도 많이 생각해야 했던 부분이었다. 가장 어려웠던 부분은 select함수를 이해하고 사용하려 했는데 전체 함수를 실행한 뒤 계속 timeout으로 설정한 시간 이 지나버리면 이후는 모든 데이터가 lost된 것이 되고 함수가 끝나는 것이었다. 파악해본 결과 코드의 설명부분에도 설명했지만 select함수 이후 fd_set이 reset되는 부분이나 timeout값이 바뀌는 부분 때문에 그런 문제가 발생한 것이 아닐까 생각한다.

7. 느낀 점:

이석준(클라이언트): 지난 학기에 안드로이드 텀 프로젝트를 통해서 서버-클라이언트 통신을 수립하고자 했는데, 포트포워딩 문제로 실패한 경험이 있어 이번 프로젝트에 대해선 기대도 많았지만 걱정도 많았다. 아니나 다를까 이번에도 포트포워딩 문제로 많은 어려움을 겪었지만 해결하고 연결이 수립된 순간 너무 기분이 좋았다. 또한 실제로 통신을 수립하는 코드를 작성하고 결과가 나오는 실험을 해 봄으로써 마치 진짜 프로그래밍 개발자가 된 것만 같은 기분도 들어서 재미있는 프로젝트였다고 생각한다. 또한 TCP환경과 UDP환경 그리고 어떤 네트워크를 사용하는가에 따라서 같은 시간대에서도 결과가 다르고 다른 시간대에서도 물론 다양한 결과가 나와서 여러 환경에서 테스트해보는 것도 재미가 있었다.

정희석(서버): 항상 하려고 시도했다가 실패한 서버-클라이언트 연결 수립, 거기에 서버 소켓을 멀티 쓰레드로 많은 클라이언트들이 접근할 수 있는 서버를 만들어 보고 동작을 확인할 수 있어서 좋았다. 특히 이번 텀 프로젝트를 수행하기 전에 친구와 마인크래프트 라는 게임의 Java 가상 머신 기반 도메인 서버를 열어 봄으로써 포트 포워딩과 같은 기본 설정을 해봤던 경험이 이번 프로젝트에 도움이 되었다. 그리고 서버를 열 때 방화벽 규칙설정과 해당 프로그램이 방화벽에 의해 들어와야 되는 연결이 차단되고 있는 것은 아닌지 확인해 보라는 교훈을 얻을 수 있었다.