컴퓨터 네트워크 텀 프로젝트 과제 보고서

TCP 와 UDP 의 성능 비교를 위한 소켓프로그래밍

과목 : 컴퓨터 네트워크

교수님 : 유 영환 교수님

분반 : 060

제출일 : 2019-12-23

00. 목차

201524461 박 성국

201724557 장 수현

01. 프로젝트 주제 및 목적

네트워크 통신 시 전송 계층 프로토콜로 TCP 를 사용할 경우와 UDP 를 사용할 경우에 대해, echo server 와 client 를 구현하는 소켓프로그래밍을 통해 패킷 왕복 시간과 에러율의 관점에서 성능을 비교해본다.

02. 프로젝트 사전 지식 및 결과 예상

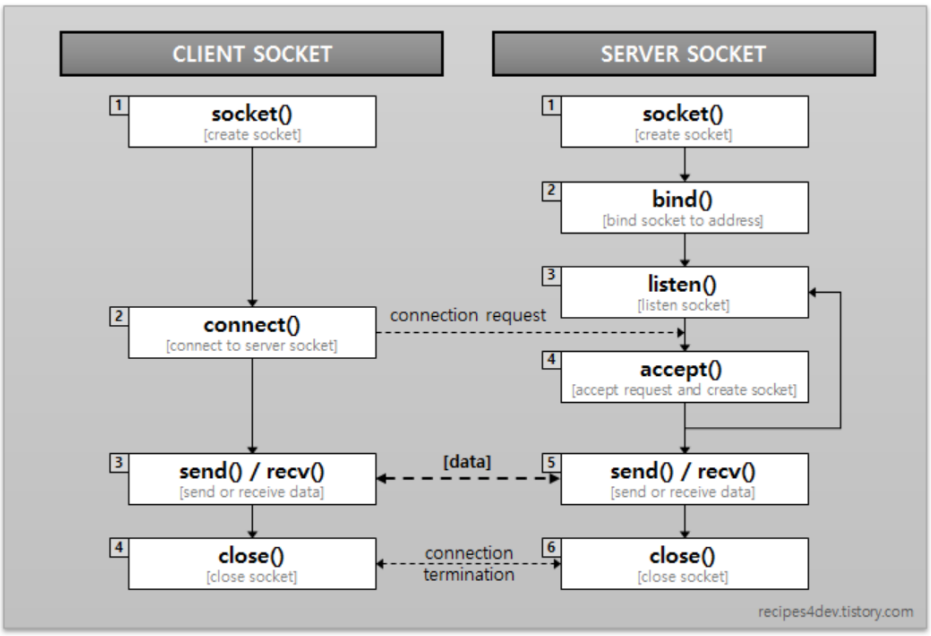
TCP 와 UDP 는 전송계층의 프로토콜로써 각자 다른 특징을 가지고 있다. 먼저 TCP 부터 말해보자면 TCP 는 신뢰성있는 전송이 특징이다. 여기서 신뢰성이란 데이터를 보냈을 때 전송 중 데이터의 손실이 없음을 의미한다. 신뢰성있는 전송이 가능한 이유는 TCP 프로토콜을 사용하는 전송계층에서 바로 위의 응용계층에 패킷을 올려보내기 전에 에러가 존재한다면 해결될때까지 재전송을 요청하기 때문이다. 때문에 전송에 걸리는 시간은 UDP 보다 비교적 오래걸리지만 결과적으로 데이터의 손실은 거의 없다고 볼 수 있다.

반면 UDP 는 비교적 가벼운 프로토콜이다. TCP 와 다르게 에러가 존재하더라도 재전송을 요청하지않고 그대로 응용계층에 넘겨준다. 물론 비트에러를 체크할 수는 있지만 이를 바탕으로 추가적인 처리를 하는 것은 프로그래머의 몫이며, UDP 프로토콜에서는 에러를 Detecting 만 한다. 때문에 신뢰성은 낮지만, TCP 에 비해 속도가 빠르다. 이런 장단점으로 인해 TCP 는 주로 메일 같은 신뢰성이 필요한 전송에 사용하고, UDP 는 멀티미디어 데이터처럼 Loss 에 Tolerant 한 전송에 사용한다.

이번 프로젝트에서 테스트해볼 내용은 이 TCP 와 UDP 프로토콜을 사용하여 성능을 비교해보는것이다. 각각의 프로토콜을 사용하여 실제로 전송을 해보고, 에러율과 전송시간이라는 두가지 지표를 바탕으로 두 프로토콜을 비교한다. 하루 중 서로 다른 세가지 시간대에 4KB 크기의 패킷을 1000 번 전송하여 걸린 전송시간과 발생한 에러의 횟수를 체크함으로써 두 프로토콜의 성능을 비교해 볼것이다.

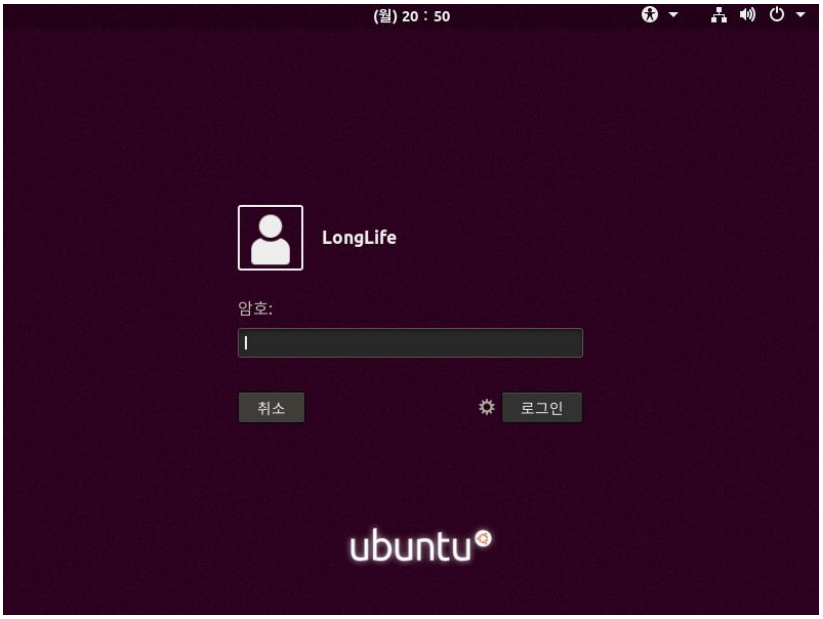
사전지식을 바탕으로 결과를 예상해보자면 TCP 를 사용하는 프로토콜의 경우에는 UDP 에 비교하여 시간이 오래걸릴것이다. 반면 에러는 이론적으로는 하나도 없을것이라고 예상하는데, 실제로 정말 단 하나의 에러도 없는것인지는 확신할 수 없다고 생각한다.

UDP 의 경우에는 TCP 보다 빠른 속도를 보여줄것이다. 하지만 에러가 발생한 횟수가 TCP 보다는 분명히 많을것인데, 기술이 발달한 요즘에도 과연 얼마나 에러가 많이 발생할것인지는 직접 측정을 해 보아야 할 것같다.

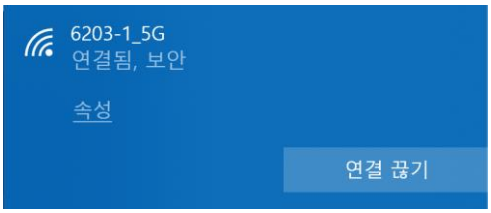


03. 프로젝트 준비

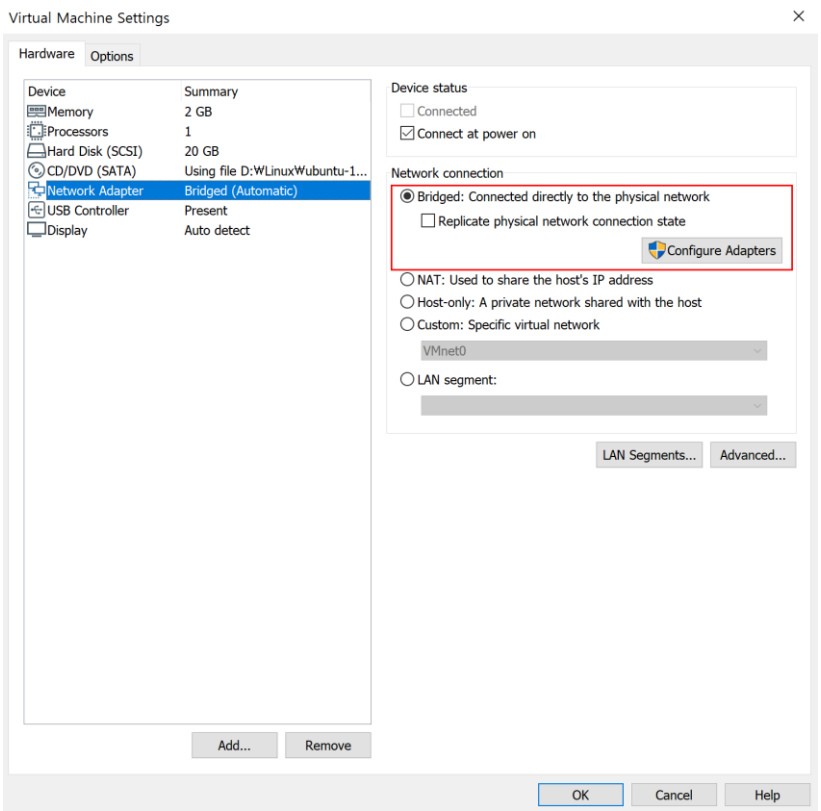
3.1 서버 환경



운영체제 - Windows 에 설치된 가상 머신(VMware)을 통한 Ubuntu Linux

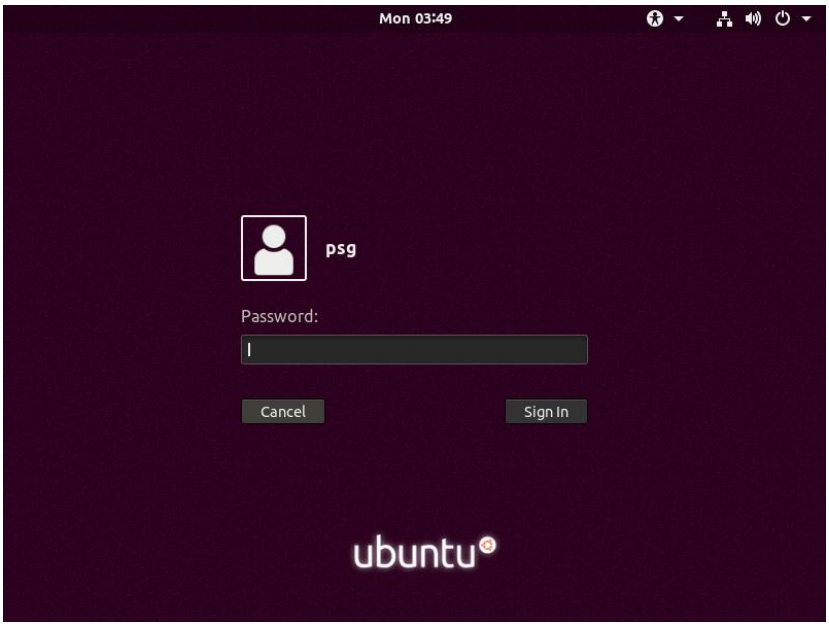


네트워크 - 부산대학교 컴퓨터공학관 창의마루에 설치된 공유기.

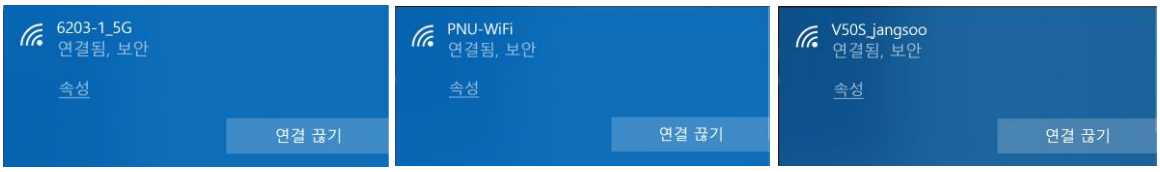


서버의 경우 네트워크 연결 방식을 Bridged 로 설정한다.

3.2 클라이언트 환경

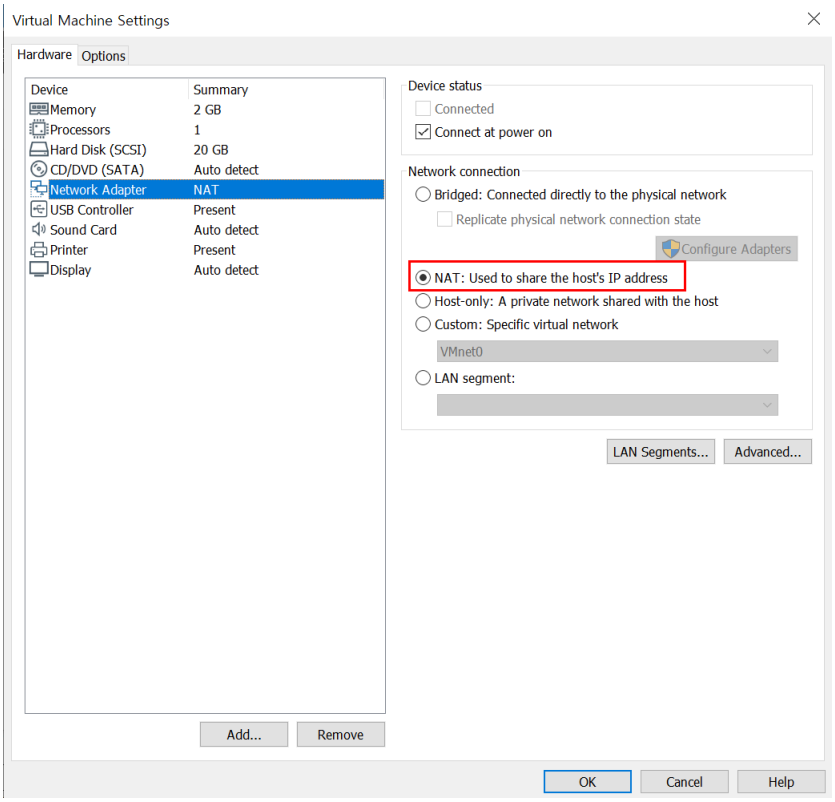


운영체제 - Windows 에 설치된 가상 머신(VMware)을 통한 Ubuntu Linux.



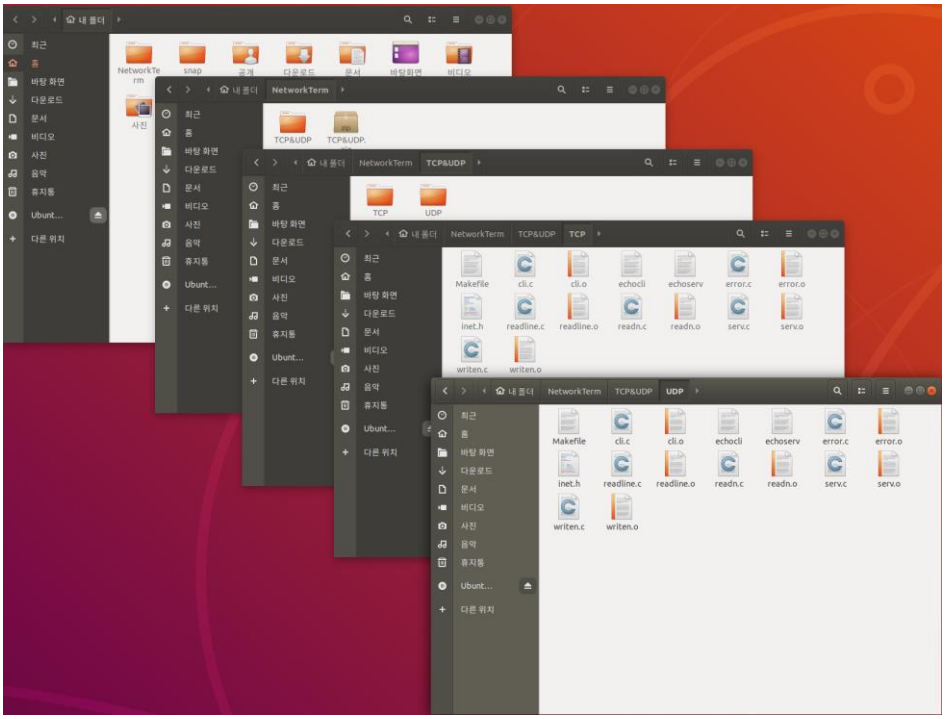
네트워크

- 서버와 같은 환경 : 부산대학교 컴퓨터공학관 창의마루에 설치된 공유기
- 서버와 다른 환경 : PNU-WiFi , V50S_jangsoo(개인 핫스팟)



클라이언트의 경우 네트워크 연결 방식을 NAT 로 설정한다.

3.3 파일 다운로드

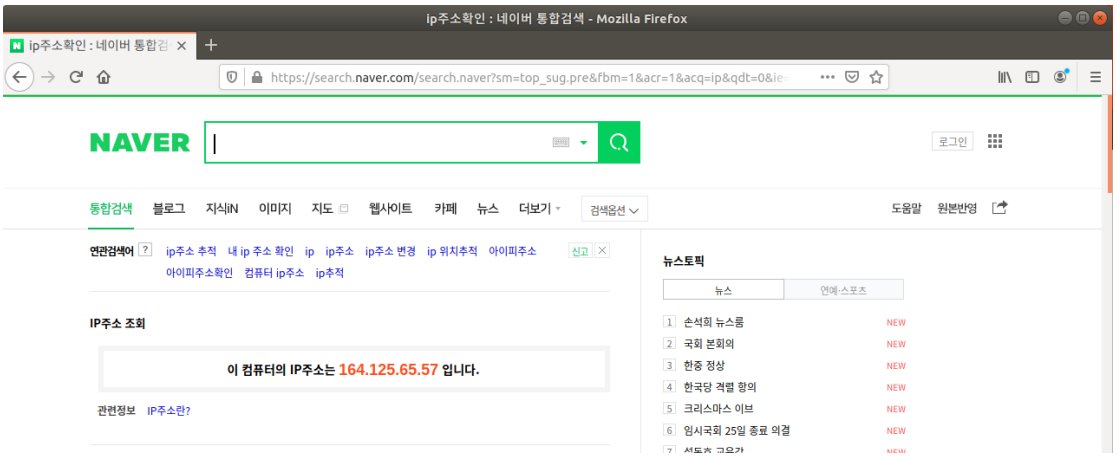


Windows 에서 drag&drop 을 하거나 Firefox 를 통해 Linux 에 필요한 파일들을 다운로드 한다.

04. 프로젝트 구현

4.1 서버 주소 확인

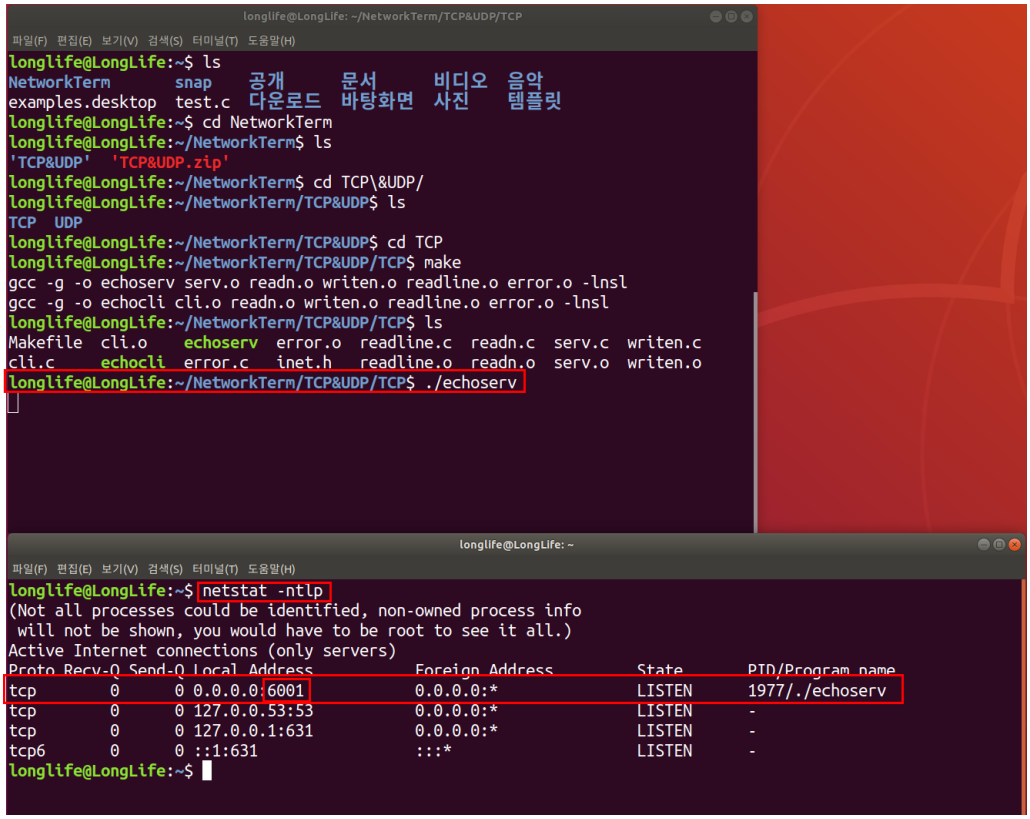
Bridged 모드를 사용하고 있으므로 Linux 의 서버 주소를 알아야한다. 그래서 Linux 의 웹 브라우저를 통해 서버의 IP 주소를 확인한다. 확인한 IP 주소는 164.125.65.57 이다.



4.2 포트 포워딩

4.1.1 TCP 서버 포트번호 확인

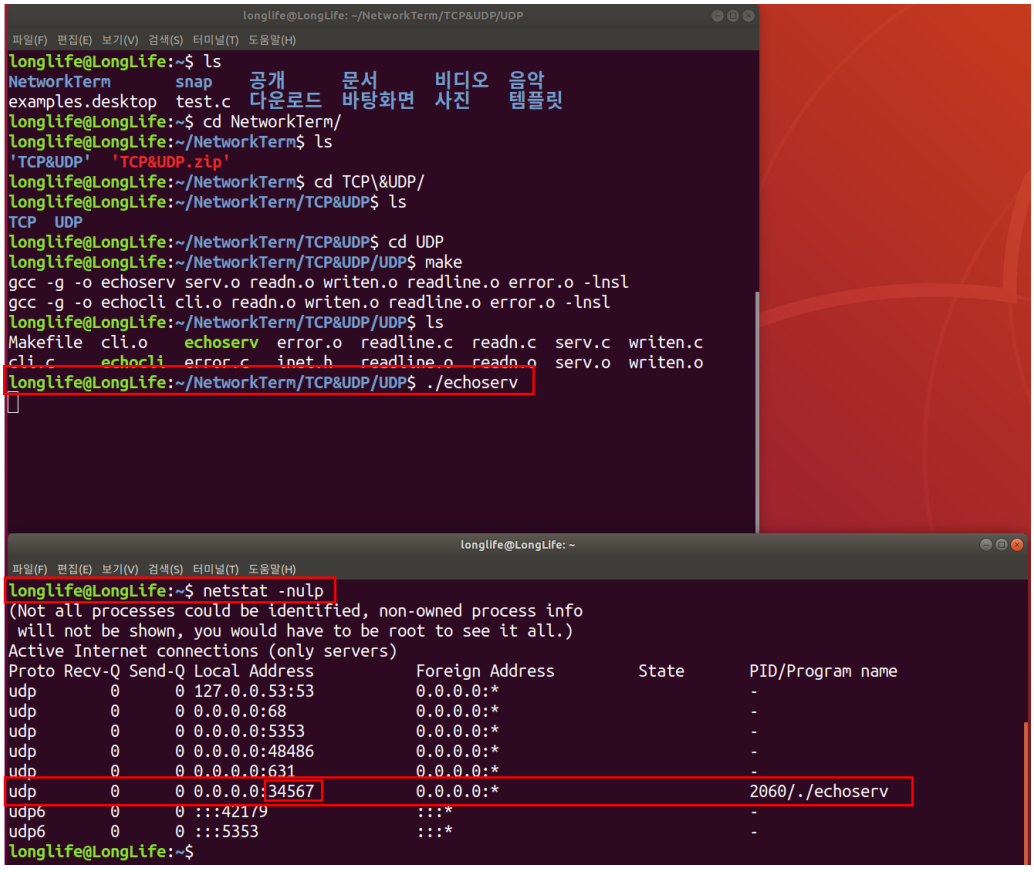
TCP 의 server 를 실행시킨 후 또 다른 터미널을 띄워 netstat -ntlp 명령어를 통해 서버의 포트번호를 확인한다. 이때, ntlp 명령어의 t 는 TCP 를 의미한다.



확인한 TCP 서버의 포트번호는 6001 이었다.

4.1.2 UDP 서버 포트번호 확인

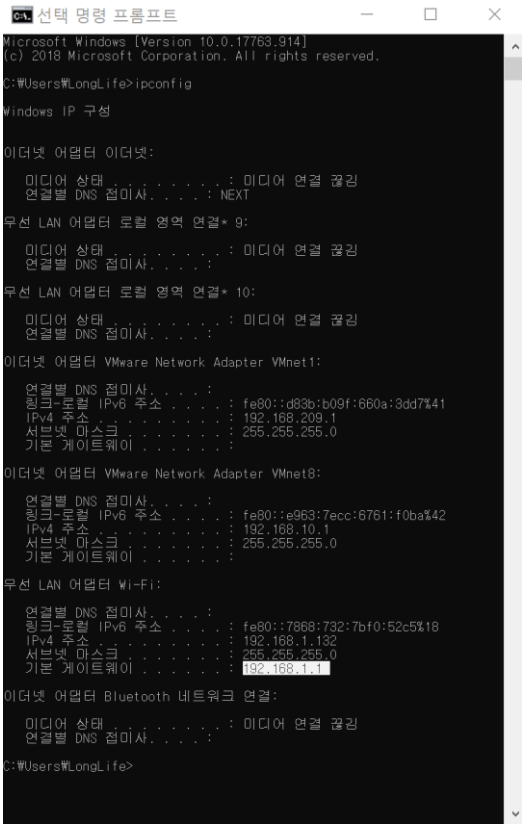
UDP 의 server 를 실행시킨 후 또 다른 터미널을 띄워 netstat -nulp 명령어를 통해 서버의 포트번호를 확인한다. 이때, nulp 명령어의 t 는 UDP 를 의미한다.



확인한 UDP 서버의 포트번호는 34567 이었다.

4.1.3 포트 포워딩

Windows 의 cmd 를 열어 ipconfig 명령어를 통해 현재 연결된 네트워크의 기본 게이트웨이 주소를 확인한다.



확인한 주소를 웹 브라우저에서 검색하면 다음과 같이 라우터(공유기)의 정보 및 설정 창을 확인할 수 있다.

Linux 에서 확인한 TCP 와 UDP 의 서버 포트번호를 사진과 같이 등록하여 포트 포워딩을 한다.

* 학과사무실에 양해를 구해 관리자 계정에 접근할 수 있었다.

4.2클라이언트의 inet.h

4.1 에서 구한 서버의 주소와 포트번호를 삽입해준다.

```
#ifndef TCPEXAMPLE_INET_H
#define TCPEXAMPLE_INET_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>  // To use memset (not bzero)

#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>  // To use fork()

#define SERV_UDP_PORT 34567
#define SERV_TCP_PORT 6001
#define SERV_HOST_ADDR "164.125.65.57"
#define MAXLINE 4096

char *pname;

#endif //TCPEXAMPLE_INET_H
```

4.3 TCP

4.3.1 TCP 서버

```
// Example of server using TCP protocol.

#include <ctype.h>
#include "inet.h"
void str_echo(int sockfd);
void ConvertLowerToUpper(char* line, int length);

int main(int argc, char *argv[]) {
  int sockfd;
```

```
int newsockfd;
int clilen;
int childpid;
struct sockaddr_in cli_addr;
struct sockaddr_in serv_addr;

pname = argv[0];

// Open a TCP socket (an Internet stream socket)
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
  err_dump("server : can't open stream socket");
}

// Bind our local address so that the client can send us
memset((char *) &serv_addr, 0, 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);

if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
  err_dump("server : can't bind local address");
}

listen(sockfd, 5); // connection 요청을 기다린다.
for (; ;) { // 요청이 들어왔을 경우
  // Wait for a connection from a client process.
  // This is an example of a concurrent server.

  clilen = sizeof(cli_addr);
  newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
  if (newsockfd < 0) {
    err_dump("server : accept error");
  }

  if ((childpid = fork()) < 0) {
    err_dump("server : fork error");
  } else if (childpid == 0) {  // 정상적으로 포크가 되었을 경우 받은 메시지를 다시 되돌려준다
    close(sockfd);
    str_echo(newsockfd);
```

```
    exit(0);
   }
   close(newsockfd);
  }
}

//
// Reads a stream socket one line at a time, and write each
// line back to the sender.
// Return when the connection is terminated.
//
void str_echo(int sockfd) {
  int n;
  char line[MAXLINE];
  char* convertedLine = "From Server: ";

  for (; ;) { // 계속 입력을 기다리며 입력이 들어오면 그대로 되돌려준다(echo).
   n = readline(sockfd, line, MAXLINE);

   if (n == 0) {
    return;  // connection terminated
   } else if (n < 0) {
    err_dump("str_echo : readline error");
   }

   //printf("Received string from Client: %s", line);
   ConvertLowerToUpper(line, strlen(line));
   if (write(sockfd, line, n) != n) {
    err_dump("str_echo : writen error");
   }
  }
}

void ConvertLowerToUpper(char* line, int length) {
  int i = 0;
  for (i = 0; i < length; ++i) {
   line[i] = toupper(line[i]);
  }
}
```

4.3.2 TCP 클라이언트

```
// Example of client using TCP protocol.

#include "inet.h"
#include <time.h>

void str_cli(register FILE *fp, register int sockfd);

int main(int argc, char *argv[]) {
 int sockfd;
 struct sockaddr_in serv_addr;
 pname = argv[0];
 //
 // Fill in the structure "serv_addr" with the address of
 // the server that we want to connect with
 //

 memset((char *) &serv_addr, 0, 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);

 //
 // Open a TCP socket (an Internet stream socket)
 //
 if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
  err_dump("client : can't open stream socket");
 }

 //
 // Connect the server
 //
 if (connect(sockfd, (  // TCP 는 Connection-Oriented Protocol 이므로 서버와 연결을 수립한다.
    struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
            perror("connect fail: ");
  err_dump("client : can't connect to server");
 }
```

```
 str_cli(stdin, sockfd);
 close(sockfd);

 return EXIT_SUCCESS;
}

void str_cli(register FILE *fp, register int sockfd) {
 int n;

 int write_err_count=0;
 int read_err_count=0;
/* 4KB 의 메시지를 만든다 */
 char sendline[MAXLINE] =
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAA";
 char recvline[MAXLINE + 1];
 clock_t start, end;

  start = clock();  // 시간 측정을 시작한다.

  for(int i=0;i<1000;++i) {  // 패킷전송 1000 번 반복
    n = 4096;

    if (writen(sockfd, sendline, n) != n) {
      write_err_count++;  // 쓰기 에러 카운트
      //err_dump("str_cli : written error on socket");
    }

    n = readline(sockfd, recvline, MAXLINE);
    recvline[n] = '₩0';

    if (n < 0) {
      read_err_count++;  // 읽기 에러 카운트
      //err_dump("str_cli : reading error");
    }
```

```
  }
  end = clock();   // 시간측정 끝

  float res = (float)(end-start)/CLOCKS_PER_SEC;
  char s1[20];
  sprintf(s1, "%f₩n", res);
  fputs(s1, stdout);
    fprintf(stdout,"W_err_cnt:  %d₩nR_err_cnt:  %d₩nTotal_err_cnt:  %d₩n",  write_err_count,
read_err_count, write_err_count+read_err_count);

 if (ferror(fp)) {
  err_dump("str_cli : error reading file");
 }
}
```

4.4 UDP

4.4.1 UDP 서버// UDP 는 Connectionless 이므로 Connect 과정이 없다.

```
/* Example of server using UDP protocol. */

#include "inet.h"
#include <ctype.h>
#include <sys/time.h>

void dg_echo(int sockfd, struct sockaddr* pcli_addr, int maxclilen);

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


   pname = argv[0];

   /*
    * Open a UDP socket (an Internet datagram socket).
    */

   if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
```

```
    err_dump("server : can't open datagram socket");

  //struct timeval optVal = {5, 0};
  //int optLen = sizeof(optVal);

  //state = setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, &optVal, optLen);
  //state = setsockopt(sockfd, SOL_SOCKET, SO_SNDTIMEO, &optVal, optLen);
  /*
   * Bind our local address so that the client send to us.
   */

  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);

  if(bind(sockfd, (struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0) // 지금부터 내가 이
포트를 쓸거라는 바인드만 한다.
    err_dump("server : can't bind local address");

  dg_echo(sockfd, (struct sockaddr *) &cli_addr, sizeof(cli_addr));

  /* NOT REACHED */

}

  /*
   * Read a datagram from a connectionless socket and write * it back to the sender.
   * We never return, as we never know when datagram client is * done.
   */

#define MAXMESG 2048

void dg_echo(int sockfd, struct sockaddr* pcli_addr, int maxclilen) {
  int n, clilen;
  char mesg[MAXMESG];
```

```
  for(;;) {  // 받은 메시지를 에코해줌.
    clilen = maxclilen;
    n = recvfrom (sockfd, mesg, MAXMESG, 0, pcli_addr, &clilen);
    if(n < 0)
      err_dump("dg_echo : recvfrom error");
    if(sendto(sockfd, mesg, n, 0, pcli_addr, clilen) != n)
      err_dump("dg_echo : sendto error");
  }
}
```

4.4.2 UDP 클라이언트 // 마찬가지로 ConnectionLess 이므로 연결수립 X

```
/* Example of client using UDP protocol. */

#include "inet.h"
#include <time.h>
#include <sys/time.h> // add 1

void dg_cli(FILE* fp, int sockfd, struct sockaddr* pserv_addr, int servlen);

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

  pname = argv[0];

  /*
   * Fill in the structure "serv_addr" with the address of * the server that we want to send to.
   */

  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_UDP_PORT);

  /*
   * Open a UDP socket (an Internet stream socket). */

  if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
```

```
    err_dump("client : can't open datagram socket");
  /* add from here */
  struct timeval optVal = {0, 6};         // Receive Timeout 을 설정해준다.
  int optLen = sizeof(optVal);            // 이때 Timeout 시간은 TCP 의 1 회 평균 전송시간인
  int state = setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, &optVal, optLen);  // 6ms 를 기준으로한다.
  /* add until here */
  dg_cli (stdin, sockfd, (struct sockaddr *) &serv_addr,sizeof(serv_addr));
  close (sockfd);
  exit(0);
}

/*

* Read the contents of the FILE *fp, write each line to * the data gram socket, then read a line back from the * datagram socket and write it to the standard output. * Return to caller when an EOF is encountered on the * input file.

*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>

void dg_cli(FILE* fp, int sockfd, struct sockaddr* pserv_addr, int servlen) {
  int n;
    char sendline[MAXLINE + 1] =
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
/* 4KB 의 패킷 설정*/
  char recvline[MAXLINE + 1];
  clock_t start, end;
  int write_err_count=0;
  int read_err_count=0;


/* 패킷을 1000 번 전송한다 */
```

```
  for(int i=0;i<1000;++i) {
    n = 4096;

    if( sendto (sockfd, sendline, n, 0, pserv_addr, servlen) != n)
          write_err_count++; //쓰기 에러 카운트
      //err_dump("dg_cli : sendto error on socket");

    n = recvfrom(sockfd, recvline, MAXLINE, 0,(struct sockaddr *) 0, (int *) 0);
    recvline[n] = 0;

    if(n < 0)
          read_err_count++;   //읽기 에러 카운트
      //err_dump("dg_cli : recvfrom error");
  }
  end = clock();


  /* null terminate */
  //fputs(recvline,stdout);

    fprintf(stdout,"W_err_cnt:  %d₩nR_err_cnt:  %d₩nTotal_err_cnt:  %d₩n",  write_err_count,
read_err_count, write_err_count+read_err_count ); // 화면에 결과 출력

  float res = (float)(end-start)/CLOCKS_PER_SEC;
  char s1[20];
  sprintf(s1, "%f₩n", res);
  fputs(s1,stdout);

  if(ferror(fp))
    err_dump("dg_cli : error reading file");
}
```

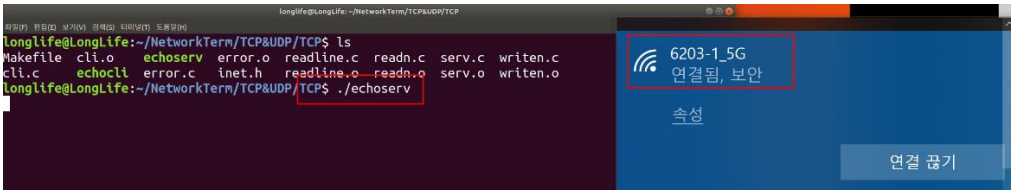
05. 프로젝트 결과

5.1 시행 1 차

일시 : 12 월 23 일 월요일 00:00 ~ 01:00

5.1.1 TCP 통신

서버측 실행 화면. 창의마루 공유기를 사용 중이다.

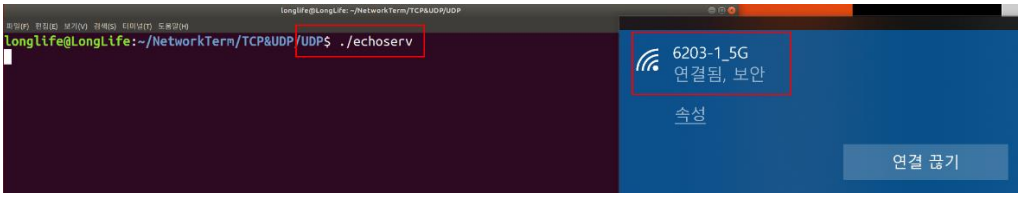


클라이언트 측 실행화면. V50S_jangsoo 을 사용 중이다.

4Kbytes 의 데이터가 1,000 번 왕복하는 데에 6.004752 의 시간이 걸렸다. 그리고 패킷은 에러가 난 것이 0 으로 아무것도 없었다. 즉, 에러율 0%를 보여주었다.

5.1.2 UDP 통신

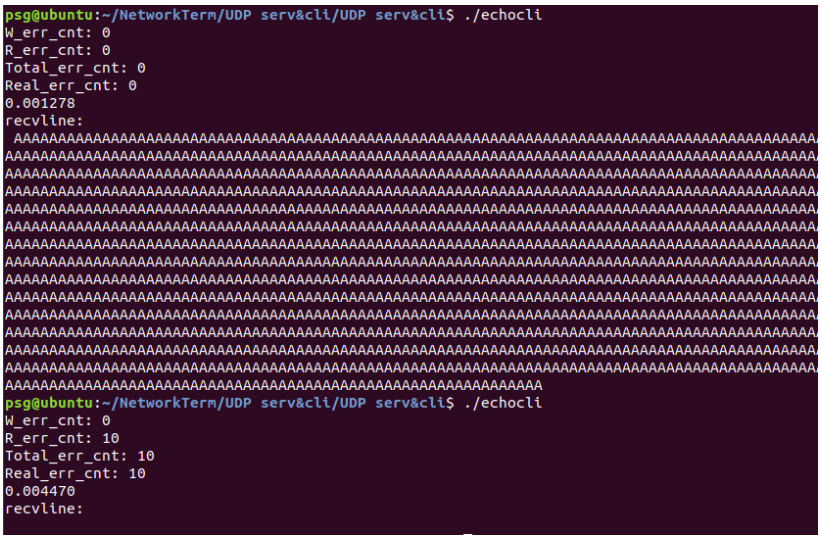
서버측 실행화면. 창의마루 공유기를 사용 중이다.



클라이언트 측 실행. *실수로 화면 캡쳐를 하지 못하였다.

4Kbytes 데이터가 1,000 번 왕복하는데 걸리는 시간이 TCP 에 비해 확연하게 빨라지는 것을 확인하였지만, 일부 패킷에 대해서만 에러가 날 것이라는 예상과는 달리 모든 패킷에 대해서 에러가 발생하였다. 이를 해결하기 위해 연결 방식을 변경, 방화벽 해제 등 다양한 시도를 해결하였지만, 다른 네트워크 상에서의 에러율 100%를 해결하지 못하였다.

해결하려 시도했던 과정 중 하나의 캡쳐 사진이다. 1,000 회의 전송 횟수를 10 회로 수정하였는데, 여전히 에러율은 100%였다.

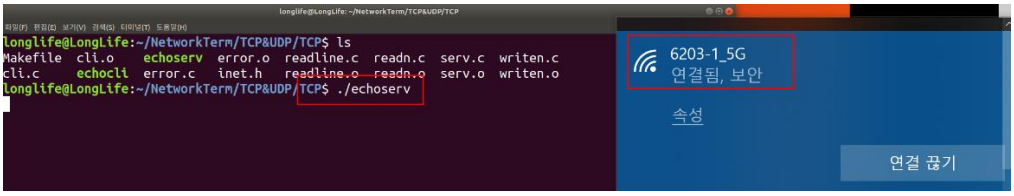


5.2 시행 2 차

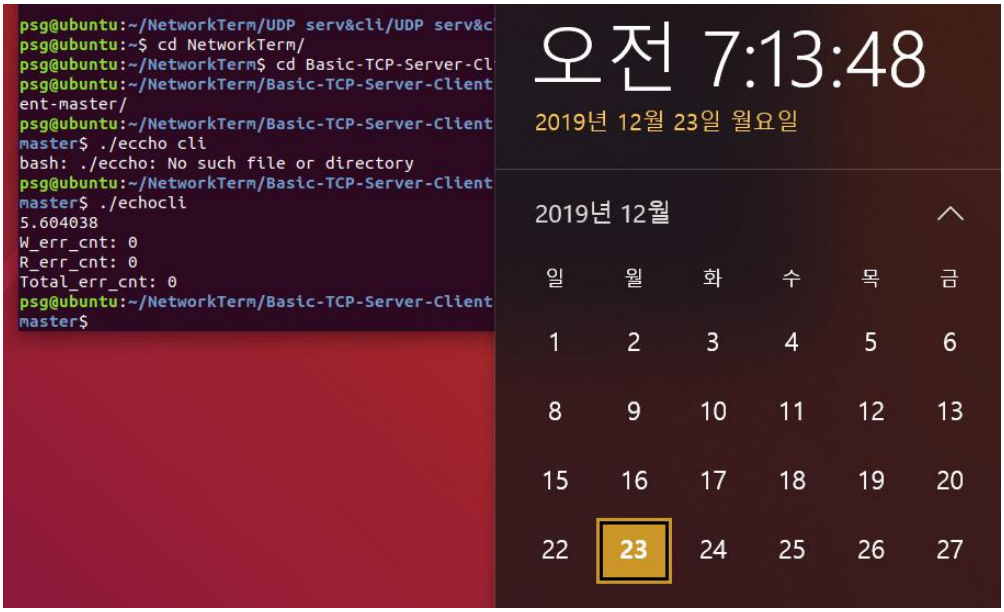
일시 : 12 월 23 일 월요일 07:00 ~ 09:00

5.2.1 TCP 통신

서버측 실행 화면. 창의마루 공유기를 사용 중이다.



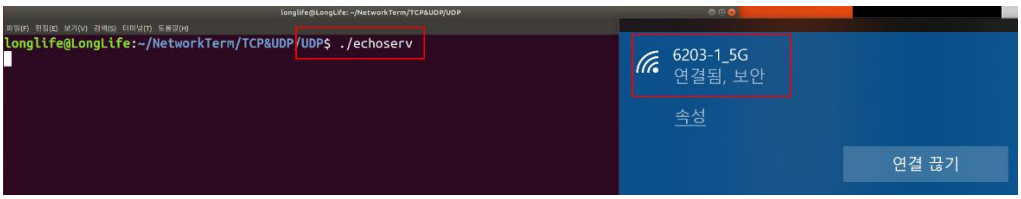
클라이언트측 실행화면. V50S_jangsoo 을 사용중이다.



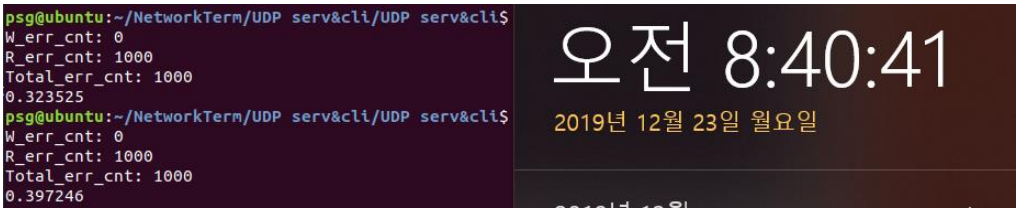
에러는 0%이고, 왕복 시간이 약 5 초로 1 차 시행보다 단축되었음을 확인할 수 있다.

5.2.2 UDP 통신

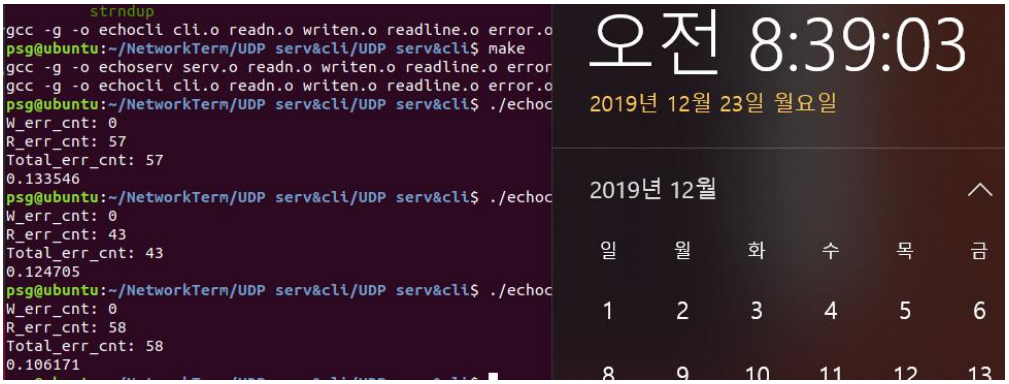
서버측 실행화면. 창의마루 공유기를 사용 중이다.



클라이언트측 실행화면. V50S_jangsoo 을 사용 중이다.

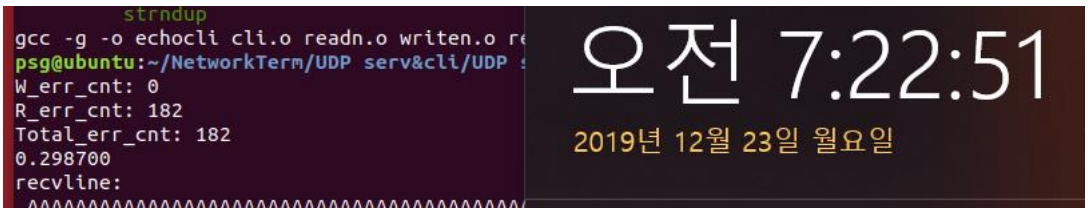


TCP 와 마찬가지로 4Kbytes 의 데이터가 1,000 번 왕복하는데 걸리는 시간은 약 0.3 초로 출력되었다. 이는 TCP 보다 UDP 가 빠를 것이라는 예측 결과와 일치하였다. 그러나 에러가 발생한 패킷이 1,000 개로 에러율 100%를 보여주는데 이는 예상치 못한 결과였다.



반면, 다음과 같이 서버와 같은 네트워크를 사용하여 UDP 통신을 해보았더니 43~58 개로 에러가 감소한 결과를 확인할 수 있었다. 시간도 같은 네트워크를 사용해서인지 약 0.1 초대로 더 빨라졌다.

추가적으로 아래는 창의마루의 공유기이지만 완전히 같진 않은 6203-1_2G 를 사용했을 때의 결과이다.



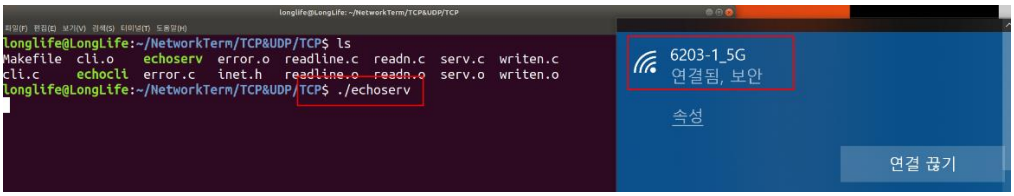
다른 네트워크를 사용했을 때보다 에러율이 감소했지만 같은 네트워크를 사용했을 때보다는 에러가 높음을 확인할 수 있다.

5.3 시행 3 차

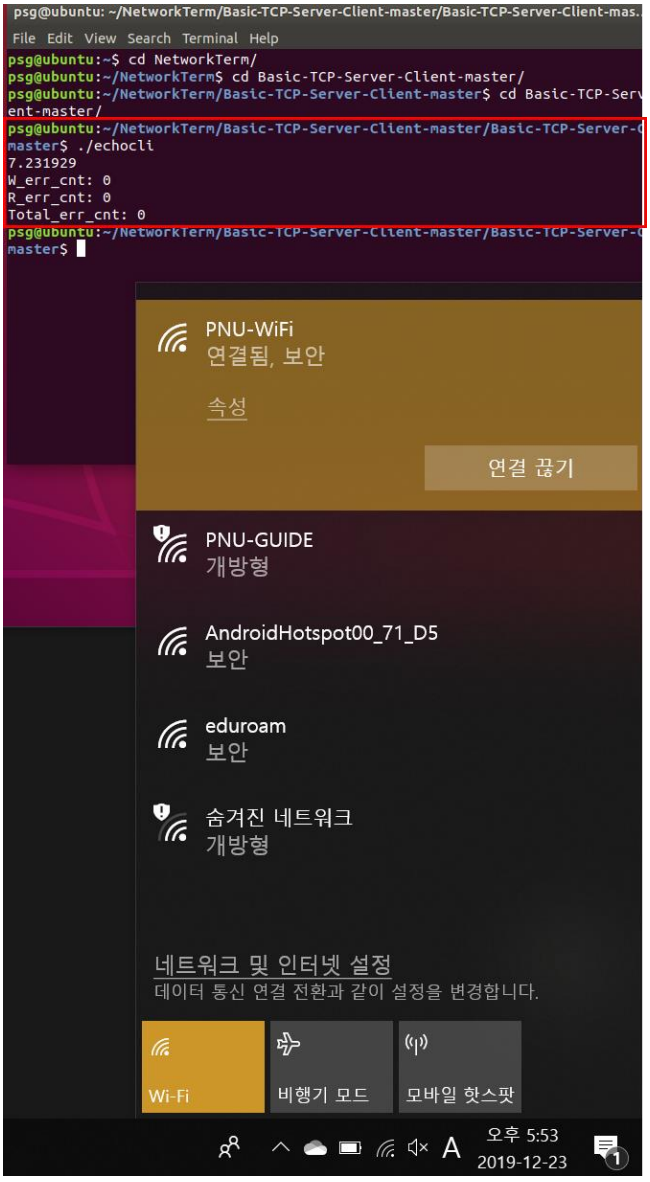
일시 : 12 월 23 일 월요일 17:30 ~ 19:30

5.3.1 TCP 통신

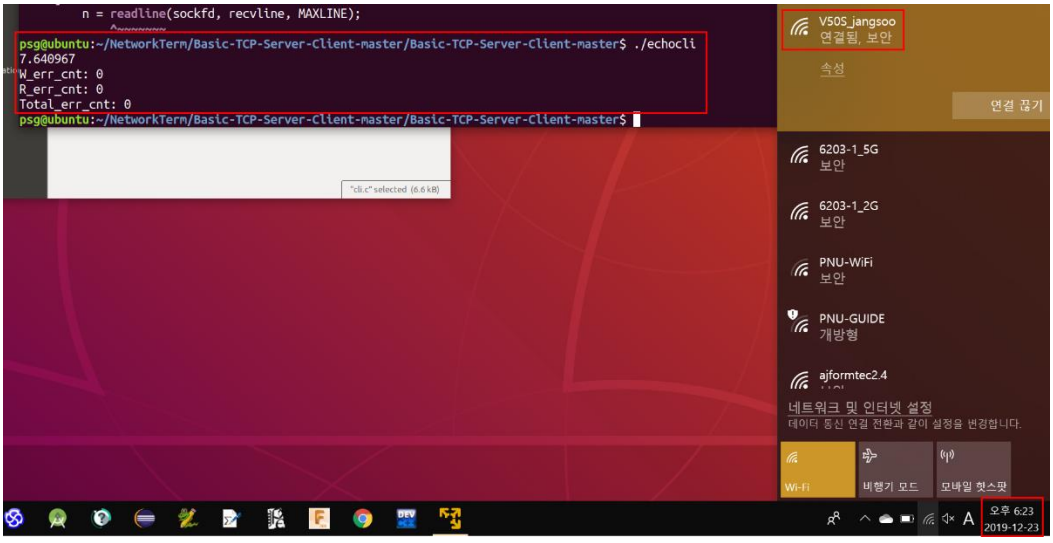
서버측 실행 화면. 창의마루 공유기를 사용 중이다.



클라이언트측 실행화면. PNU-WiFi 를 사용 중이다.



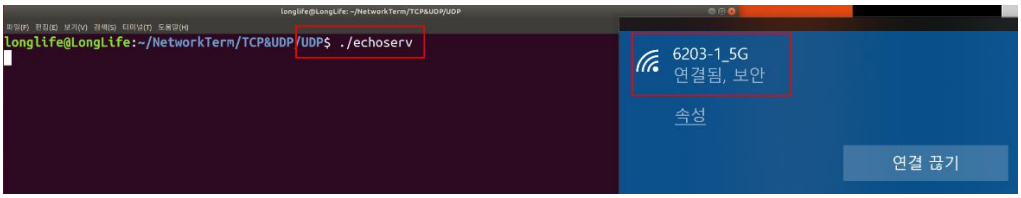
시간은 약 7 초이고, 에러가 발생한 패킷의 수는 0 으로 에러율은 0%임을 확인할 수 있다.



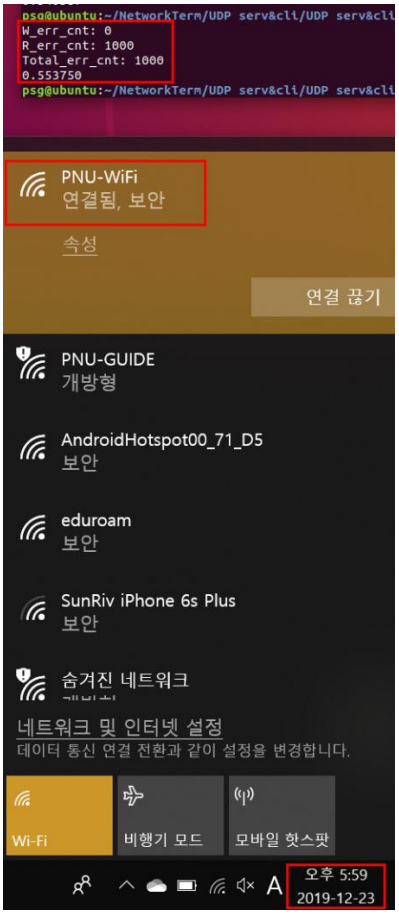
혹시 몰라서 다른 네트워크인 휴대폰 핫스팟을 사용해보았는데 왕복 시간이 조금 더 길어졌다.

5.3.2 UDP 통신

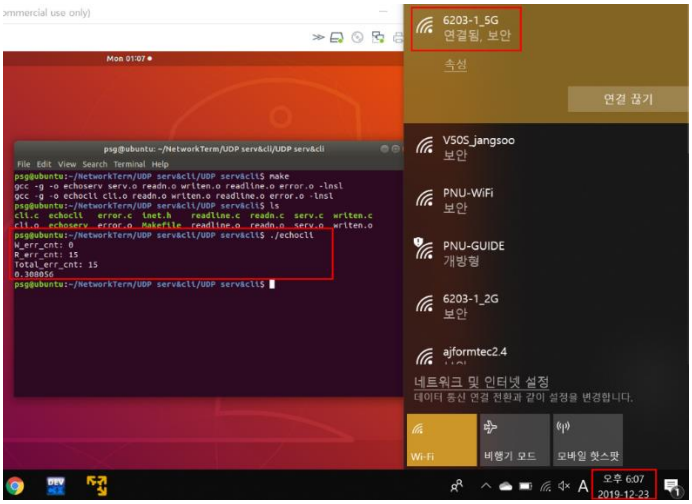
서버측 실행화면. 창의마루 공유기를 사용 중이다.



클라이언트측 실행화면. PNU-WiFi 를 사용 중이다.



서버와 다른 네트워크를 사용했더니 모든 패킷에 대해서 에러가 발생하여 에러율이 100%였다.



서버와 같은 네트워크를 사용했을 때는 일부 패킷에만 에러가 발생했고, 시간도 더 빨랐다.

06. 결론

6.1 에러율 관점

우선 TCP 는 신뢰성있는 전송이라고 배운 그대로 정말 에러가 하나도 발생하지 않았다. 에러가 발생하면 재전송을 요청해서 에러가 없을때까지 재전송을 요청한다는 프로토콜이라고 배웠었는데, 그때문인지 사진에서도 볼 수 있지만 에러가 하나도 발생하지 않았다. 창의마루에 서버를 두고 클라이언트의 IP 는 같은 창의마루내의 같은 와이파이, 창의마루내의 다른 와이파이, 부산대 전체에 있는 PNU-Wifi, 모바일 핫스팟 이렇게 총 네가지를 시험해보았다. 하지만 TCP 는 그 중 단 한번의 에러도 발생하지 않았다. 신뢰성있는 전송이라고는 하지만 한두번의 에러는 발생할 줄 알았는데, 예상과는 약간 다른 결과였다.

반면 UDP 는 TCP 와 극명한 차이를 보여주었다. 아까 TCP 에서 말했던대로 UDP 를 실험할때에도 네가지 IP 에서 실험했었는데, 같은 창의마루 내에 존재하는 두가지 와이파이로 테스트했을때에는 비교적 낮은 에러율을 보였다. 완전 같은 와이파이인 A 와이파이의 경우에는 1000 번 중 평균 50 번 정도의 에러가 발생했다면, 같은 창의마루내에 존재하더라도 다른 와이파이를 사용한다면 200 회 전후의 에러가 발생했다.

심지어 PNU-Wifi 나 모바일 핫스팟을 사용할 경우에는 1000 번의 전송중에 1000 번의 에러가 발생했다. 학교 무선 네트워크에서 외부의 패킷을 막아버리는 방화벽이 존재하는지는 알 수 없지만 핫스팟과 PNU-Wifi 가 1000 번 다 실패한 것은 의외의 결과였다. 여러 번 시도해보았지만 결과는 달라지지 않았다.

약 4 일에 걸쳐 UDP 에 관한 조건만 여러가지 변경해보았다. 방화벽문제인가 싶어서 해당 공유기의 방화벽 설정도 해제해보고, VMware 를 사용한 실험이기에 VMware 의 네트워크 설정도 Bridged 에서 NAT 을 왔다갔다 하며 실험해보았다. 또한 노트북 자체의 방화벽 설정도 완전히 해제해보았으며 인터넷에 존재하는 다른 코드로도 실행해보았지만 UDP 는 외부 IP 에 대해서는 100%의 에러율을 보였다.

혹시나 이 에러라는 것이 패킷로스뿐만 아니라 단순한 비트에러도 카운트하는것인가 싶어서 비트에러만을 검사하는 코드도 따로 작성해보았지만 전체 패킷에서 비트에러가 발견되었다. 즉 측정한 결과에 따르면 UDP 는 같은 서브넷내에서는 괜찮은 에러율을 보이지만, 다른 서브넷 IP 에서 전송한 패킷의 경우에는 항상 조금이라도 비트에러가 존재한다는 결론을 내릴 수 있었다.

결론적으로 TCP 는 에러율이 0%였으며, UDP 는 네트워크에 따라 다른 에러율을 보였다. 하지만 같은 네트워크를 사용할 경우 비교적 낮은 에러율을 보였으며 TCP 와 큰 차이가 없는 수준이었다. 다른 네트워크를 사용했을 경우 100%의 에러율을 보여 예상과는 다른 결과를 측정할 수 있었다.

6.2 전송시간 관점

에러율 관점에서와는 달리 전송시간 관점에서는 예상과 비슷한 결과를 관측할 수 있었다. TCP 는 재전송요청으로 인해 전송시간이 비교적 길것이고, UDP 는 재전송요청이 없기 때문에 전송시간이 비교적 짧을것이라는 예상대로였다. 1 차 관측은 밤 12 시에 시행했는데, 이때 TCP 는 1000 번 전송에 약 6 초라는 시간이 걸렸다. 동시간대의 UDP 는 약 0.44 초정도로 TCP 에 비해 열배 이상 빠른 전송속도를 보였다. 이때 Timeout 은 1 초로 설정된 상태였다.

2 차 관측은 아침 7 시, 사람들이 비교적 덜 이용하는 시간대에 측정하였는데, 이때 TCP 는 5.6 초라는 전송시간을 측정할 수 있었다. 이는 1 차에 관측된 TCP 의 6 초라는 전송시간보다 0.4 초 빨라진 전송시간인데, 1000 번 전송이라는 것을 감안하면 1 회당 약 0.0004 초정도로 미묘한 속도 증가를 보였다.

동시간대에 측정한 UDP 는 세가지 IP 로 나누어서 측정하였다. 아래의 표를 참고하면 다른 네트워크의 IP 를 사용했을때는 약 0.4 초, 같은 네트워크를 사용했을때는 약 0.12 초, 같은 창의마루 내의 다른 와이파이를 사용했을때는 0.298 초가 걸렸다. IP 에 따라서도 시간차이가 꽤 발생했지만, 전체적으로 TCP 보다 훨씬 더 빠른 속도였다. 다른 네트워크와 같은 네트워크간에는 두배 이상의 차이를 측정할 수 있었다.

3 차 측정은 같은날 저녁 6 시였다. 시간대가 시간대인지라 1 차와 2 차보다 더 많은 트래픽을 예상할 수 있었고, 실제로도 더 긴 시간이 걸렸다. TCP 의 경우에는 1000 회전송에 약 7.23 초가 걸렸다. 1 차와 2 차의

TCP 전송시간과 비교해보더라도 가장 긴 시간이 걸렸으며, UDP 와는 비교가 의미가 없을 정도로 큰 차이가 났다.

UDP 의 경우 두가지 IP 로 나누어서 측정했는데, 다른 네트워크에 속한 IP 의 경우 0.553 초정도로 2 차에 측정했던 0.4 초보다 증가한 전송시간을 볼 수 있었다. 같은 네트워크에 속하는 IP 로 실험했을때에는 약 0.308 초로 이 역시 2 차에 측정했을때보다 두배 이상의 시간이 걸렸다.

결론적으로 저녁시간대 > 밤시간대 > 아침시간대 순으로 트래픽이 많았으며 이에 따라 전송시간도 더 오래걸림을 관찰할 수 있었다. 또한 TCP 와 UDP 는 최소 10 배이상의 시간 차이를 보였으며 TCP 가 가장 빠를때에도 UDP 가 가장 느린 경우를 따라잡지 못했다.

	1 차 – 23 일 00 시	2 차 – 23 일 07 시	3 차 – 23 일 18 시
TCP	6.004752	5.604038	7.231929
UDP_1 (다른 네트워크)	0.4470	0.397246	0.553750
UDP_2 (같은 네트워크)	.	0.124705	0.308056
UDP_3 (중간 네트워크)	.	0.298700	.

TCP, UDP 시간 비교 표

	1 차 – 23 일 00 시	2 차 – 23 일 07 시	3 차 – 23 일 18 시
TCP	0	0	0
UDP_1 (다른 네트워크)	1,000	1,000	1,000
UDP_2 (같은 네트워크)	.	43~58	15
UDP_3 (중간 네트워크)	.	182	.

TCP, UDP 에러 비교 표