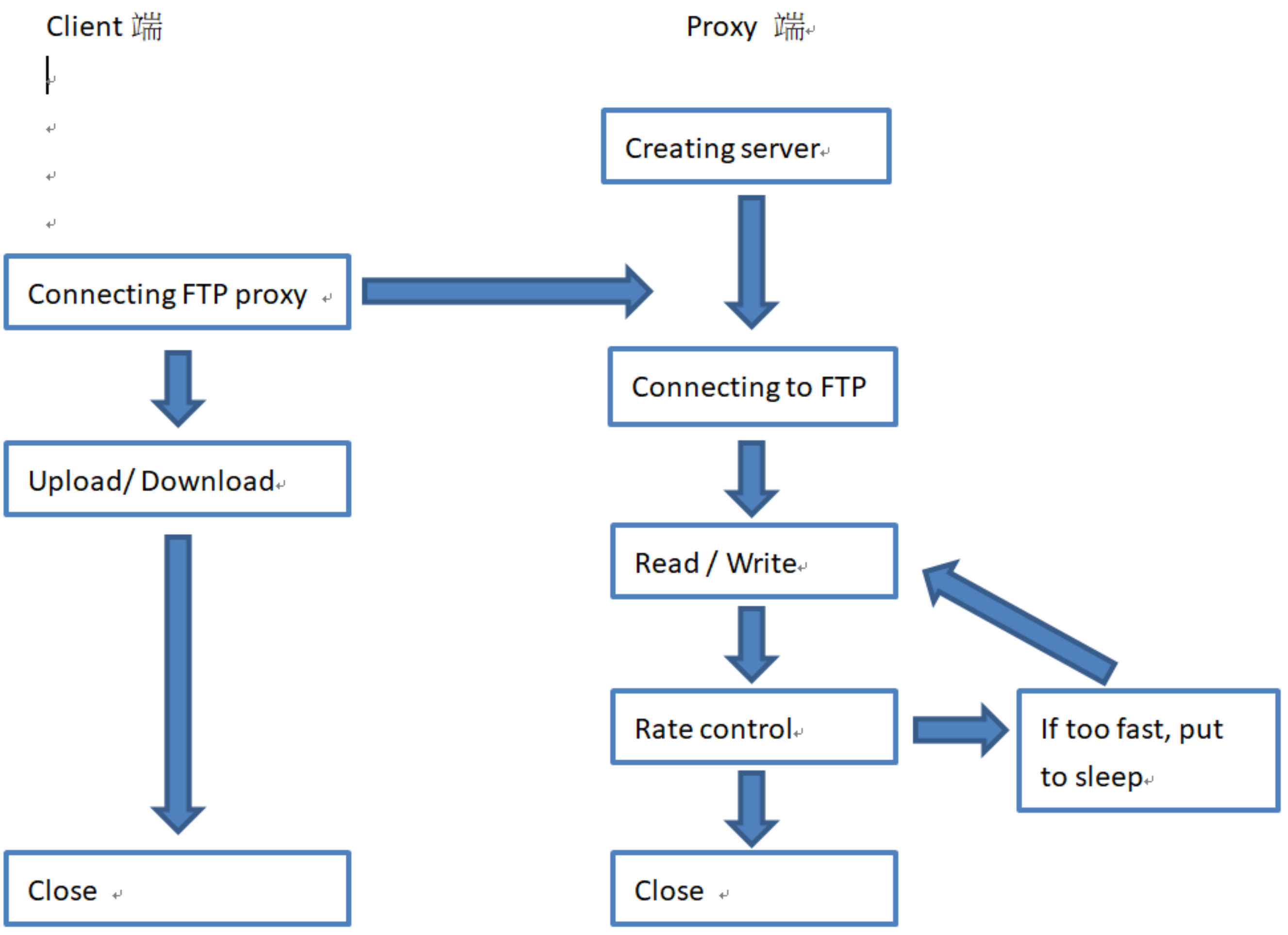
Team 28 Report

Architecture of the projcet

Flow chart



Code Explanation

main()
在main function裡，首先透過argv[]取得使用者輸入的rate值，接著create server 等待client連線，當成功連線時會呼叫proxy_func()來連FTP server。
proxy_func()
在這裡會呼叫connect_FTP()來連結FTP server，連上之後會開始進行upload和download，第一筆資料上傳前，意即當byte_num為零時會先記下當下的時間，傳完資料時會再記一次時間，接著把開始時間、結束時間、byte數等資訊一併傳入rate_control()裡。
How the proxy works? How the proxy communicates with the server and the client?
首先在main裡頭create server，接著listen and bind，連上server。

```
if (bind(listenfd, (struct sockaddr *)&servaddr , sizeof(servaddr)) < 0) {
    //print the error message
    perror("bind failed. Error");
    return -1;
  }
```

當成功bind後連上client，接著main裡面會使用fork的方式來呼叫proxy_fun()並且會在proxy_fun()裡連結FTP server。 在proxy_fun()中，當proxy連上server與client後，即可開始傳輸資料，FD_ISSET()負責管理upload和download，每次upload和download，proxy會fork出data channel來處理和server client的溝通。

```
if (FD_ISSET(clifd, &rset)) {
         // Client -> Server
  memset(buffer, 0, MAXSIZE);
  if ((byte_num = read(clifd, buffer, MAXSIZE)) <= 0) {
             print_is_forked(is_forked);
    printf("[!] Client terminated the connection.\n");
    break;
  }
```

upload data時，proxy先讀client傳來的東西

```
if (write(serfd, buffer, byte_num) < 0) {
  printf("\n[x] Write to server failed.\n");
  break;
}
```

接著proxy再將資料傳到server

How do you implement the approach of rate control

我們使用了定義在sys/time.h裡的struct

```
struct timeval {
    time_t       tv_sec;      /* seconds */
    suseconds_t tv_usec;      /* microseconds */
};
```

以及定義在sys/time.h裡的gettimeofday(...)來取的現在時間

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

現在時間會被存到timeval裡面，精確度可到達微秒。也因為精確度被提升到微秒的關係，我們使用usleep()而非sleep()。

main

要從command line parameter讀取rate，因為rate是固定的所以把它設成global variable。

```
    if (argc < 4) {
        // Missing parameter
        printf("\n[v] Usage: ./executableFile <ProxyIP> <ProxyPort> <rate> \n");
        return -1;
    }

    sscanf(argv[1], " %d.%d.%d.%d", &proxy_IP[0],
           &proxy_IP[1], &proxy_IP[2], &proxy_IP[3]);
    port = atoi(argv[2]);
    rate = atoi(argv[3]);    // Global variable
```

print_is_forked(int is_forked)
這個function是用來印出debug message，用來分辨是經過proxy的data channel還是command channel有資料流通。不是必要的功能但是能讓人更了解ftp protocol。

```
void print_is_forked(int is_forked){
        if(is_forked)
                printf("\nForked:");    // Data channel,
                                       // forked from command channel
        else
                printf("\n______:");    // Command channel
}
```

和原本code裡的debug message組合起來後會變成：

```
print_is_forked(is_forked);
printf("[v] Connect to FTP server\n");
// Console:
//     ______:[v] Connect to FTP server
//                    -> Command chennel connect to FTP server

// Console:
//     Forked:[v] Connect to FTP server
//                    -> Data chennel connect to FTP server
```

proxy_func(int ser_port, int clifd, int is_forked)
多傳入的is_forked是用來分辨在data chennel還是command channel。
這裡會執行在做rate control時的前置工作，rate control前置流程如下：
0. 記錄第一次proxy傳送bytes的時間

1. 傳送bytes並且得到這次傳送的bytes數量
2. 計算總傳送bytes數量
3. 記錄現在時間
4. rate_control()

單次傳送bytes數量即是read的return value。再用一個variable蒐集每次傳送量即可得到傳送總量

取得時間方式：在第一次傳送，也就是第一次nready > 0時取得起始時間。之後每次取得現在時間。

這裡把所有需要的new variable都設成local variable，因為fork之後global variable是相同的，所以當data channel重新呼叫一次proxy_func時會拿到記錄著command channel狀態的total_upload_bytes/total_download_bytes/first_transmit，這種情況會造成誤差，因此我們統一把所有新增的variables設定成local variable。

```
int proxy_func(int ser_port, int clifd, int is_forked){
    /* SKIP */

    // Must be local variable!
    int byte_num = 0;
    struct timeval start_time, current_time;    // Save time
    int first_transmit = 1;    // Is this the 1st time proxy transmit data?

    unsigned long total_upload_bytes = 0;    // #Bytes uploaded
    unsigned long total_download_bytes = 0;  // #Bytes downloaded

    // connect to FTP server
    if ((serfd = connect_FTP(ser_port, clifd, is_forked)) < 0) {
        // 判斷是data channel還是command channel連到ftp server
        print_is_forked(is_forked);
        printf("[x] Connect to FTP server failed.\n");
        return -1;
    }

    /* SKIP */

    // selecting
    for (;;) {
        /* SKIP */

        if (nready > 0) {
            if(first_transmit){
                // 1st time transfering bytes
                // 存下第一次轉傳時的時間
                gettimeofday(&start_time, NULL);
                first_transmit = 0;
            }
            // check FTP client socket fd
            if (FD_ISSET(clifd, &rset)) {
                // Client -> Server
                memset(buffer, 0, MAXSIZE);
                // Receive bytes
                // 得到單次傳送量
                if ((byte_num = read(clifd, buffer, MAXSIZE)) <= 0) {
                    print_is_forked(is_forked);
                    printf("[!] Client terminated the connection.\n");
                    break;
                }
                // Send bytes
                if (write(serfd, buffer, byte_num) < 0) {
                    printf("\n[x] Write to server failed.\n");
                    break;
                }

                // 記錄現在時間，和初始時間的時間差會交給rate_control()去處理
                gettimeofday(&current_time, NULL);
                // 算總傳送量
                total_upload_bytes += byte_num;
                // Rate control!
                rate_control(start_time, current_time, total_upload_bytes);
            }

            // check FTP server socket fd
            if (FD_ISSET(serfd, &rset)) {
                // Client <- Server
                memset(buffer, 0, MAXSIZE);
                // Receive bytes
                // 得到單次傳送量
                if ((byte_num = read(serfd, buffer, MAXSIZE)) <= 0) {
                    print_is_forked(is_forked);
                    printf("[!] Server terminated the connection.\n");
                    break;
                }

                /* SKIP */

                if (status == FTP_PASV_CODE && ser_port == FTP_PORT) {
                    /* SKIP */
                    // 進入passive mode
                    // Fork出data chennel

                    if ((childpid = fork()) == 0) {
                        // 只有被fork出來的data chennel才會執行這裡
                        // Port convertion
                        // Create server to accept client
                        // Client connected!
                        printf("\nForked:");
                        printf("[v] Data connection from:
                                %s:%d connect.\n", inet_ntoa(cliaddr.sin_addr),
                                htons(cliaddr.sin_port));
                        // Call proxy_func again to connect to ftp server
                        //  (with new data port!)
                        proxy_func(new_data_port, connfd, 1); // File transmiting...
                        // File transmition done,
                        // Close data chennel
                        printf("\nForked:");
                        printf("[!] End of data connection!\n");
                        // 直接用exit!
                        exit(0);
                    }
                }


                if (write(clifd, buffer, byte_num) < 0) {
                    printf("[x] Write to client failed.\n");
                    break;
                }
                // 記錄現在時間，和初始時間的時間差會交給rate_control()去處理
                gettimeofday(&current_time, NULL);
                // 算總傳送量
                total_download_bytes += byte_num;
                // Rate control
                rate_control(start_time, current_time, total_download_bytes);
            }
        } else {
            printf("\n[x] Select() returns -1. ERROR!\n");
            return -1;
        }
    }
    return 0;
}
```

rate_control(struct timeval start_time, struct timeval current_time, unsigned long total_send_bytes)
Rate control流程：

1. 算出起始時間到現在時間的時間差(elapsed_time)
2. 根據global variable rate算出理想時間差(ideal_elapsed_time)

   > 理想時間 = 總傳送量(bytes) / rate(bytes / μs)

   從command line得到的rate單位是KB，總傳送量單位是bytes，時間差的單位是μs。因此要對rate做單位換算。

   > rate * 1024(KB -> B) / 1000000(s -> μs)

3. 算出理想時間跟實際時間的時間差，如果理想時間大於實際時間就用usleep。透過sleep把實際時間拉成跟理想時間相同

```
void rate_control(struct timeval start_time,
                  struct timeval current_time,
                  unsigned long total_send_bytes) {

        // Real elapsed time from the beginning in us
        double elapsed_time =
            (current_time.tv_usec - start_time.tv_usec) +
            (current_time.tv_sec - start_time.tv_sec) * 1000000.0;
        // Ideal elapsed time from the beginning in us
         // Remember to do type casting!
        double ideal_elapsed_time =
            (double)total_send_bytes / ((double)rate * 1024.0 / 1000000);

        if(ideal_elapsed_time - elapsed_time > 0)
                usleep(ideal_elapsed_time - elapsed_time);

}
```

Probelm confronted

Global variabl or local variable

一開始把所有新增的變數都設成global variable，結果在進入因為data channel之後first_transmit, total_download/upload_byte都存著command chennel的狀態而使程式crash。

Debug Message

原本程式裡就有印出一些debug message，但是為了能分辨是data channel還是command channel印出的訊息而增加print_is_forked（int isforked)

How to run our code

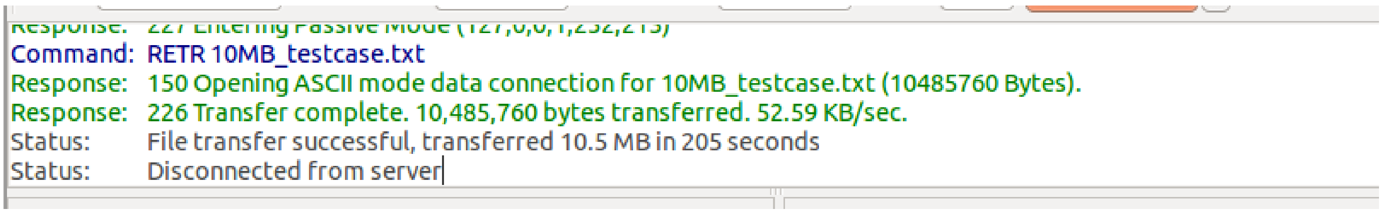
Naive way

```
# 1. Compile proxy
gcc -g 28.c -o ftp_proxy

# 2. Run proxy
./ftp_proxy 127.0.0.1 8888 <rate>
```

Lazy way

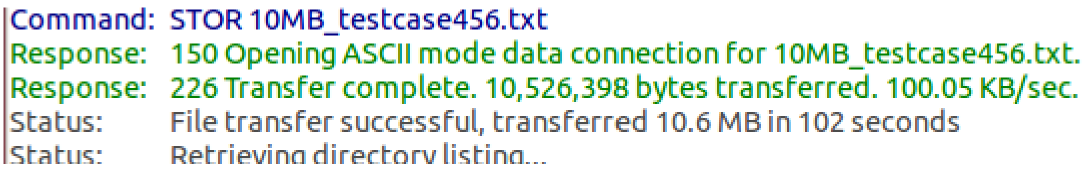
```
# 1. Compile proxy
make

# 2. Run proxy
sh runServer.sh <rate>
```

Experiment results



```
Response:  227 Entering Passive Mode (127,0,0,1,232,213)
Command:   RETR 10MB_testcase.txt
Response:  150 Opening ASCII mode data connection for 10MB_testcase.txt (10485760 Bytes).
Response:  226 Transfer complete. 10,485,760 bytes transferred. 52.59 KB/sec.
Status:    File transfer successful, transferred 10.5 MB in 205 seconds
Status:    Disconnected from server
```

downloading 10 MB file in rate 50 KB/s



```
Command:   STOR 10MB_testcase456.txt
Response:  150 Opening ASCII mode data connection for 10MB_testcase456.txt.
Response:  226 Transfer complete. 10,526,398 bytes transferred. 100.05 KB/sec.
Status:    File transfer successful, transferred 10.6 MB in 102 seconds
Status:    Retrieving directory listing...
```

uploading 10 MB file in rate 100 KB/s

Team members:

103033241林宛頤
103033121王立友