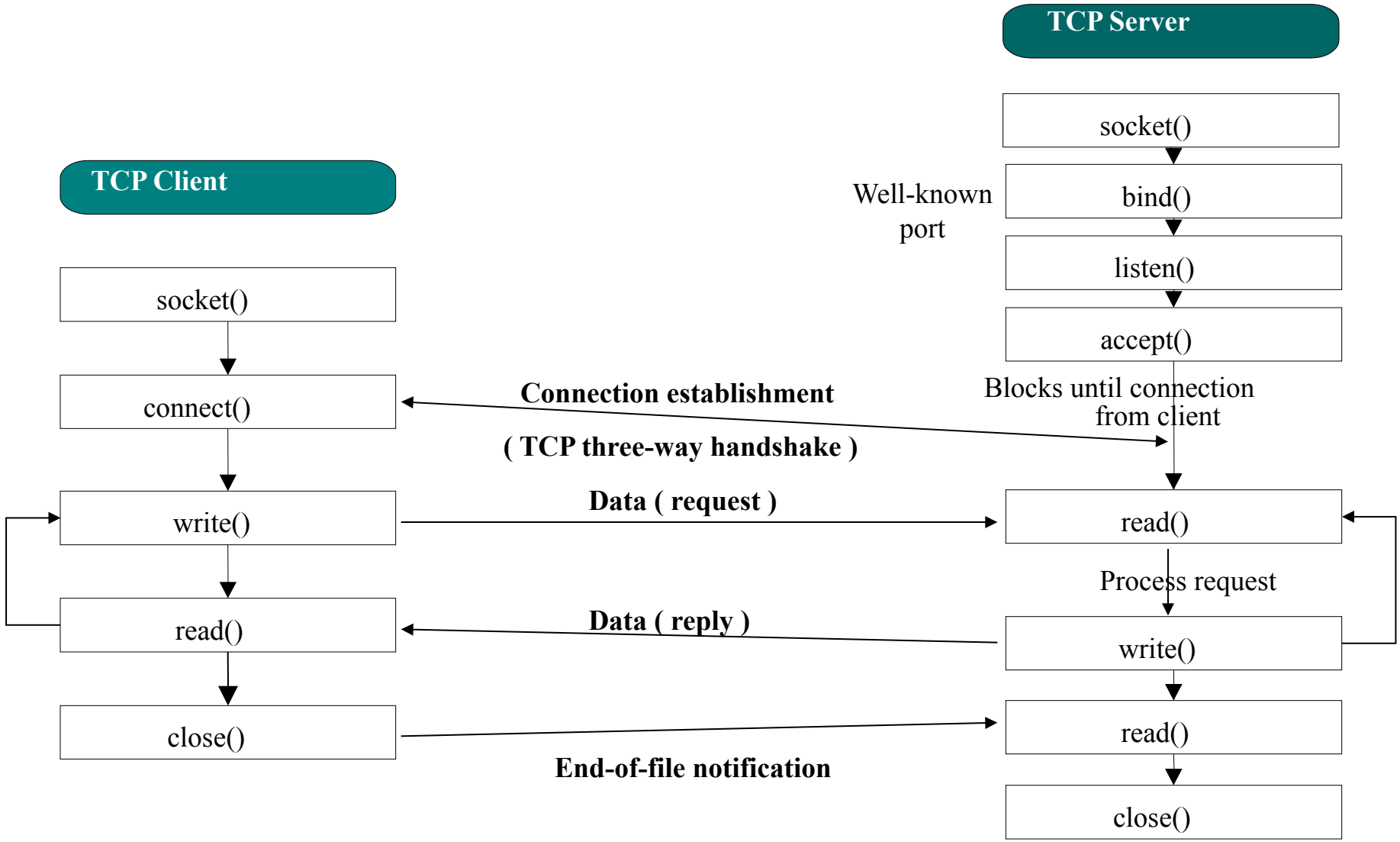
Lab 3
TCP Socket Programming

Guideline

- Read text materials and practice TCP Socket Programming
- Use your notebook PC to examine context and produce the source code when you finish the successful practice.
- Server and client should be executed in different computer

Socket Communication Procedure



Practice 1: Lottery Server / Lottery Client

- Lottery Server manage the 6 numbers. Each number is not coincide and range is from 1 to 45. When client send the ‘GEN’ message, server generate new random number until count is lower than 6. And when client send the ‘LIST’ message, server return the 6 numbers to client. And when client send ‘END’ message server close the connection and end the program.
- Lottery Client get the command from user. And send user’s command to server. Close the connection and end the program when user’s command is ‘END’.
  - 0: GEN
  - 1: LIST
  - 2: END

Server Process

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

#define GEN 0
#define LIST 1
#define END 2

void error(char* msg) {
        perror(msg);
        exit(1);
}

int main(int argc, char* argv[]) {
        int listenfd, connfd;
        struct sockaddr_in cliaddr, servaddr;
        socklen_t clilen;
        int number[6] = {0,};
        int count = 0;

        ⓐ   //socket()

        bzero(&servaddr, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        servaddr.sin_port = htons(atoi(argv[1]));

        ⓑ   //bind()
```

```
#include <sys/socket.h>                /* UNIX */
SOCKET socket (int family, int type, int protocol);
Returns: socket descriptor on success, -1 (UNIX)
```

```
#include <sys/socket.h>                /* UNIX */
int bind (SOCKET s, const struct sockaddr *myaddr, int addrlen);
Returns: 0 on success, -1 (UNIX)
```

Server Process

```
ⓒ //listen()

ⓓ //accept()

srand(time(NULL));
while(1) {
    int recv_msg;
    char send_msg[256];
    ⓔ //read()
    if(recv_msg == GEN) {
        if(count < 6) {
            int i, new_num = rand()%45 + 1;
            for(i = 0; i < count; ++i) {
                if(new_num == number[i]) {
                    new_num = rand()%45 + 1;
                    i = -1;
                }
            }
            sprintf(send_msg, "New number generated: %d",
new_num);
            number[count++] = new_num;
        }
        else
            sprintf(send_msg, "Full");
    }
    else if(recv_msg == LIST) {
        sprintf(send_msg, "numbers: %d %d %d %d %d %d",
number[0], number[1], number[2], number[3], number[4], number[5]);
    }
    else {
        close(connfd);
        break;
    }
    ⓕ //write()
}
close(listenfd);
return 0;
}
```

```
#include <sys/socket.h>          /* UNIX */
int listen (SOCKET s, int backlog);
Returns: 0 on success, -1 on error
```

```
#include <sys/socket.h>

int accept (int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);

                                  Returns: non-negative descriptor if OK, -1 on error
```

```
#include <sys/socket.h>   /* UNIX */
int read (SOCKET s, void *buf, size_t bufsize);          /* UNIX */
Returns: # of bytes read  (>0), 0 if received FIN and no more data, -1 on failure

int write (SOCKET s, const void *buf, size_t len);       /* UNIX */
Returns: # of bytes transferred on success, -1 on failure
```

```
#include <unistd.h>

int close (int sockfd);

                  Returns: 0 if OK, -1 on error
```

Client Process

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

#define GEN 0
#define LIST 1
#define END 2

void error(char* msg) {
        perror(msg);
        exit(1);
}

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

    ⓖ  //socket()

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr(argv[1]);
    servaddr.sin_port = htons(atoi(argv[2]));

    ⓗ  //connect()

    while(1) {
        int send_msg, len;
        char recv_msg[256];
        printf("Command (0:GEN 1:LIST 2:END): ");
        scanf("%d", &send_msg);
        ⓘ  //write()
        if(send_msg == END)
            break;
        ⓙ  //read()
        recv_msg[len] = 0;
        printf("%s\n", recv_msg);
    }
    close(sockfd);
    return 0;
}
```

```
#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);

                                                    Returns: 0 if OK, -1 on error
```

Execution

```
[ancl@anclab2 Lab5]$ ./LotteryServer 10000
[ancl@anclab2 Lab5]$
```

```
[ancl@anclab2 Lab5]$ ./LotteryClient 127.0.0.1 10000
Command (0:GEN 1:LIST 2:END): 0
New number generated: 32
Command (0:GEN 1:LIST 2:END): 0
New number generated: 5
Command (0:GEN 1:LIST 2:END): 0
New number generated: 22
Command (0:GEN 1:LIST 2:END): 1
numbers: 32 5 22 0 0 0
Command (0:GEN 1:LIST 2:END): 0
New number generated: 42
Command (0:GEN 1:LIST 2:END): 0
New number generated: 20
Command (0:GEN 1:LIST 2:END): 0
New number generated: 28
Command (0:GEN 1:LIST 2:END): 0
Full
Command (0:GEN 1:LIST 2:END): 1
numbers: 32 5 22 42 20 28
Command (0:GEN 1:LIST 2:END): 2
[ancl@anclab2 Lab5]$
```

Requirements

1) Please fill the blank and execute the program
2) Please check out the each function's return value, implement error handling procedure and execute 2 server in same port
3) Please examine the network packet using wireshark or tcpdump
4) Please explain the problem of executing and connecting two (or more) client to one server in this program (include the screenshot)

Practice 2: File Transfer

- This simple version of file transfer. Server get the file path for transfer from client (4th argument) and manage the file descriptor got from open() function
- Server read the data from file and send to client until EOF (End Of File)
- Client write the received data into file (4th argument)

Server Process

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <fcntl.h>

#define BUFSIZE 256

typedef struct file_segment {
        int len;
        char buf[BUFSIZE];
} file_segment;

void error(char* msg) {
        perror(msg);
        exit(1);
}

int main(int argc, char* argv[]) {
        int listenfd, connfd, filefd, len;
        struct sockaddr_in cliaddr, servaddr;
        socklen_t clilen;
        char recv_msg[BUFSIZE];
        file_segment segment;

        ⓐ  //socket()

        bzero(&servaddr, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        servaddr.sin_port = htons(atoi(argv[1]));

        ⓑ  //bind()

        ⓒ  //listen()

        ⓓ  //accept()

        ⓔ  //read file path from client
        recv_msg[len] = 0;
        printf("%s\n", recv_msg);
        if((filefd = open(recv_msg, O_RDONLY)) == -1) {
                close(connfd);
                close(listenfd);
                error("file open() error");
        }

        while(ⓕ  //read data from file) {
                segment.len = len;
                ⓖ  //write segment to client
        }

        close(filefd);
        close(connfd);
        close(listenfd);
        return 0;
}
```

Client Process

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

#define BUFSIZE 256

typedef struct file_segment {
        int len;
        char buf[BUFSIZE];
} file_segment;

void error(char* msg) {
        perror(msg);
        exit(1);
}

int main(int argc, char* argv[]) {
        int sockfd, filefd, len;
        struct sockaddr_in servaddr;
        file_segment segment;

        ⓗ   //socket()

        bzero(&servaddr, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = inet_addr(argv[1]);
        servaddr.sin_port = htons(atoi(argv[2]));

        ⓘ   //connect()

        if((filefd = open(argv[4], O_CREAT|O_WRONLY,
S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH|S_IXOTH)) == -1)
                error("file open() error");

        ⓙ   //write file path(argv[3]) to server

        while(ⓚ   //read segment from server)
                Ⓛ   //write data to file

        close(filefd);
        close(sockfd);
        return 0;
}
```

Execution

```
[ancl@anclab2 Lab5]$ cat ~/temp1
temp file
for test
network programming
[ancl@anclab2 Lab5]$ ./FTServer 10000
/home/ancl/temp1
[ancl@anclab2 Lab5]$
```

```
[ancl@anclab2 Lab5]$ ls
FTClient  FTClient.c  FTServer  FTServer.c  LotteryClient  LotteryClient.c  LotteryServer  LotteryServer.c
[ancl@anclab2 Lab5]$ ./FTClient 127.0.0.1 10000 /home/ancl/temp1 ./temp1
[ancl@anclab2 Lab5]$ ls
FTClient  FTClient.c  FTServer  FTServer.c  LotteryClient  LotteryClient.c  LotteryServer  LotteryServer.c  temp1
[ancl@anclab2 Lab5]$ cat temp1
temp file
for test
network programming
[ancl@anclab2 Lab5]$
```

Requirements

1) Please fill the blank and execute the program
2) Please examine the network packet using wireshark or tcpdump
3) Please modify this program to iterative file transfer program using ‘for’ or ‘while’. When each file transfer is end, get the new file path from user for next file transfer (it will not get the file path with argument anymore)

Lab 3 - Supplement

Examine the Network Packet

Objectives

- Understand the functionalities of the network protocols
- How to utilize toolkits to identify the TCP operation
- Through Lab 3, you will learn and practice …
  - How TCP works in packet communication levels….

Preparation for LAB 3

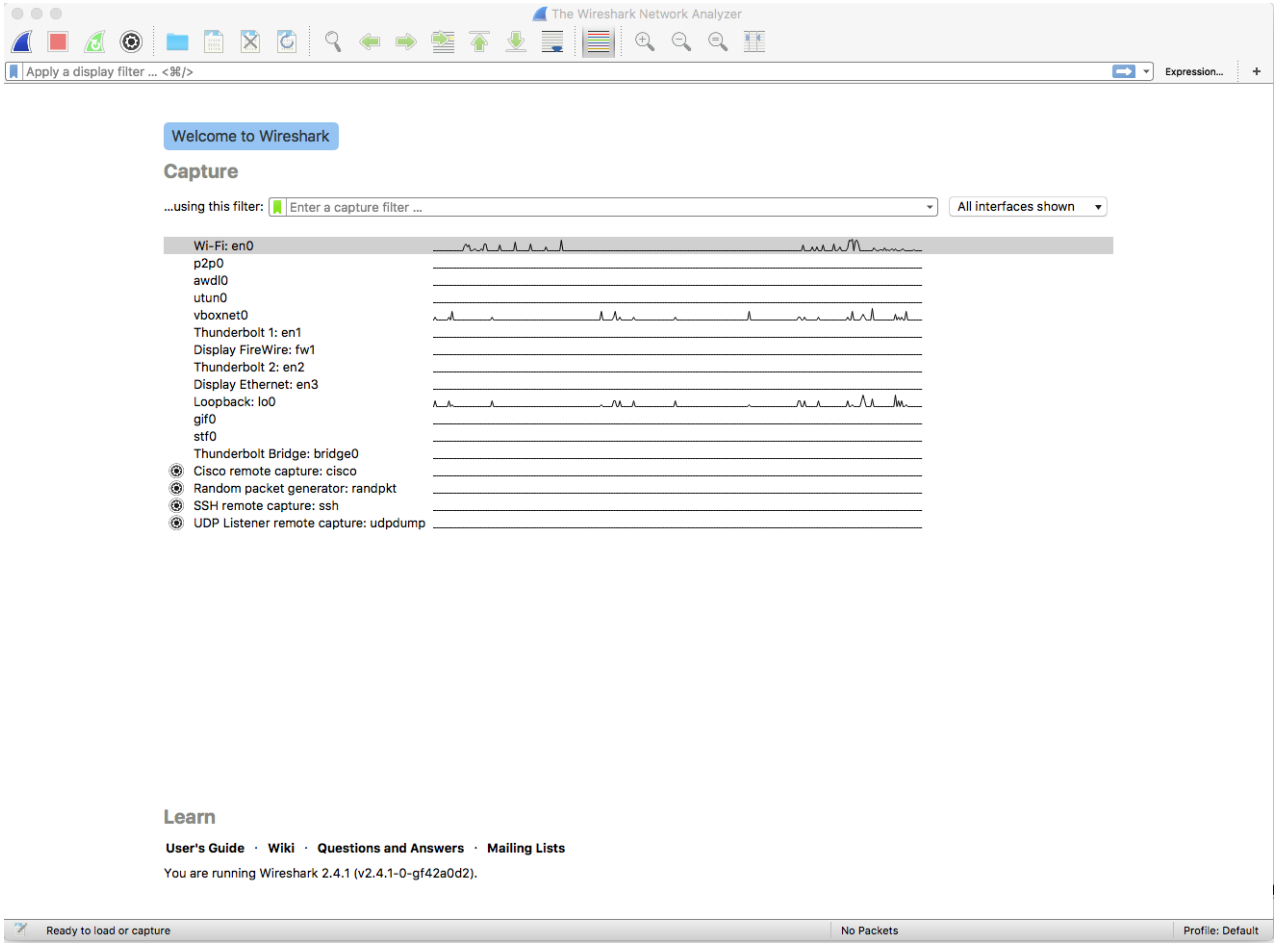
- Wireshark is a network protocol analyzer for UNIX and Windows. We use this tool in this LAB 3.
- Before installing Wireshark, we must install WinPcap. WinPcap is a library for capturing packets and loock network status. WinPcap can be downloaded from the following site.

http://www.winpcap.org/install/default.htm

- We also can download Wireshark from

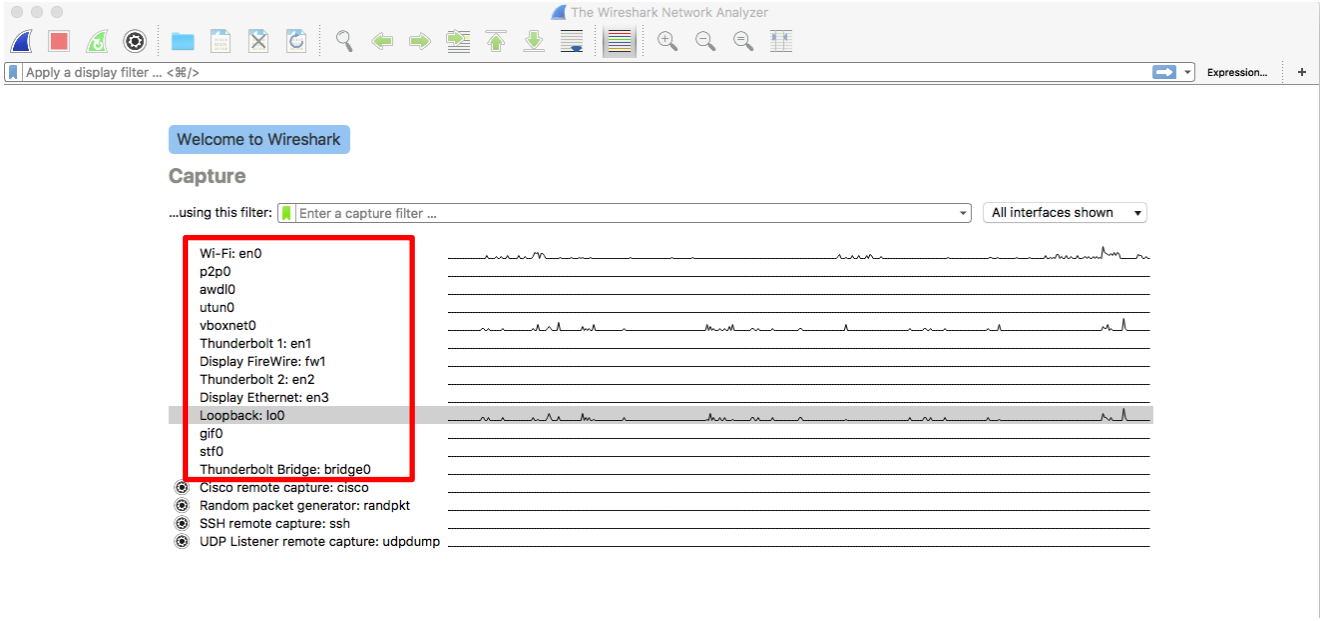
https://www.wireshark.org/download.html

Starting Wireshark



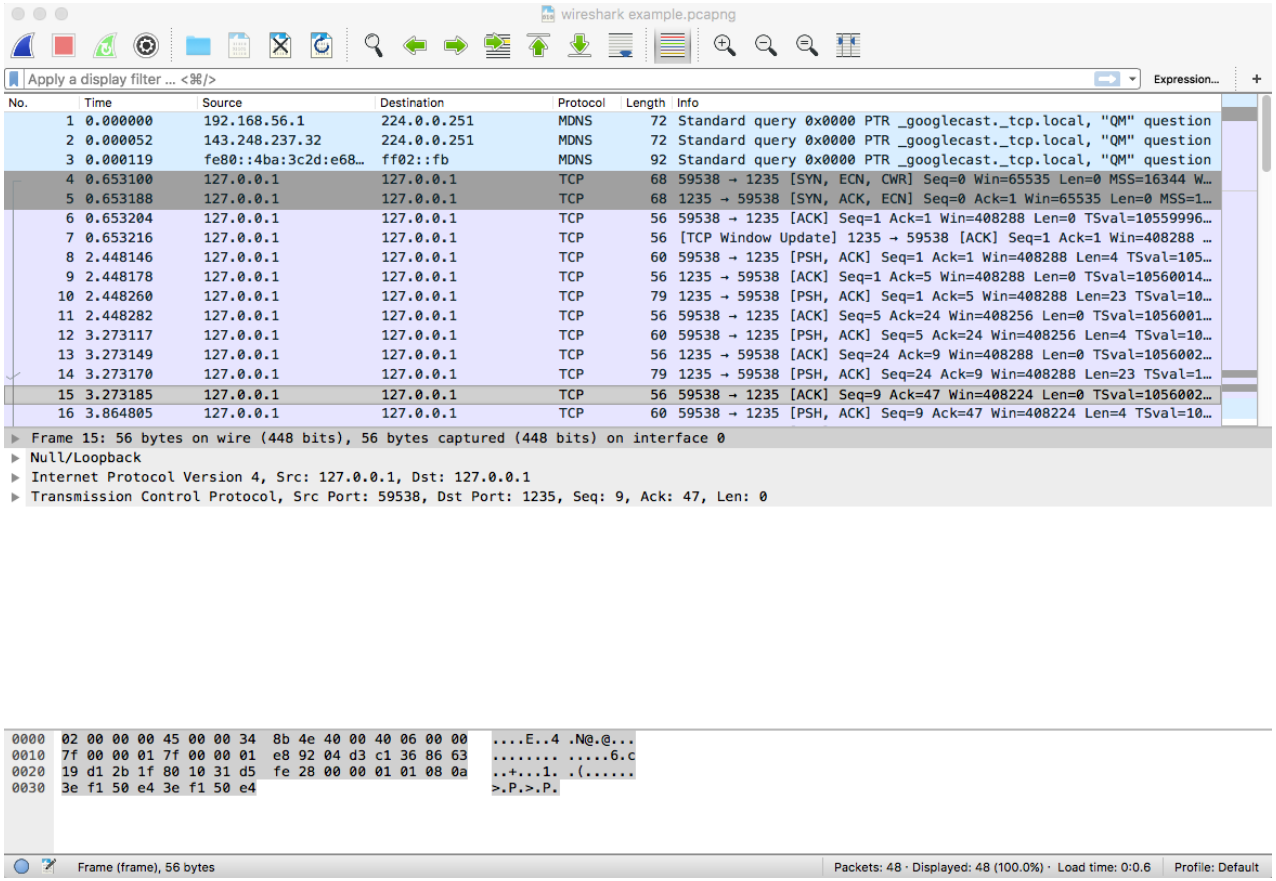
LAB 3 TCP operation (1)

- After starting Wireshark, you can choose which interface capture the network packet. If you want to check local network, double click Loopback.



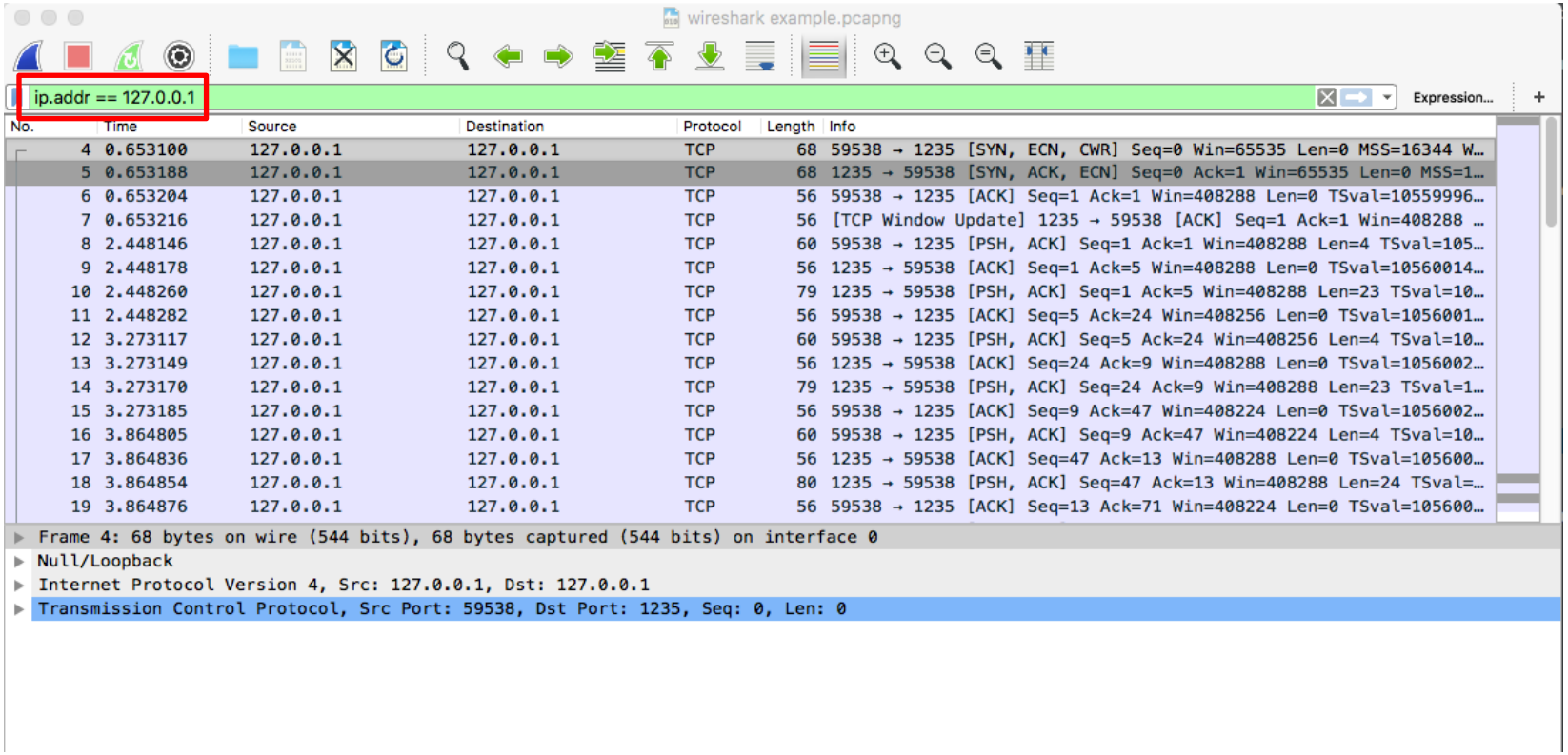
LAB 3 TCP operation (2)

- After choosing interface, we can see the TCP operation results.



LAB 3 TCP operation (3)

- If you want to removes packets that are not currently sent and received by ip, you can filter by typing <ip.addr == {my ip address}>.



LAB 3 TCP operation (4)

(1) How to check TCP Three-way Handshaking?

Server – execute server program

```
SeongHwanui-MacBook-Pro:Lab5_sol jihwankim$ ./LotteryServer 1235
```

Client - execute client program

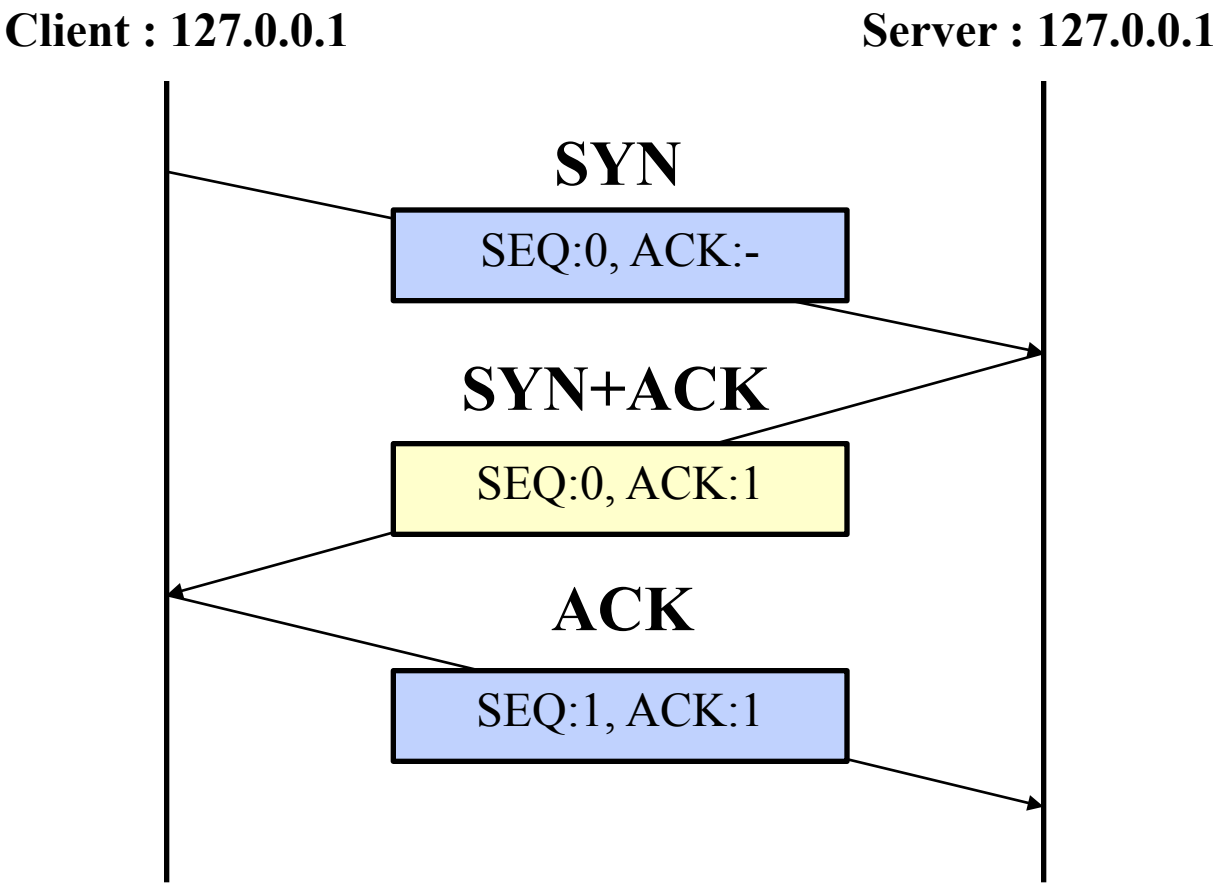
```
SeongHwanui-MacBook-Pro:Lab5_sol jihwankim$ ./LotteryClient 127.0.0.1 1235
```

Wireshark - check TCP Three-way Handshaking

No.	Time	Source	Destination	Protocol	Length	Info
4	0.653100	127.0.0.1	127.0.0.1	TCP	68	59538 → 1235 [SYN, ECN, CWR] Seq=0 Win=65535 Len=0 MSS=16344 W…
5	0.653188	127.0.0.1	127.0.0.1	TCP	68	1235 → 59538 [SYN, ACK, ECN] Seq=0 Ack=1 Win=65535 Len=0 MSS=1…
6	0.653204	127.0.0.1	127.0.0.1	TCP	56	59538 → 1235 [ACK] Seq=1 Ack=1 Win=408288 Len=0 TSval=10559996…

- TCP connections are established with a set of three messages called the three-way handshaking. This is for reliable and connection-oriented communication between server and client. Three-way handshaking follows these steps
- First, client sends SYN packet to server. This segment requests the establishment of the connection.
- Second, server sends SYN packet and ACK packet to client. ACK packet is for notifying well-receiving of packet that client sends.
- Third, client sends ACK packet to server. Then, three-way handshaking ends

LAB 3 TCP operation (5)



LAB 3 TCP operation (6)

(2) Two-way data flow

Server – receive command number from client and send some string to client

Client – send message "0" to server and receive some string from server

```
Command (0:GEN 1:LIST 2:END): 0
New number generated: 8
```

Wireshark - check TCP Three-way Handshaking

8	2.448146	127.0.0.1	127.0.0.1	TCP	60 59538 → 1235 [PSH, ACK] Seq=1 Ack=1 Win=408288 Len=4 TSval=105…
9	2.448178	127.0.0.1	127.0.0.1	TCP	56 1235 → 59538 [ACK] Seq=1 Ack=5 Win=408288 Len=0 TSval=10560014…
10	2.448260	127.0.0.1	127.0.0.1	TCP	79 1235 → 59538 [PSH, ACK] Seq=1 Ack=5 Win=408288 Len=23 TSval=10…
11	2.448282	127.0.0.1	127.0.0.1	TCP	56 59538 → 1235 [ACK] Seq=5 Ack=24 Win=408256 Len=0 TSval=1056001…

LAB 3 TCP operation (7)

(2) Two-way data flow

8	2.448146	127.0.0.1	127.0.0.1	TCP	60 59538 → 1235 [PSH, ACK] Seq=1 Ack=1 Win=408288 Len=4 TSval=105…
9	2.448178	127.0.0.1	127.0.0.1	TCP	56 1235 → 59538 [ACK] Seq=1 Ack=5 Win=408288 Len=0 TSval=10560014…
10	2.448260	127.0.0.1	127.0.0.1	TCP	79 1235 → 59538 [PSH, ACK] Seq=1 Ack=5 Win=408288 Len=23 TSval=10…
11	2.448282	127.0.0.1	127.0.0.1	TCP	56 59538 → 1235 [ACK] Seq=5 Ack=24 Win=408256 Len=0 TSval=1056001…

1. Client → Server / Send data (“0”)

```
▸ Frame 8: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Null/Loopback
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 59538, Dst Port: 1235, Seq: 1, Ack: 1, Len: 4
▾ Data (4 bytes)
    Data: 00000000
    [Length: 4]

0000  02 00 00 00 45 02 00 38  99 7e 40 00 40 06 00 00   ....E..8 .~@.@...
0010  7f 00 00 01 7f 00 00 01  e8 92 04 d3 c1 36 86 5b   ........ .....6.[
0020  19 d1 2a f1 80 18 31 d7  fe 2c 00 00 01 01 08 0a   ..*...1. .,......
0030  3e f1 4d b0 3e f1 46 b1  00 00 00 00               >.M.>.F. ....
```

2. Server → Client / Send Ack

```
▸ Frame 9: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
▸ Null/Loopback
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 1235, Dst Port: 59538, Seq: 1, Ack: 5, Len: 0

0000  02 00 00 00 45 00 00 34  72 a3 40 00 40 06 00 00   ....E..4 r.@.@...
0010  7f 00 00 01 7f 00 00 01  04 d3 e8 92 19 d1 2a f1   ........ ......*.
0020  c1 36 86 5f 80 10 31 d7  fe 28 00 00 01 01 08 0a   .6._..1. .(......
0030  3e f1 4d b0 3e f1 4d b0                           >.M.>.M.
```

Ack increase 4 (data length)

3. Server → Client / Send data (“New number …”)

```
▸ Frame 10: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface 0
▸ Null/Loopback
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 1235, Dst Port: 59538, Seq: 1, Ack: 5, Len: 23
▾ Data (23 bytes)
    Data: 4e6577206e756d6265722067656e6572617465643a2038
    [Length: 23]

0000  02 00 00 00 45 02 00 4b  41 3c 40 00 40 06 00 00   ....E..K A<@.@...
0010  7f 00 00 01 7f 00 00 01  04 d3 e8 92 19 d1 2a f1   ........ ......*.
0020  c1 36 86 5f 80 18 31 d7  fe 3f 00 00 01 01 08 0a   .6._..1. .?......
0030  3e f1 4d b0 3e f1 4d b0  4e 65 77 20 6e 75 6d 62   >.M.>.M. New numb
0040  65 72 20 67 65 6e 65 72  61 74 65 64 3a 20 38      er gener ated: 8
```

4. Client → Server / Send Ack

```
▸ Frame 11: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
▸ Null/Loopback
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 59538, Dst Port: 1235, Seq: 5, Ack: 24, Len: 0

0000  02 00 00 00 45 00 00 34  e3 d4 40 00 40 06 00 00   ....E..4 ..@.@...
0010  7f 00 00 01 7f 00 00 01  e8 92 04 d3 c1 36 86 5f   ........ .....6._
0020  19 d1 2b 08 80 10 31 d6  fe 28 00 00 01 01 08 0a   ..+...1. .(......
0030  3e f1 4d b0 3e f1 4d b0                           >.M.>.M.
```

Ack increase 23 (data length)

LAB 3 TCP operation (8)

(2) Two-way data flow

1. Client → Server / Send data (“0”)

```
Frame 8: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 59538, Dst Port: 1235, Seq: 1, Ack: 1, Len: 4
Data (4 bytes)
    Data: 00000000
    [Length: 4]

0000  02 00 00 00 45 02 00 38  99 7e 40 00 40 06 00 00   ....E..8 .~@.@...
0010  7f 00 00 01 7f 00 00 01  e8 92 04 d3 c1 36 86 5b   ........ .....6.[
0020  19 d1 2a f1 80 18 31 d7  fe 2c 00 00 01 01 08 0a   ..*...1. .,......
0030  3e f1 4d b0 3e f1 46 b1  00 00 00 00               >.M.>.F. ....
```

2. Server → Client / Send Ack

```
Frame 9: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 1235, Dst Port: 59538, Seq: 1, Ack: 5, Len: 0

0000  02 00 00 00 45 00 00 34  72 a3 40 00 40 06 00 00   ....E..4 r.@.@...
0010  7f 00 00 01 7f 00 00 01  04 d3 e8 92 19 d1 2a f1   ........ ......*.
0020  c1 36 86 5f 80 10 31 d7  fe 28 00 00 01 01 08 0a   .6._..1. .(......
0030  3e f1 4d b0 3e f1 4d b0                           >.M.>.M.
```

Ack increase 4 (data length)

3. Server → Client / Send data (“New number …”)

```
Frame 10: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 1235, Dst Port: 59538, Seq: 1, Ack: 5, Len: 23
Data (23 bytes)
    Data: 4e6577206e756d6265722067656e6572617465643a2038
    [Length: 23]

0000  02 00 00 00 45 02 00 4b  41 3c 40 00 40 06 00 00   ....E..K A<@.@...
0010  7f 00 00 01 7f 00 00 01  04 d3 e8 92 19 d1 2a f1   ........ ......*.
0020  c1 36 86 5f 80 18 31 d7  fe 3f 00 00 01 01 08 0a   .6._..1. .?......
0030  3e f1 4d b0 3e f1 4d b0  4e 65 77 20 6e 75 6d 62   >.M.>.M. New numb
0040  65 72 20 67 65 6e 65 72  61 74 65 64 3a 20 38      er gener ated: 8
```

4. Client → Server / Send Ack

```
Frame 11: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 59538, Dst Port: 1235, Seq: 5, Ack: 24, Len: 0

0000  02 00 00 00 45 00 00 34  e3 d4 40 00 40 06 00 00   ....E..4 ..@.@...
0010  7f 00 00 01 7f 00 00 01  e8 92 04 d3 c1 36 86 5f   ........ .....6._
0020  19 d1 2b 08 80 10 31 d6  fe 28 00 00 01 01 08 0a   ..+...1. .(......
0030  3e f1 4d b0 3e f1 4d b0                           >.M.>.M.
```

Ack increase 23 (data length)

Server Seq	1
Server Ack	1
Client Seq	1
Client Ack	1

1

Server Seq	1
Server Ack	5
Client Seq	1
Client Ack	1

2

Server Seq	1
Server Ack	5
Client Seq	5
Client Ack	23

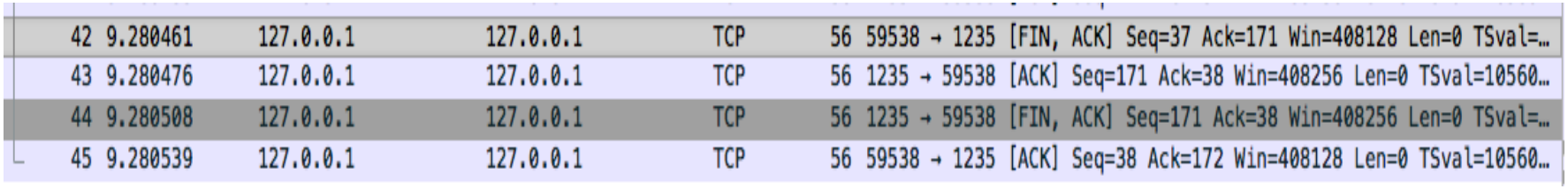
3

Server Seq	23
Server Ack	5
Client Seq	5
Client Ack	23

4

LAB 3 TCP operation (9)

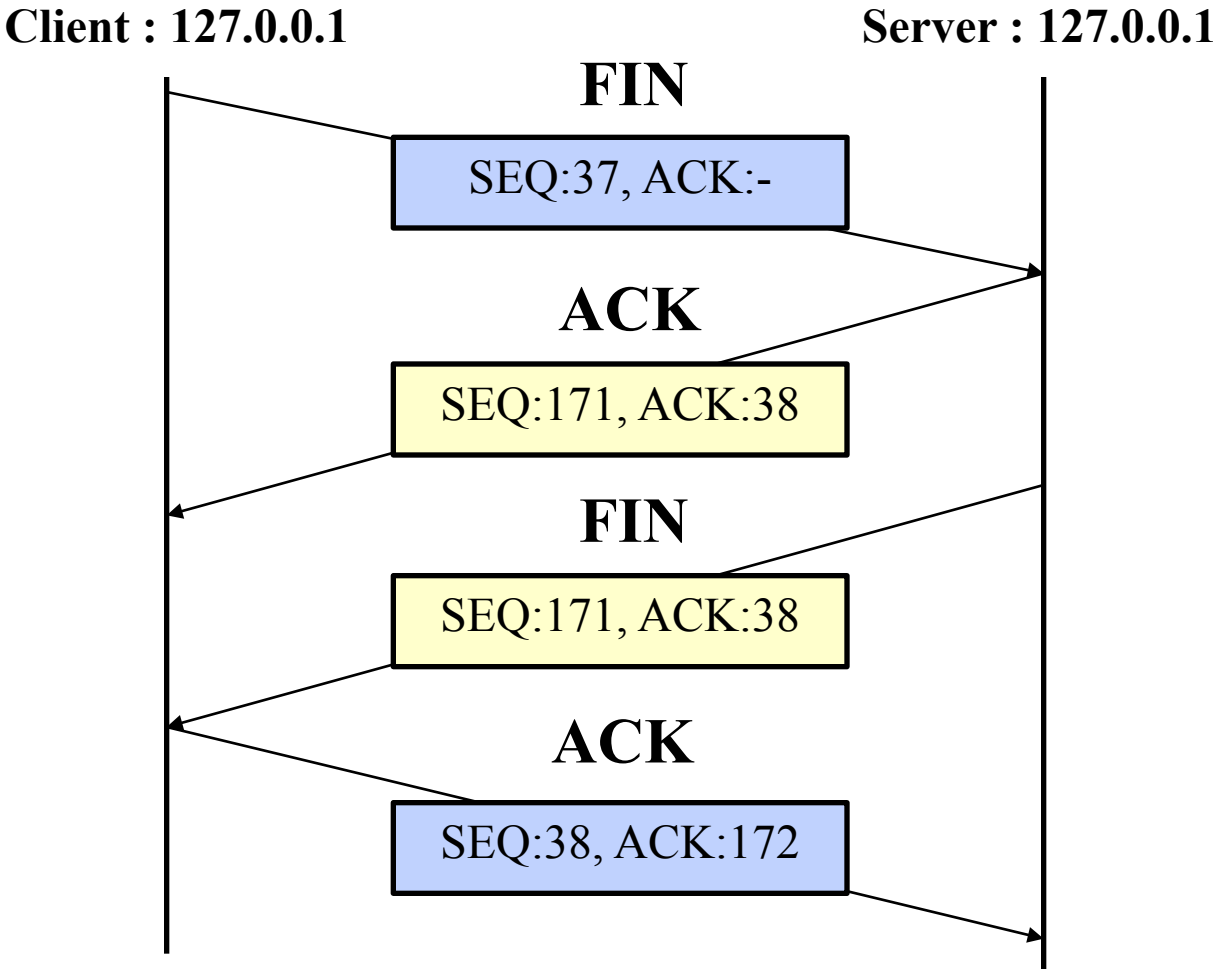
(3) Closing the connection



42	9.280461	127.0.0.1	127.0.0.1	TCP	56 59538 → 1235 [FIN, ACK] Seq=37 Ack=171 Win=408128 Len=0 TSval=…
43	9.280476	127.0.0.1	127.0.0.1	TCP	56 1235 → 59538 [ACK] Seq=171 Ack=38 Win=408256 Len=0 TSval=10560…
44	9.280508	127.0.0.1	127.0.0.1	TCP	56 1235 → 59538 [FIN, ACK] Seq=171 Ack=38 Win=408256 Len=0 TSval=…
45	9.280539	127.0.0.1	127.0.0.1	TCP	56 59538 → 1235 [ACK] Seq=38 Ack=172 Win=408128 Len=0 TSval=10560…

- Steps for closing TCP connection are like following.
- First, client sends FIN packets to server to notifying close of connection. Then, server sends ACK packets.
- Second, server sends FIN packets to client to notifying close of connection. Then, client sends ACK packets.

LAB 3 TCP operation (10)



Packet Analyzer - tcpdump

- tcpdump is a common packet analyzer that runs under the command line.
- It allows the user to display TCP/IP and other packets being transmitted or received over a network to which the computer is attached.
- Tcpdump works on most Unix-like operating systems. Tcpdump uses the libpcap library to capture packets.
- In WinDump, it uses WinPcap, the Windows port of libpcap.

Tcpdump packet capturing

- Use the -i option to select the network interface to analyzing
- **ifconfig** command allows you to check the available network interfaces

< available network interfaces >

```
SeongHwanKimui-MacBook-Air:~ wody34$ ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
        options=1203<RXCSUM,TXCSUM,TXSTATUS,SW_TIMESTAMP>
        inet 127.0.0.1 netmask 0xff000000
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
        nd6 options=201<PERFORMNUD,DAD>
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
stf0: flags=0<> mtu 1280
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        ether 9c:f3:87:cc:ed:4a
        inet6 fe80::c1a:fede:e8cb:9552%en0 prefixlen 64 secured scopeid 0x4
        inet 10.0.1.19 netmask 0xffffff00 broadcast 10.0.1.255
        nd6 options=201<PERFORMNUD,DAD>
        media: autoselect
        status: active
en1: flags=963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX> mtu 1500
        options=60<TSO4,TSO6>
        ether 32:00:1e:68:40:00
        media: autoselect <full-duplex>
        status: inactive
```

< Example of capturing packets on en0 interface >

```
SeongHwanKimui-MacBook-Air:~ wody34$ sudo tcpdump -i en0
Password:
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on en0, link-type EN10MB (Ethernet), capture size 262144 bytes
23:30:48.686937 IP 10.0.1.19.53485 > ec2-54-251-183-39.ap-southeast-1.compute.a
57297445], length 63
23:30:48.700862 IP 10.0.1.19.64403 > 10.0.1.1.domain: 34228+ PTR? 19.1.0.10.in-
23:30:48.705983 IP 10.0.1.1.domain > 10.0.1.19.64403: 34228 NXDomain* 0/0/0 (40
23:30:48.707409 IP 10.0.1.19.51856 > 10.0.1.1.domain: 53253+ PTR? 39.183.251.54
23:30:48.779682 IP 10.0.1.19.53485 > ec2-54-251-183-39.ap-southeast-1.compute.a
23:30:48.805173 IP ec2-54-251-183-39.ap-southeast-1.compute.amazonaws.com.https
23:30:48.885555 IP ec2-54-251-183-39.ap-southeast-1.compute.amazonaws.com.https
gth 0
23:30:49.043368 IP ec2-54-251-183-39.ap-southeast-1.compute.amazonaws.com.https
23:30:49.043479 IP 10.0.1.19.53485 > ec2-54-251-183-39.ap-southeast-1.compute.a
23:30:49.218697 IP 10.0.1.1.domain > 10.0.1.19.51856: 53253 1/0/0 PTR ec2-54-25
23:30:49.223221 IP 10.0.1.19.57054 > 10.0.1.1.domain: 47260+ PTR? 1.1.0.10.in-a
23:30:49.226005 IP 10.0.1.1.domain > 10.0.1.19.57054: 47260 NXDomain* 0/0/0 (39
23:30:49.834902 IP ec2-54-251-183-39.ap-southeast-1.compute.amazonaws.com.https
23:30:49.835054 IP 10.0.1.19.53485 > ec2-54-251-183-39.ap-southeast-1.compute.a
23:30:49.967288 IP 10.0.1.19.50363 > 10.0.1.1.domain: 55355+ A? play.google.com
23:30:49.984917 IP 10.0.1.1.domain > 10.0.1.19.50363: 55355 2/0/0 CNAME play.l.
23:30:49.989129 IP 10.0.1.19.53800 > hkg07s02-in-f14.1e100.net.https: UDP, leng
23:30:49.989888 IP 10.0.1.19.55799 > 10.0.1.1.domain: 40114+ PTR? 142.221.58.21
23:30:50.002886 IP 10.0.1.1.domain > 10.0.1.19.55799: 40114 2/0/0 PTR hkg07s02-
```

Tcpdump packet filtering options

- # tcpdump host A
  ; A Output all incoming / outgoing packets to / from host
- # tcpdump host A and \(B or C \)
  ; All traffic output between host A and B or C
- # tcpdump -i eth0 tcp port 80
  ; Packets forwarded through tcp port 80 via eth0 interface
- # tcpdump src x.x.x.x and not dst port 22
  ; All packets with src ip x.x.x.x and dst port not 22
- More details on lab supplement (tcpdump man page - https://github.com/wody34/ee614/blob/master/LabMaterials/Lab3/TCPDUMP Man Page.pdf)

Tcpdump - (dump to / read from) file options

- # tcpdump -w tcpdump.log
  ; Dump packet data to log file in binary format (not text format)
- # tcpdump -Xnr tcpdump.log
  ; Read packet data from log file with printing the data of each packet in hex and ASCII. Don't convert addresses to names.

```
22:32:54.876902 IP 127.0.0.1.52875 > 127.0.0.1.12345: Flags [SEW], seq 3150304300, win 65535, options [mss 16344,nop,wscale 5,nop,nop,TS val 1059417287 ecr
        0x0000:  4500 0040 3e83 4000 4006 0000 7f00 0001  E..@>.@.@.......
        0x0010:  7f00 0001 ce8b 3039 bbc5 d42c 0000 0000  ......09...,....
        0x0020:  b0c2 ffff fe34 0000 0204 3fd8 0103 0305  .....4....?.....
        0x0030:  0101 080a 3f25 6cc7 0000 0000 0402 0000  ....?%l.........
22:32:54.876990 IP 127.0.0.1.12345 > 127.0.0.1.52875: Flags [S.], seq 3968470670, ack 3150304301, win 65535, options [mss 16344,nop,wscale 5,nop,nop,TS val
        0x0000:  4500 0040 dc8a 4000 4006 0000 7f00 0001  E..@..@.@.......
        0x0010:  7f00 0001 3039 ce8b ec8a 0e8e bbc5 d42d  ....09.........-
        0x0020:  b012 ffff fe34 0000 0204 3fd8 0103 0305  .....4....?.....
        0x0030:  0101 080a 3f25 6cc7 3f25 6cc7 0402 0000  ....?%l.?%l.....
22:32:54.877006 IP 127.0.0.1.52875 > 127.0.0.1.12345: Flags [.], ack 1, win 12759, options [nop,nop,TS val 1059417287 ecr 1059417287], length 0
        0x0000:  4500 0034 ff7a 4000 4006 0000 7f00 0001  E..4.z@.@.......
        0x0010:  7f00 0001 ce8b 3039 bbc5 d42d ec8a 0e8f  ......09...-....
        0x0020:  8010 31d7 fe28 0000 0101 080a 3f25 6cc7  ..1..(......?%l.
        0x0030:  3f25 6cc7                                ?%l.
22:32:54.877019 IP 127.0.0.1.12345 > 127.0.0.1.52875: Flags [.], ack 1, win 12759, options [nop,nop,TS val 1059417287 ecr 1059417287], length 0
        0x0000:  4500 0034 87ef 4000 4006 0000 7f00 0001  E..4..@.@.......
        0x0010:  7f00 0001 3039 ce8b ec8a 0e8f bbc5 d42d  ....09.........-
        0x0020:  8010 31d7 fe28 0000 0101 080a 3f25 6cc7  ..1..(......?%l.
        0x0030:  3f25 6cc7                                ?%l.
22:32:56.098098 IP 127.0.0.1.52875 > 127.0.0.1.12345: Flags [P.], seq 1:5, ack 1, win 12759, options [nop,nop,TS val 1059418505 ecr 1059417287], length 4
        0x0000:  4500 0038 1bfe 4000 4006 0000 7f00 0001  E..8..@.@.......
        0x0010:  7f00 0001 ce8b 3039 bbc5 d42d ec8a 0e8f  ......09...-....
        0x0020:  8018 31d7 fe2c 0000 0101 080a 3f25 7189  ..1..,......?%q.
        0x0030:  3f25 6cc7 0000 0000                      ?%l.....
22:32:56.098138 IP 127.0.0.1.12345 > 127.0.0.1.52875: Flags [.], ack 5, win 12759, options [nop,nop,TS val 1059418505 ecr 1059418505], length 0
        0x0000:  4500 0034 c89e 4000 4006 0000 7f00 0001  E..4..@.@.......
        0x0010:  7f00 0001 3039 ce8b ec8a 0e8f bbc5 d431  ....09.........1
        0x0020:  8010 31d7 fe28 0000 0101 080a 3f25 7189  ..1..(......?%q.
        0x0030:  3f25 7189                                ?%q.
22:32:56.098242 IP 127.0.0.1.12345 > 127.0.0.1.52875: Flags [P.], seq 1:25, ack 5, win 12759, options [nop,nop,TS val 1059418505 ecr 1059418505], length 24
        0x0000:  4502 004c d910 4000 4006 0000 7f00 0001  E..L..@.@.......
        0x0010:  7f00 0001 3039 ce8b ec8a 0e8f bbc5 d431  ....09.........1
        0x0020:  8018 31d7 fe40 0000 0101 080a 3f25 7189  ..1..@......?%q.
        0x0030:  3f25 7189 4e65 7720 6e75 6d62 6572 2067  ?%q.New.number.g
        0x0040:  656e 6572 6174 6564 3a20 3131            enerated:.11
```