**Ch.Waleed Ishtiaq**

**Lab 09**

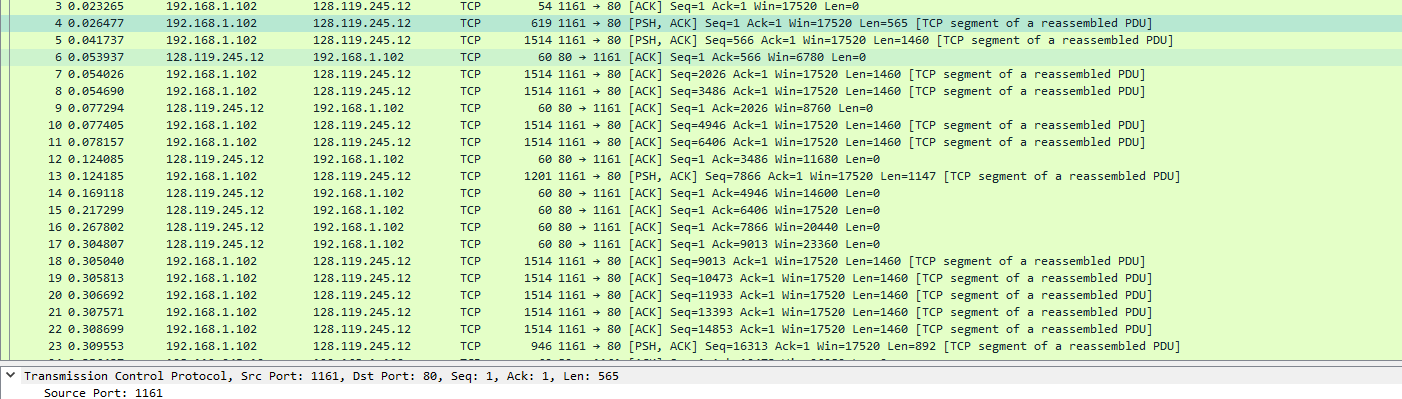
**Lab Statement 1: Analyzing TCP Packets using Wireshark (10)**

**Question 1:**

**Answer:**

IP address = 192.168.1.102

Port Number = 1161

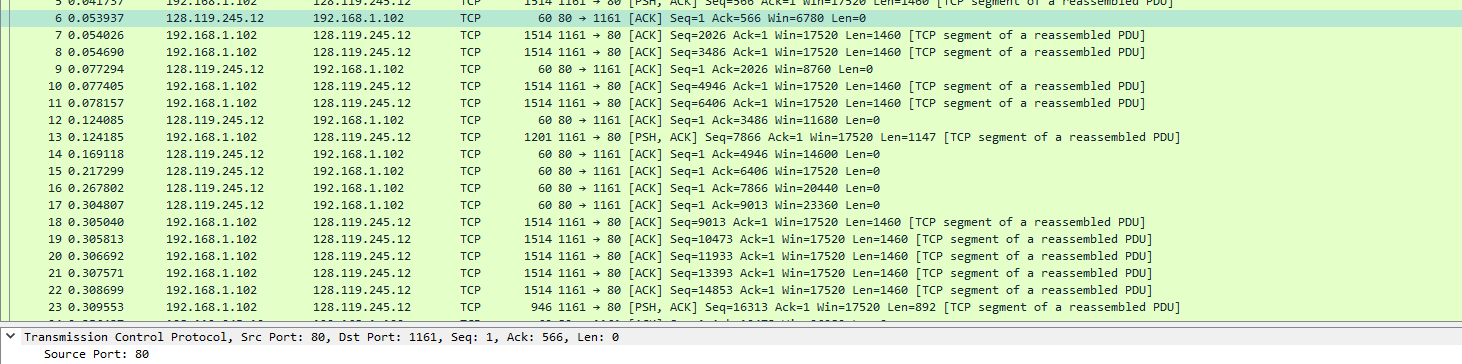


**Question 2:**

**Answer:**

IP address = 128.119.245.12

Port = 80

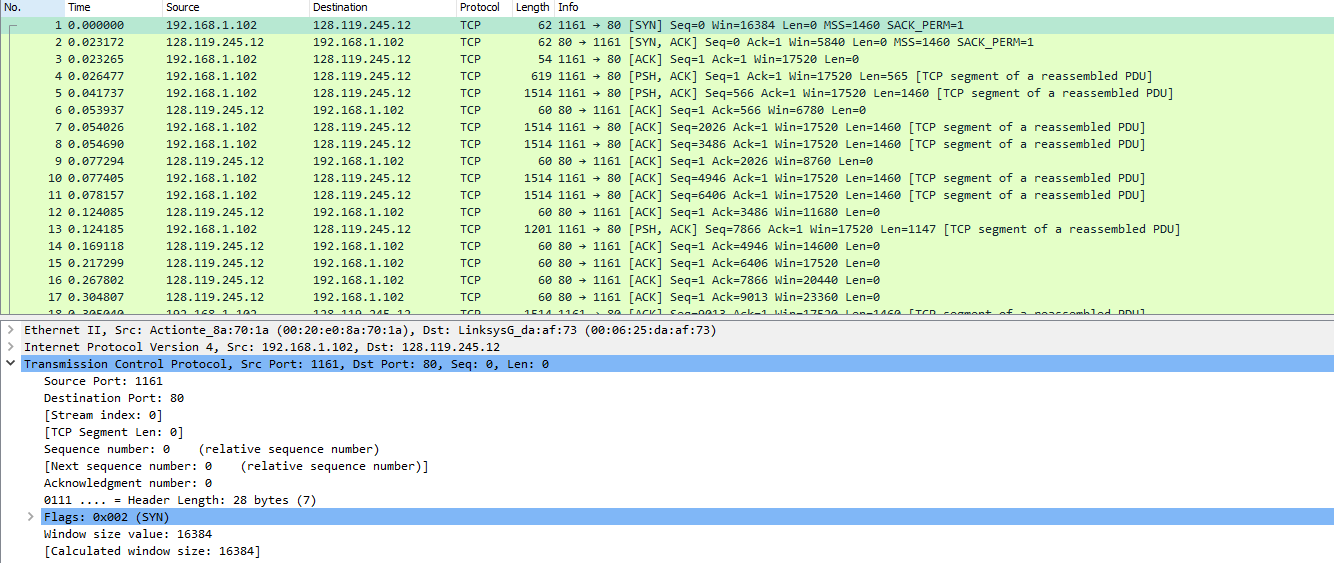


**Question 3:**

**Answer:**

Sequence Number = 0;

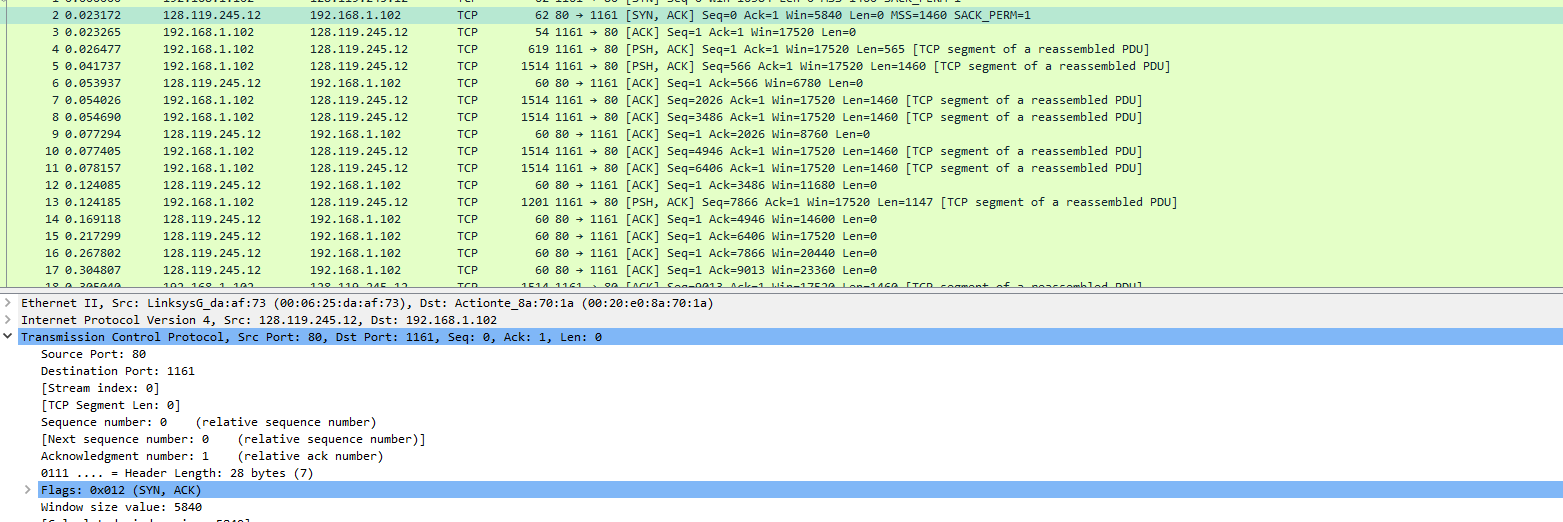
Flags in the segment tells us that this is an SYN segment.



**Question 4:**

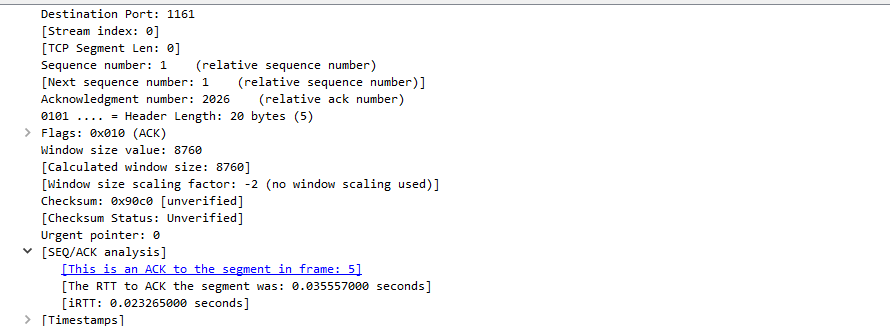
**Answer**: Sequence Number = 0, Ack number = 1,

Flags in the segment tells us that this is an SYNACK segment.



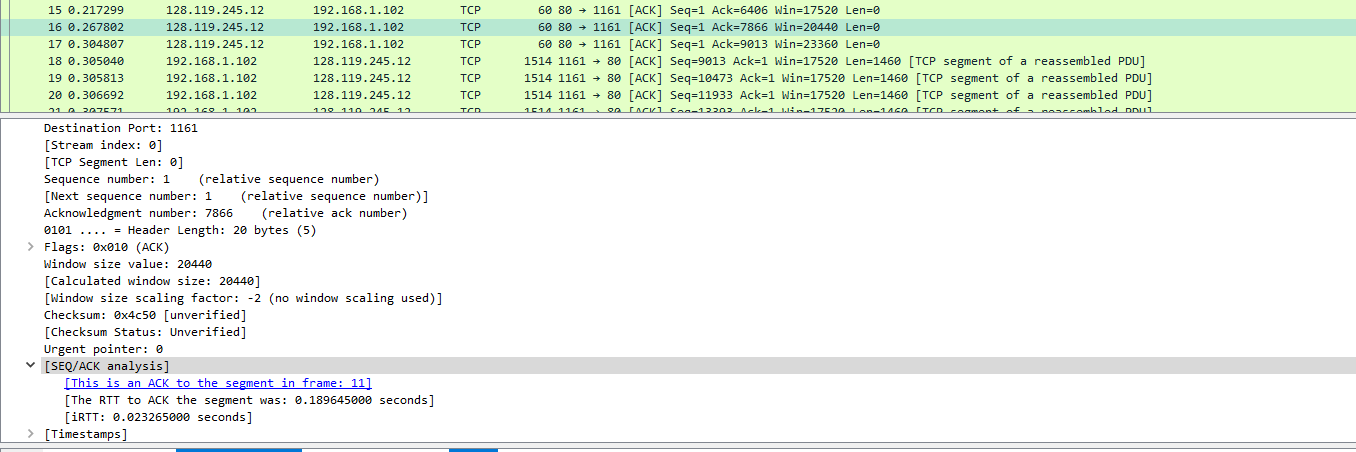
**Question 5:**

**Answer**: Because server is replying to frame 5, so the addition of sequence number of frame 5 and its length is ACK number of packet 9.



**Question 6:**

**Answer**: Because server is replying to frame 11, so the addition of sequence number of frame 11 and its length is ACK number of packet 16.



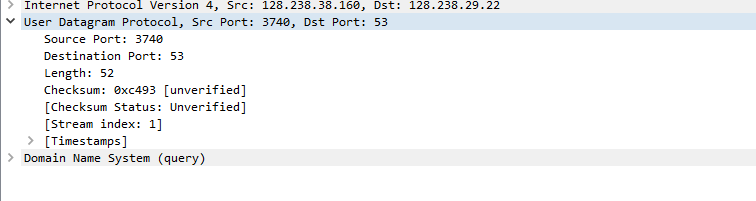
**Question 7:**

**Answer:** To make it easy to read and compare.

**Lab Statement 2: Analyzing UDP Packets using Wireshark (5)**

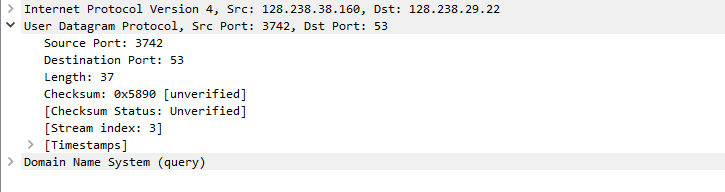
**Question 1 (Statement-2):**

**Answer:** Four fields



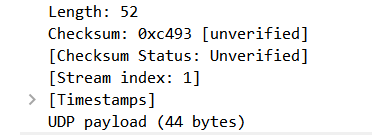
**Question 2 (Statement-2):**

**Answer**: 8 Bytes



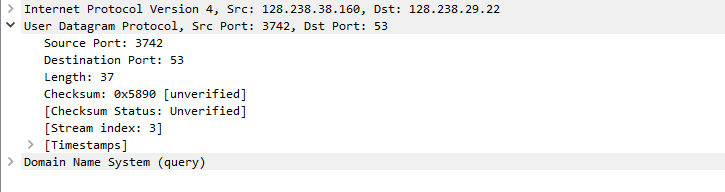
**Question 3:**

**Answer**: The value in the Length field is the length of whole packet including header and data, as UDP payload is 44 bytes and header length is 8 bytes… So length should be equal to 52 bytes which is mentioned in the UDP packet’s header.



**Question 4:**

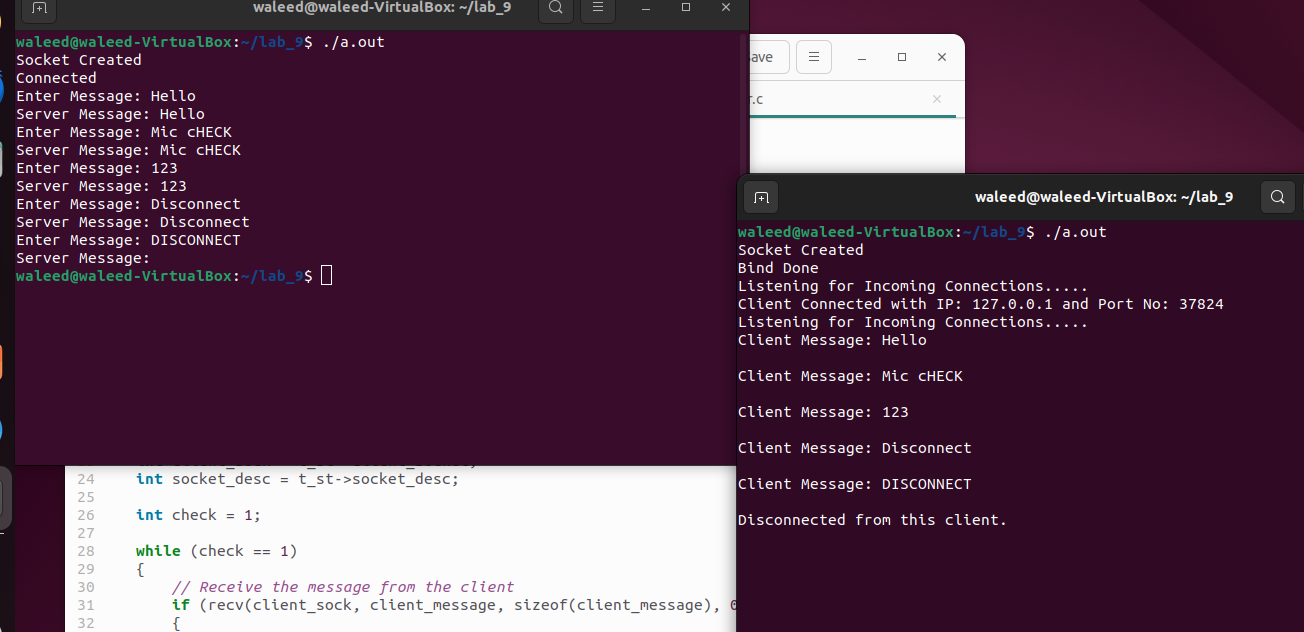
**Answer:** Port Number = 53.



**Lab Statement 3: Multithreaded ECHO server using TCP**

**(10)**

**Screen Shot:**

****

**Server:**

/\*

TCP\_Server. This Program will will create the Server side for TCP\_Socket Programming.

It will receive the data from the client and then send the same data back to client.

\*/

#include <stdio.h>

#include <pthread.h>

#include <string.h>

#include <stdlib.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#define MAX\_THREADS 3

struct Rec\_data

{

int client\_socket;

int socket\_desc;

};

void \*print\_thread\_info(void \*rec\_data)

{

char client\_message[2000], server\_message[2000];

memset(client\_message, '\0', sizeof(client\_message));

struct Rec\_data \*t\_st = (struct Rec\_data \*)rec\_data;

int client\_sock = t\_st->client\_socket;

int socket\_desc = t\_st->socket\_desc;

int check = 1;

while (check == 1)

{

// Receive the message from the client

if (recv(client\_sock, client\_message, sizeof(client\_message), 0) < 0)

{

printf("Receive Failed. Error!!!!!\n");

break; // exit the loop if receive fails

}

printf("Client Message: %s\n\n", client\_message);

if (strcmp(client\_message, "DISCONNECT") != 0)

{

// Send the message back to the client

strcpy(server\_message, client\_message);

if (send(client\_sock, server\_message, strlen(client\_message), 0) < 0)

{

printf("Send Failed. Error!!!!!\n");

break; // exit the loop if send fails

}

memset(server\_message, '\0', sizeof(server\_message));

memset(client\_message, '\0', sizeof(client\_message));

}

else

{

check = 0;

}

}

printf("Disconnected from this client.\n\n");

close(client\_sock);

close(socket\_desc);

free(rec\_data); // Free allocated memory for Rec\_data structure

pthread\_exit(NULL);

}

int main(void)

{

int socket\_desc, client\_sock, client\_size;

struct sockaddr\_in server\_addr, client\_addr;

char server\_message[2000], client\_message[2000];

memset(server\_message, '\0', sizeof(server\_message));

memset(client\_message, '\0', sizeof(client\_message));

socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);

if (socket\_desc < 0)

{

printf("Could Not Create Socket. Error!!!!!\n");

return -1;

}

printf("Socket Created\n");

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(2000);

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

if (bind(socket\_desc, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0)

{

printf("Bind Failed. Error!!!!!\n");

return -1;

}

printf("Bind Done\n");

pthread\_t threads[MAX\_THREADS];

int t\_in = 0;

while (1)

{

if (listen(socket\_desc, 1) < 0)

{

printf("Listening Failed. Error!!!!!\n");

return -1;

}

printf("Listening for Incoming Connections.....\n");

client\_size = sizeof(client\_addr);

client\_sock = accept(socket\_desc, (struct sockaddr \*)&client\_addr, &client\_size);

if (client\_sock < 0)

{

printf("Accept Failed. Error!!!!!!\n");

return -1;

}

printf("Client Connected with IP: %s and Port No: %i\n", inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

if (t\_in < MAX\_THREADS)

{

struct Rec\_data \*data = (struct Rec\_data \*)malloc(sizeof(struct Rec\_data));

data->client\_socket = client\_sock;

data->socket\_desc = socket\_desc;

pthread\_create(&threads[t\_in], NULL, print\_thread\_info, data);

pthread\_detach(threads[t\_in]); // Detach the thread to avoid memory leaks

++t\_in;

}

else

{

if (send(client\_sock, "Server Full.", strlen("Server Full."), 0) < 0)

{

printf("Send Failed. Error!!!!!\n");

}

close(client\_sock);

close(socket\_desc);

break;

}

}

// Closing the Socket

close(socket\_desc);

return 0;

}

**Client:**

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#define MAX\_MESSAGE\_SIZE 2000

int main(void)

{

int socket\_desc;

struct sockaddr\_in server\_addr;

char server\_message[MAX\_MESSAGE\_SIZE], client\_message[MAX\_MESSAGE\_SIZE];

memset(server\_message, '\0', sizeof(server\_message));

memset(client\_message, '\0', sizeof(client\_message));

// Creating Socket

socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);

if (socket\_desc < 0)

{

printf("Could Not Create Socket. Error!!!!!\n");

return -1;

}

printf("Socket Created\n");

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(2000);

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

if (connect(socket\_desc, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0)

{

printf("Connection Failed. Error!!!!!\n");

return -1;

}

printf("Connected\n");

int check = 1;

while (check == 1)

{

printf("Enter Message: ");

fgets(client\_message, sizeof(client\_message), stdin);

// Remove newline character from the input

client\_message[strcspn(client\_message, "\n")] = '\0';

if (strcmp(client\_message, "DISCONNECT") == 0)

{

check = 0;

}

if (send(socket\_desc, client\_message, strlen(client\_message), 0) < 0)

{

printf("Send Failed. Error!!!!\n");

return -1;

}

// Receive the message back from the server

if (recv(socket\_desc, server\_message, sizeof(server\_message), 0) < 0)

{

printf("Receive Failed. Error!!!!!\n");

return -1;

}

printf("Server Message: %s\n", server\_message);

memset(server\_message, '\0', sizeof(server\_message));

memset(client\_message, '\0', sizeof(client\_message));

}

// Closing the Socket

close(socket\_desc);

return 0;

}