

**Assignment 1
Of
Network & Distributed System Lab (CS2051)
Masters of Technology in Computer Science And Engineering**

submitted to
Dr Sujoy Saha
Assistant Professor
&
Dr Suvrojit Das
Associate Professor
Dept. of CSE



National Institute of Technology, Durgapur

submitted by
Arghya Bandyopadhyay
RollNo. 20CS4103

27th March 2021

1. Write simple TCP and UDP program using socket API which will transfer simple text messages, and check TCP and UDP packets using Wireshark.

Answer.

The following code is the implementation for **TCP client side programming**.

```
/* tcpclient.c */

#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>

int main()
{
    int sock, bytes_recieved;
    char send_data[1024], recv_data[1024];
    struct hostent *host;
    struct sockaddr_in server_addr;

    host = gethostbyname("127.0.0.1");

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("Socket");
        exit(1);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5000);
    server_addr.sin_addr = *((struct in_addr *)host->h_addr);
    bzero(&(server_addr.sin_zero),8);

    if (connect(sock, (struct sockaddr *)&server_addr,
                sizeof(struct sockaddr)) == -1)
    {
```

```

    perror("Connect");
    exit(1);
}

while(1)
{
    bytes_recieved=recv(sock ,recv_data ,1024 ,0);
    recv_data[bytes_recieved] = '\0';

    if (strcmp(recv_data , "q") == 0 || strcmp(recv_data , "Q") == 0)
    {
        close(sock );
        break;
    }

    else
        printf("\nReceived data =%s " , recv_data);

    printf("\nSEND (q or Q to quit) : ");
    gets(send_data);

    if (strcmp(send_data , "q") != 0 && strcmp(send_data , "Q") != 0)
        send(sock ,send_data ,strlen(send_data) , 0);

    else
    {
        send(sock ,send_data ,strlen(send_data) , 0);
        close(sock );
        break;
    }
}

return 0;
}

```

The following code is the implementation for **TCP Server side programming**.

```

/* tcpserver.c */

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

```

```

#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>

int main()
{
    int sock , connected , bytes_recieved , true = 1;
    char send_data [1024] , recv_data[1024];

    struct sockaddr_in server_addr , client_addr ;
    int sin_size;

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("Socket");
        exit(1);
    }

    if (setsockopt(sock ,SOL_SOCKET,SO_REUSEADDR,&true ,sizeof(int)) == -1) {
        perror("Setsockopt");
        exit(1);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5000);
    server_addr.sin_addr.s_addr = INADDR_ANY;
    bzero(&(server_addr.sin_zero) ,8);

    if (bind(sock , (struct sockaddr *)&server_addr , sizeof(struct sockaddr))
        == -1) {
        perror("Unable to bind");
        exit(1);
    }

    if (listen(sock , 5) == -1) {
        perror("Listen");
        exit(1);
    }
}

```

```

printf("\nTCPserver Waiting for client on port 5000");
fflush(stdout);

while(1)
{
    sin_size = sizeof(struct sockaddr_in);
    connected = accept(sock, (struct sockaddr *)&client_addr,&sin_size);
    printf("\n I got a connection from (%s , %d)",
           inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
    while (1)
    {
        printf("\n SEND (q or Q to quit) : ");
        gets(send_data);
        if (strcmp(send_data , "q") == 0 || strcmp(send_data , "Q") == 0)
        {
            send(connected , send_data , strlen(send_data) , 0);
            close(connected);
            break;
        }
        else
            send(connected , send_data , strlen(send_data) , 0);
        bytes_recieved = recv(connected , recv_data ,1024 ,0);
        recv_data[bytes_recieved] = '\0';
        if (strcmp(recv_data , "q") == 0 || strcmp(recv_data , "Q") == 0)
        {
            close(connected);
            break;
        }
        else
            printf("\n RECEIVED DATA = %s " , recv_data);
    }
}

```

```
        fflush( stdout );
    }

    close( sock );
    return 0;
}
```

The image shows two terminal windows side-by-side. The left terminal window, titled 'arghya@Delton: ~/My Work/Networks/Simple Socket Pro...', displays the output of a TCP server application. It shows the server waiting for a connection on port 5000, accepting a connection from (127.0.0.1, 39350), sending 'Hello world.' to the client, receiving 'Hello world too.' back, and then exiting with a 'q' command. The right terminal window, also titled 'arghya@Delton: ~/My Work/Networks/Simple Socket Programming\$', displays the output of a TCP client application. It shows the client receiving 'Hello world.' from the server, sending 'Hello world too.' back to the server, and then exiting with a 'q' command.

```
arghya@Delton:~/My Work/Networks/Simple Socket Programming$ ./tcpserver
TCP Server Waiting for client on port 5000
I got a connection from (127.0.0.1 , 39350)
SEND (q or Q to quit) : Hello world.

RECIEVED DATA = Hello world too.
SEND (q or Q to quit) : q
```

```
arghya@Delton:~/My Work/Networks/Simple Socket Programming$ ./tcpclient
Recieved data = Hello world.
SEND (q or Q to quit) : Hello world too.
arghya@Delton:~/My Work/Networks/Simple Socket Programming$
```

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

(ip.addr eq 127.0.0.1 and ip.addr eq 127.0.0.1) and (tcp.port eq 39350 and tcp.port eq 5000)

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	39350 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1522219206 TSecr=0 WS=128
2	0.000017941	127.0.0.1	127.0.0.1	TCP	74	5000 → 39350 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1522219206 TSecr=1522219206 WS
3	0.000032727	127.0.0.1	127.0.0.1	TCP	66	39350 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1522219206 TSecr=1522219206
4	11.790191721	127.0.0.1	127.0.0.1	TCP	78	5000 → 39350 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=12 TSval=1522230996 TSecr=1522219206
5	11.790267997	127.0.0.1	127.0.0.1	TCP	66	39350 → 5000 [ACK] Seq=1 Ack=13 Win=65536 Len=0 TSval=1522230996 TSecr=1522230996
6	23.285131697	127.0.0.1	127.0.0.1	TCP	82	39350 → 5000 [PSH, ACK] Seq=1 Ack=13 Win=65536 Len=16 TSval=1522242491 TSecr=1522230996
7	23.285155812	127.0.0.1	127.0.0.1	TCP	66	5000 → 39350 [ACK] Seq=13 Ack=17 Win=65536 Len=0 TSval=1522242491 TSecr=1522242491
8	43.529719446	127.0.0.1	127.0.0.1	TCP	67	5000 → 39350 [PSH, ACK] Seq=13 Ack=17 Win=65536 Len=1 TSval=1522262735 TSecr=1522242491
9	43.529795113	127.0.0.1	127.0.0.1	TCP	66	39350 → 5000 [ACK] Seq=17 Ack=14 Win=65536 Len=0 TSval=1522262736 TSecr=1522262735
10	43.529867072	127.0.0.1	127.0.0.1	TCP	66	5000 → 39350 [FIN, ACK] Seq=14 Ack=17 Win=65536 Len=0 TSval=1522262736 TSecr=1522262736
11	43.529989756	127.0.0.1	127.0.0.1	TCP	66	39350 → 5000 [FIN, ACK] Seq=17 Ack=15 Win=65536 Len=0 TSval=1522262736 TSecr=1522262736
12	43.530036582	127.0.0.1	127.0.0.1	TCP	66	5000 → 39350 [ACK] Seq=15 Ack=18 Win=65536 Len=0 TSval=1522262736 TSecr=1522262736

Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface lo, id 0

Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 39350, Dst Port: 5000, Seq: 0, Len: 0

Source Port: 39350
 Destination Port: 5000
 [Stream index: 0]
 [TCP Segment Len: 0]
 Sequence number: 0 (relative sequence number)
 Sequence number (raw): 2816746891
 [Next sequence number: 1 (relative sequence number)]
 Acknowledgment number: 0
 Acknowledgment number (raw): 0
 1010 = Header Length: 40 bytes (10)
 Flags: 0x002 (SYN)
 000.... = Reserved: Not set
 ...0.... =Nonce: Not set
0... = Congestion Window Reduced (CWR): Not set
0... = ECN-Echo: Not set
0... = Urgent: Not set
0... = Acknowledgment: Not set
0... = Push: Not set

0020 00 01 99 b6 13 88 a7 e4 25 8b 00 00 00 00 a0 02%.....

This shows the raw value of the sequence number (tcp.seq_raw), 4 bytes

Packets: 14 · Displayed: 12 (85.7%)

Profile: Default

The following code is the implementation for **UDP Client side programming**.

```
/* udpclient.c */

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

int main()
{
    int sock;
    struct sockaddr_in server_addr;
    struct hostent *host;
    char send_data[1024];

    host= (struct hostent *) gethostbyname((char *)"127.0.0.1");

    if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
    {
        perror("socket");
        exit(1);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5000);
    server_addr.sin_addr = *((struct in_addr *)host->h_addr);
    bzero(&(server_addr.sin_zero),8);

    while (1)
    {
        printf("Type Something (q or Q to quit):");
        gets(send_data);
```

```

        if ((strcmp(send_data , "q") == 0) || strcmp(send_data , "Q") == 0)
            break;

        else
            sendto(sock , send_data , strlen(send_data) , 0 ,
                   (struct sockaddr *)&server_addr , sizeof(struct sockaddr));

    }

}

```

The following code is the implementation for **UDP Server side programming**.

```

/* udpserver.c */

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

int main()
{
    int sock;
    int addr_len , bytes_read;
    char recv_data[1024];
    struct sockaddr_in server_addr , client_addr;

    if ((sock = socket(AF_INET , SOCK_DGRAM, 0)) == -1) {
        perror("Socket");
        exit(1);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(5000);
    server_addr.sin_addr.s_addr = INADDR_ANY;
    bzero(&(server_addr.sin_zero) ,8);

```

```

if (bind(sock,( struct sockaddr *)&server_addr ,
         sizeof( struct sockaddr )) == -1)
{
    perror ("Bind");
    exit(1);
}

addr_len = sizeof( struct sockaddr );

printf ("\nUDPServer Waiting for client on port 5000");
fflush (stdout);

while (1)
{
    bytes_read = recvfrom (sock ,recv_data ,1024 ,0 ,
                          ( struct sockaddr *)&client_addr , &addr_len );

    recv_data [bytes_read] = '\0';

    printf ("\n(%s , %d) said : " ,inet_ntoa (client_addr.sin_addr) ,
           ntohs (client_addr.sin_port));
    printf ("%s" , recv_data );
    fflush (stdout);

}
return 0;
}

```

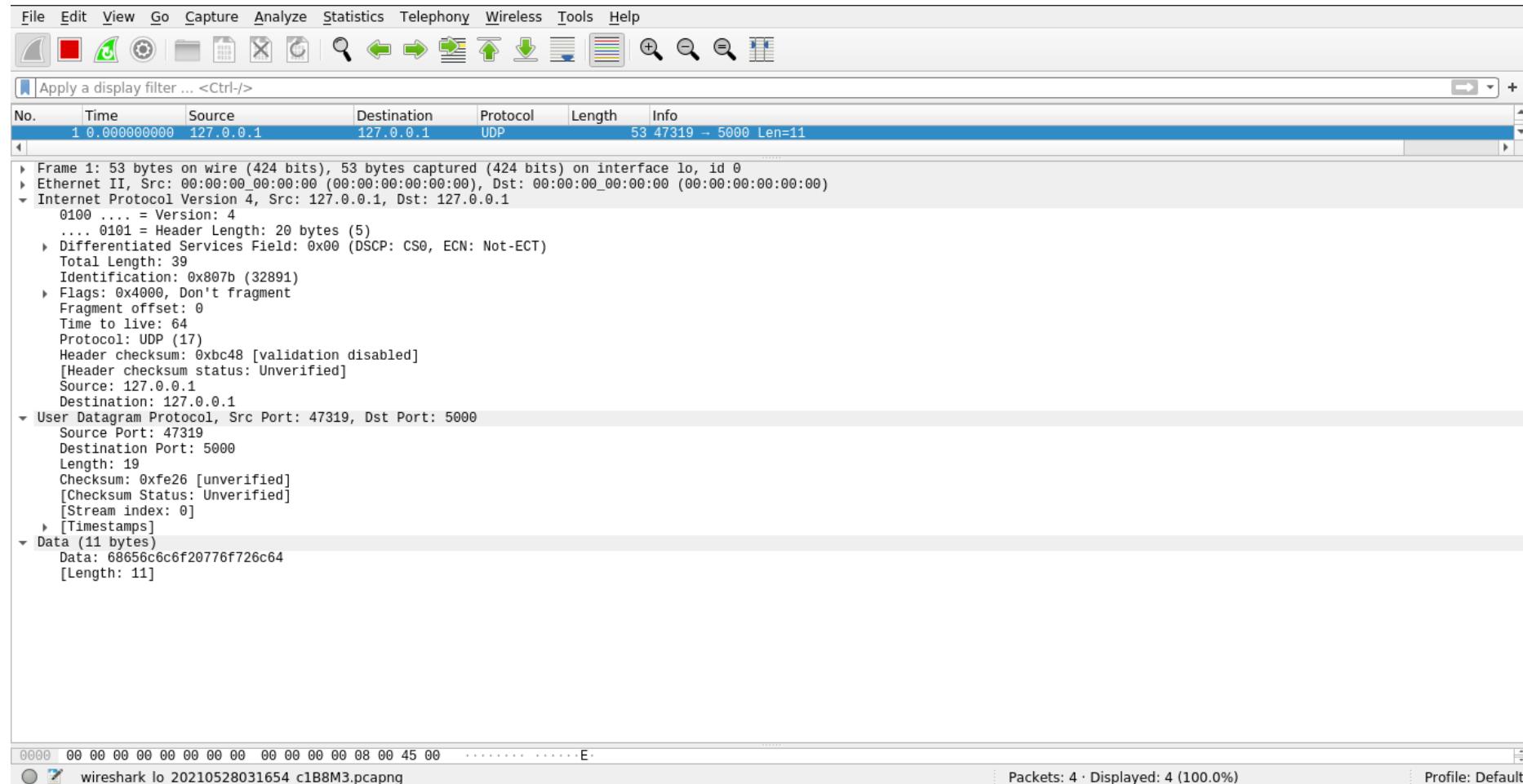
The image shows two terminal windows side-by-side, both titled "arghya@Delton: ~/My Work/Networks/Simple Socket Pro...".

The left terminal window displays the output of the `./udpserver` program:

```
No arghya@Delton:~/My Work/Networks/Simple Socket Programming$ ./udpserver
1 0.000000000 127.0.0.1 127.0.0.1 UDP 53 47319 -> 5000
UDPServer Waiting for client on port 5000
(127.0.0.1 , 47319) said : hello world
```

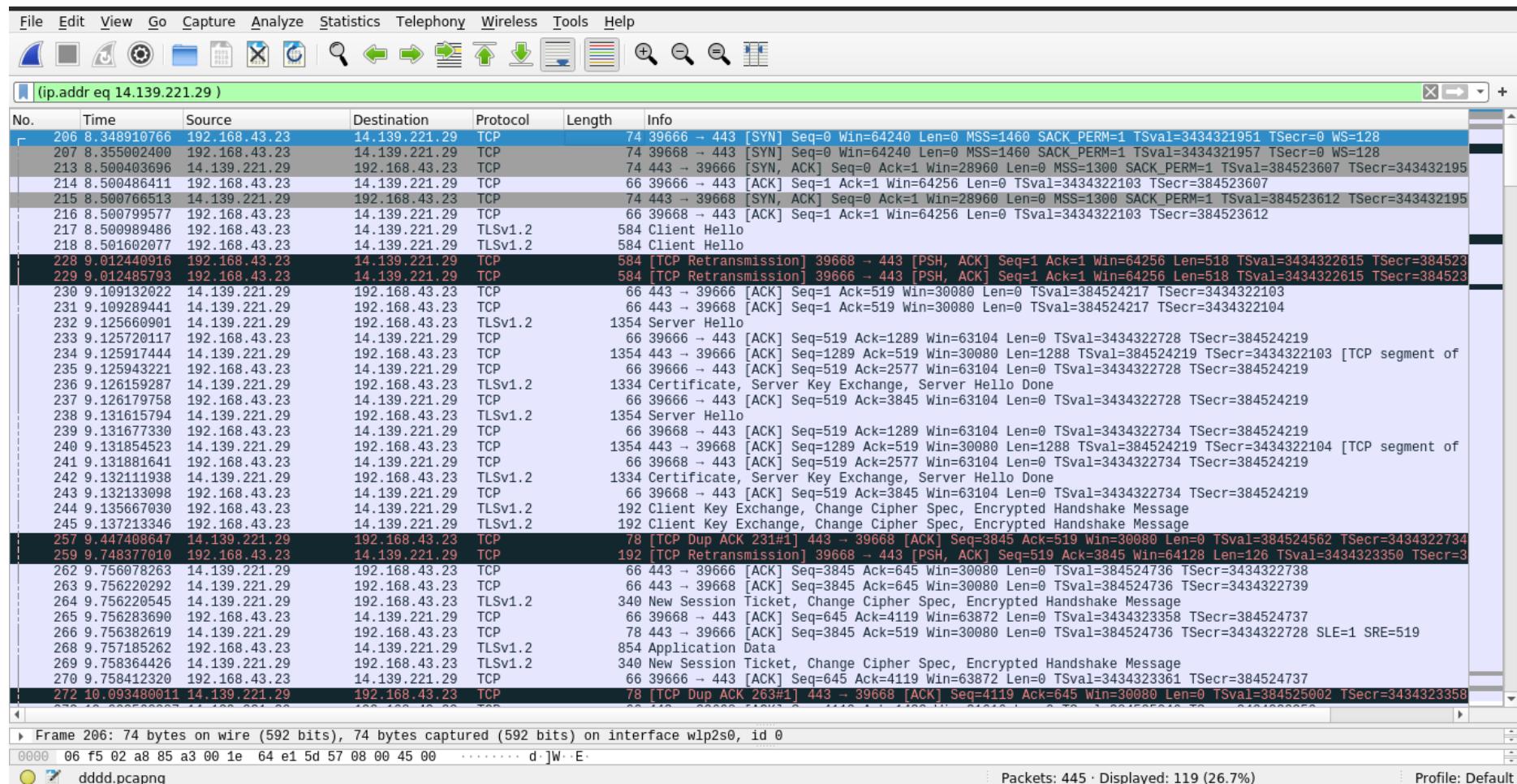
The right terminal window displays the output of the `./udpclient` program:

```
arghya@Delton:~/My Work/Networks/Simple Socket Programming$ ./udpclient
Type Something (q or Q to quit):hello world
Type Something (q or Q to quit):q
arghya@Delton:~/My Work/Networks/Simple Socket Programming$
```

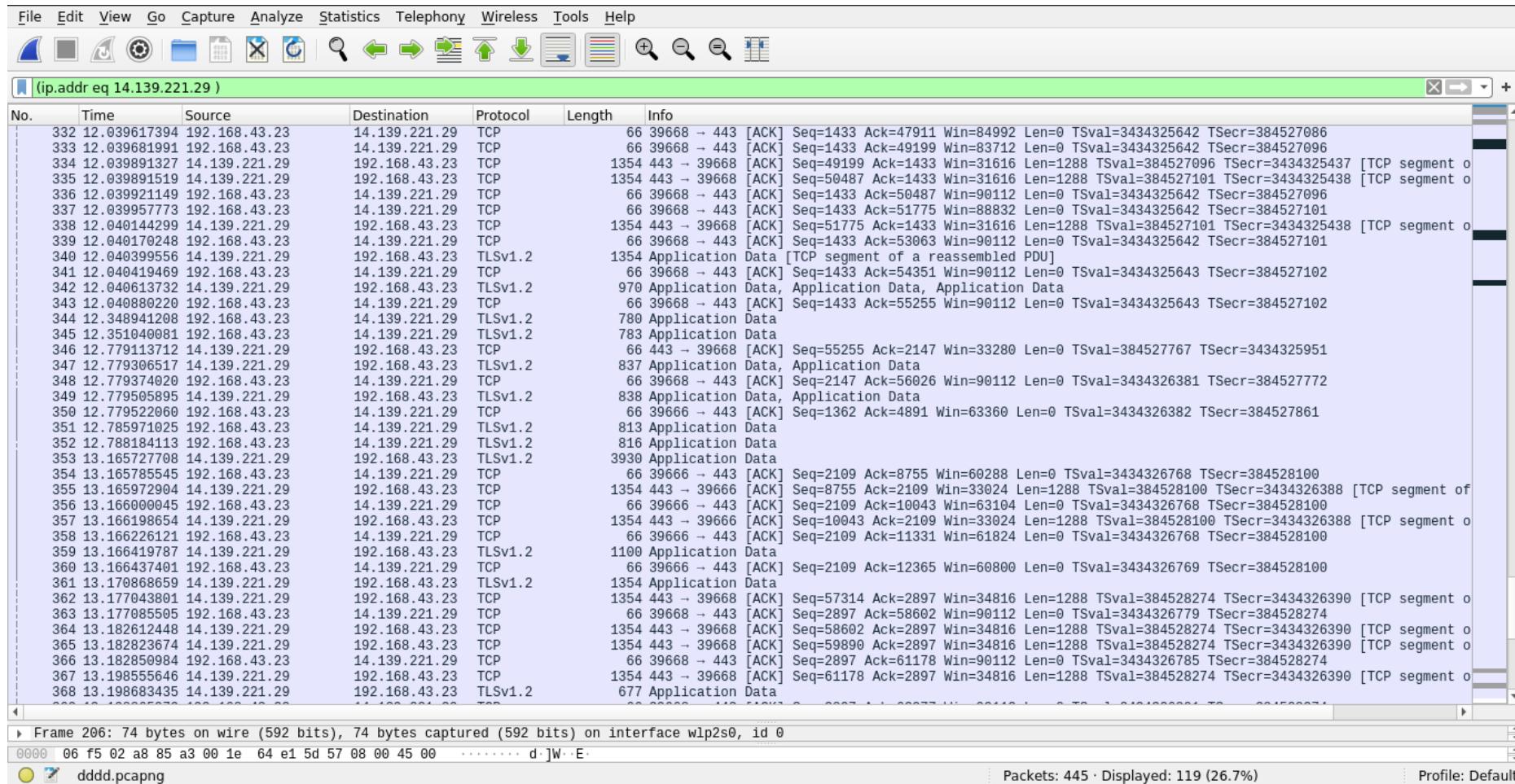


2. Using wireshark, capture the TCP headers while connecting your computer to the server of nit.dgp.ac.in.

Answer.



No.	Time	Source	Destination	Protocol	Length	Info
273	10.093562387	14.139.221.29	192.168.43.23	TCP	66	443 → 39668 [ACK] Seq=4119 Ack=1433 Win=31616 Len=0 TSval=384525046 TSecr=3434323359
294	10.913146457	14.139.221.29	192.168.43.23	TLSv1.2	6506	Application Data, Application Data
295	10.913264110	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=10559 Win=57728 Len=0 TSval=3434324515 TSecr=384525900
296	10.913512387	14.139.221.29	192.168.43.23	TCP	2642	443 → 39668 [ACK] Seq=10559 Ack=1433 Win=31616 Len=2576 TSval=384525900 TSecr=3434323359 [TCP segment o]
297	10.913555284	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=13135 Win=61056 Len=0 TSval=3434324516 TSecr=384525900
298	10.913756409	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=13135 Ack=1433 Win=31616 Len=1288 TSval=384525900 TSecr=3434323359 [TCP segment o]
299	10.913792589	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=14423 Win=59776 Len=0 TSval=3434324516 TSecr=384525900
300	10.913970733	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=14423 Ack=1433 Win=31616 Len=1288 TSval=384525900 TSecr=3434323359 [TCP segment o]
301	10.913998642	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=15711 Win=58496 Len=0 TSval=3434324516 TSecr=384525900
302	10.914293386	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=15711 Ack=1433 Win=31616 Len=1288 TSval=384525900 TSecr=3434323359 [TCP segment o]
303	10.914332305	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=16999 Win=57216 Len=0 TSval=3434324516 TSecr=384525900
306	11.118002125	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=16999 Ack=1433 Win=31616 Len=1288 TSval=384526176 TSecr=3434324515 [TCP segment o]
307	11.118078583	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=18287 Win=63104 Len=0 TSval=3434324720 TSecr=384526176
308	11.119355558	14.139.221.29	192.168.43.23	TLSv1.2	3930	Application Data [TCP segment of a reassembled PDU]
309	11.119355812	14.139.221.29	192.168.43.23	TCP	2642	443 → 39668 [ACK] Seq=22151 Ack=1433 Win=31616 Len=2576 TSval=384526182 TSecr=3434324516 [TCP segment o]
310	11.119355987	14.139.221.29	192.168.43.23	TCP	2642	443 → 39668 [ACK] Seq=24727 Ack=1433 Win=31616 Len=2576 TSval=384526201 TSecr=3434324516 [TCP segment o]
311	11.119410931	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=22151 Win=71936 Len=0 TSval=3434324722 TSecr=384526176
312	11.119468896	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=24727 Win=77056 Len=0 TSval=3434324722 TSecr=384526182
313	11.119495592	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=27303 Win=77824 Len=0 TSval=3434324722 TSecr=384526201
314	11.119548159	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=27303 Ack=1433 Win=31616 Len=1288 TSval=384526211 TSecr=3434324516 [TCP segment o]
315	11.119568436	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=28591 Win=80384 Len=0 TSval=3434324722 TSecr=384526211
316	11.119812688	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=28591 Ack=1433 Win=31616 Len=1288 TSval=384526211 TSecr=3434324516 [TCP segment o]
317	11.119842271	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=29879 Win=82944 Len=0 TSval=3434324722 TSecr=384526211
318	11.834833108	14.139.221.29	192.168.43.23	TCP	5218	443 → 39668 [ACK] Seq=29879 Ack=1433 Win=31616 Len=5152 TSval=384526806 TSecr=3434324722 [TCP segment o]
319	11.834894654	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=30501 Win=83968 Len=0 TSval=3434325437 TSecr=384526806
320	11.835002839	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=35031 Ack=1433 Win=31616 Len=1288 TSval=384526806 TSecr=3434324722 [TCP segment o]
321	11.835031076	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=36319 Win=82688 Len=0 TSval=3434325437 TSecr=384526806
322	11.835349990	14.139.221.29	192.168.43.23	TLSv1.2	2642	Application Data [TCP segment of a reassembled PDU]
323	11.835385425	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=38895 Win=88064 Len=0 TSval=3434325438 TSecr=384526806
324	11.835587511	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=38895 Ack=1433 Win=31616 Len=1288 TSval=384526806 TSecr=3434324722 [TCP segment o]
325	11.835620809	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=40183 Win=90112 Len=0 TSval=3434325438 TSecr=384526806
326	11.835849477	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=40183 Ack=1433 Win=31616 Len=1288 TSval=384526807 TSecr=3434324722 [TCP segment o]
327	11.835876610	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=4171 Win=90112 Len=0 TSval=3434325438 TSecr=384526807
328	11.836108495	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=4171 Ack=1433 Win=31616 Len=1288 TSval=384526807 TSecr=3434324722 [TCP segment o]
329	11.836135862	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=42759 Win=90112 Len=0 TSval=3434325438 TSecr=384526807
330	12.039553719	14.139.221.29	192.168.43.23	TCP	5218	443 → 39668 [ACK] Seq=42759 Ack=1433 Win=31616 Len=5152 TSval=384527086 TSecr=3434325437 [TCP segment o]
331	12.039554226	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=47911 Ack=1433 Win=31616 Len=1288 TSval=384527096 TSecr=3434325437 [TCP segment o]



No.	Time	Source	Destination	Protocol	Length	Info
340	12.040399556	14.139.221.29	192.168.43.23	TLSv1.2	1354	Application Data [TCP segment of a reassembled PDU]
341	12.040419469	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=54351 Win=90112 Len=0 TSval=3434325643 TSecr=384527102
342	12.040613732	14.139.221.29	192.168.43.23	TLSv1.2	970	Application Data, Application Data, Application Data
343	12.040880220	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=1433 Ack=55255 Win=90112 Len=0 TSval=3434325643 TSecr=384527102
344	12.348941208	192.168.43.23	14.139.221.29	TLSv1.2	780	Application Data
345	12.351040081	192.168.43.23	14.139.221.29	TLSv1.2	783	Application Data
346	12.779113712	14.139.221.29	192.168.43.23	TCP	66	443 → 39668 [ACK] Seq=55255 Ack=2147 Win=33280 Len=0 TSval=384527767 TSecr=3434325951
347	12.779306517	14.139.221.29	192.168.43.23	TLSv1.2	837	Application Data, Application Data
348	12.779374020	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=2147 Ack=56026 Win=90112 Len=0 TSval=3434326381 TSecr=384527772
349	12.779505895	14.139.221.29	192.168.43.23	TLSv1.2	838	Application Data, Application Data
350	12.779522060	192.168.43.23	14.139.221.29	TCP	66	39666 → 443 [ACK] Seq=1362 Ack=4891 Win=63360 Len=0 TSval=3434326382 TSecr=384527861
351	12.785971025	192.168.43.23	14.139.221.29	TLSv1.2	813	Application Data
352	12.788184113	192.168.43.23	14.139.221.29	TLSv1.2	816	Application Data
353	13.165727708	14.139.221.29	192.168.43.23	TLSv1.2	3930	Application Data
354	13.165785545	192.168.43.23	14.139.221.29	TCP	66	39666 → 443 [ACK] Seq=2109 Ack=8755 Win=60288 Len=0 TSval=3434326768 TSecr=384528100
355	13.165972904	14.139.221.29	192.168.43.23	TCP	1354	443 → 39666 [ACK] Seq=8755 Ack=2109 Win=33024 Len=1288 TSval=384528100 TSecr=3434326388 [TCP segment of a reassembled PDU]
356	13.166000045	192.168.43.23	14.139.221.29	TCP	66	39666 → 443 [ACK] Seq=2109 Ack=10043 Win=63104 Len=0 TSval=3434326768 TSecr=384528100
357	13.166198654	14.139.221.29	192.168.43.23	TCP	1354	443 → 39666 [ACK] Seq=10043 Ack=2109 Win=33024 Len=1288 TSval=384528100 TSecr=3434326388 [TCP segment of a reassembled PDU]
358	13.166226121	192.168.43.23	14.139.221.29	TCP	66	39666 → 443 [ACK] Seq=2109 Ack=11331 Win=61824 Len=0 TSval=3434326768 TSecr=384528100
359	13.166419787	14.139.221.29	192.168.43.23	TLSv1.2	1100	Application Data
360	13.166437401	192.168.43.23	14.139.221.29	TCP	66	39666 → 443 [ACK] Seq=2109 Ack=12365 Win=60800 Len=0 TSval=3434326769 TSecr=384528100
361	13.170868659	14.139.221.29	192.168.43.23	TLSv1.2	1354	Application Data
362	13.177043801	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=57314 Ack=2897 Win=34816 Len=1288 TSval=384528274 TSecr=3434326390 [TCP segment of a reassembled PDU]
363	13.177085505	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=2897 Ack=58602 Win=90112 Len=0 TSval=3434326779 TSecr=384528274
364	13.182612448	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=58602 Ack=2897 Win=34816 Len=1288 TSval=384528274 TSecr=3434326390 [TCP segment of a reassembled PDU]
365	13.182823674	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=59890 Ack=2897 Win=34816 Len=1288 TSval=384528274 TSecr=3434326390 [TCP segment of a reassembled PDU]
366	13.182850984	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=2897 Ack=61178 Win=90112 Len=0 TSval=3434326785 TSecr=384528274
367	13.198555646	14.139.221.29	192.168.43.23	TCP	1354	443 → 39668 [ACK] Seq=61178 Ack=2897 Win=34816 Len=1288 TSval=384528274 TSecr=3434326390 [TCP segment of a reassembled PDU]
368	13.198683435	14.139.221.29	192.168.43.23	TLSv1.2	677	Application Data
369	13.198865079	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=2897 Ack=63077 Win=90112 Len=0 TSval=3434326801 TSecr=384528274
372	18.183257942	14.139.221.29	192.168.43.23	TLSv1.2	97	Encrypted Alert
373	18.183258263	14.139.221.29	192.168.43.23	TCP	66	443 → 39666 [FIN, ACK] Seq=12396 Ack=2109 Win=33024 Len=0 TSval=384533105 TSecr=3434326769
374	18.183258363	14.139.221.29	192.168.43.23	TLSv1.2	97	Encrypted Alert
375	18.183315009	192.168.43.23	14.139.221.29	TCP	66	39666 → 443 [ACK] Seq=2109 Ack=12396 Win=64128 Len=0 TSval=3434331785 TSecr=384533105
376	18.183387770	14.139.221.29	192.168.43.23	TCP	66	443 → 39668 [FIN, ACK] Seq=63108 Ack=2897 Win=34816 Len=0 TSval=384533280 TSecr=3434326801
377	18.224406030	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=2897 Ack=63109 Win=90112 Len=0 TSval=3434331827 TSecr=384533280
378	18.224448660	192.168.43.23	14.139.221.29	TCP	66	39666 → 443 [ACK] Seq=2109 Ack=12397 Win=64128 Len=0 TSval=3434331827 TSecr=384533105

3. (a) Show how the six flags (SYN, ACK, PUSH, URGENT, RST, FIN) are working in TCP protocol

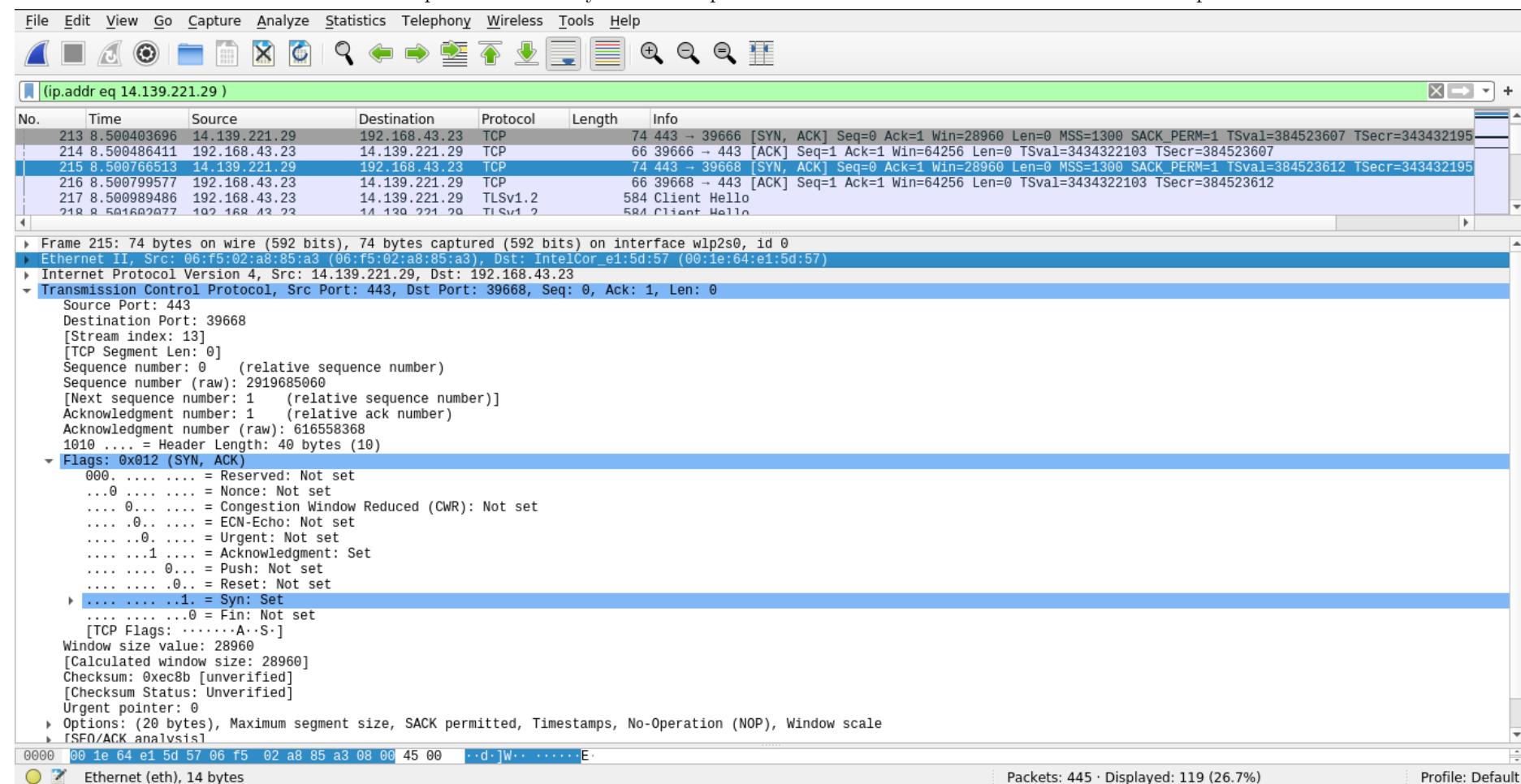
Answer.

SYN – For connection establishment The synchronization flag is used as a first step in establishing a three-way handshake between two hosts. Only the first packet from both the sender and receiver should have this flag set.

The screenshot shows a Wireshark capture window with the following details:

- File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help**
- File (ip.addr eq 14.139.221.29)**
- Table Headers:** No., Time, Source, Destination, Protocol, Length, Info
- Packets:**
 - 206 8.348910766 192.168.43.23 14.139.221.29 TCP 74 39666 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3434321951 TSeср=0 WS=128
 - 207 8.355002400 192.168.43.23 14.139.221.29 TCP 74 39668 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3434321957 TSeср=0 WS=128
 - 213 8.500403696 14.139.221.29 192.168.43.23 TCP 74 443 → 39666 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1300 SACK_PERM=1 TSval=384523607 TSeср=343432195
- Details pane:**
 - Frame 206: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface wlp2s0, id 0
 - Ethernet II, Src: IntelCor_e1:5d:57 (00:1e:64:e1:5d:57), Dst: 06:f5:02:a8:85:a3 (06:f5:02:a8:85:a3)
 - Internet Protocol Version 4, Src: 192.168.43.23, Dst: 14.139.221.29
 - Transmission Control Protocol, Src Port: 39666, Dst Port: 443, Seq: 0, Len: 0
 - Source Port: 39666
 - Destination Port: 443
 - [Stream index: 12]
 - [TCP Segment Len: 0]
 - Sequence number: 0 (relative sequence number)
 - Sequence number (raw): 3520860541
 - [Next sequence number: 1 (relative sequence number)]
 - Acknowledgment number: 0
 - Acknowledgment number (raw): 0
 - 1010 = Header Length: 40 bytes (10)
 - Flags: 0x002 (SYN)
 - 000 = Reserved: Not set
 - ...0 = Nonce: Not set
 - 0.... = Congestion Window Reduced (CWR): Not set
 -0.. = ECN-Echo: Not set
 -0.. = Urgent: Not set
 -0.. = Acknowledgment: Not set
 -0.. = Push: Not set
 -0.. = Reset: Not set
 -1.. = Syn: Set
 -0.. = Fin: Not set
 - [TCP Flags: ..S..]
 - Window size value: 64240
 - [Calculated window size: 64240]
 - Checksum: 0x90cc [unverified]
 - [Checksum Status: Unverified]
 - Urgent pointer: 0
 - Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
 - [Timestamps]
- Hex dump:** 0000 06 f5 02 a8 85 a3 00 1e 64 e1 5d 57 08 00 45 00 d J W E
- Status bar:** Packets: 445 · Displayed: 119 (26.7%) · Profile: Default

ACK – indicates that the acknowledgement number is valid. The acknowledgement flag is used to acknowledge the successful receipt of a packet. the receiver sends an ACK as well as a SYN in the second step of the three way handshake process to tell the sender that it received its initial packet.



FIN – for termination of connection (I have no more data to send) The finished flag means there is no more data from the sender. Therefore, it is used in the last packet sent from the sender.

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

(ip.addr eq 14.139.221.29)

No.	Time	Source	Destination	Protocol	Length	Info
374	18.183258363	14.139.221.29	192.168.43.23	TLSv1.2	97	Encrypted Alert
375	18.183315009	192.168.43.23	14.139.221.29	TCP	66	39666 → 443 [ACK] Seq=2109 Ack=12396 Win=64128 Len=0 TSval=3434331785 TSecr=384533105
376	18.183387770	14.139.221.29	192.168.43.23	TCP	66	443 → 39668 [FIN, ACK] Seq=63108 Ack=2897 Win=34816 Len=0 TSval=384533280 TSecr=3434326801
377	18.224406030	192.168.43.23	14.139.221.29	TCP	66	39668 → 443 [ACK] Seq=2897 Ack=63109 Win=90112 Len=0 TSval=3434331827 TSecr=384533280
378	18.224448660	192.168.43.23	14.139.221.29	TCP	66	39666 → 443 [ACK] Seq=2109 Ack=12397 Win=64128 Len=0 TSval=3434331827 TSecr=384533105

Frame 376: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface wlp2s0, id 0

Ethernet II, Src: IntelCor_e1:5d:57 (00:1e:64:e1:5d:57), Dst: IntelCor_e1:5d:57 (00:1e:64:e1:5d:57)

Internet Protocol Version 4, Src: 14.139.221.29, Dst: 192.168.43.23

Transmission Control Protocol, Src Port: 443, Dst Port: 39668, Seq: 63108, Ack: 2897, Len: 0

Source Port: 443
Destination Port: 39668
[Stream index: 13]
[TCP Segment Len: 0]
Sequence number: 63108 (relative sequence number)
Sequence number (raw): 2019748168
[Next sequence number: 63109 (relative sequence number)]
Acknowledgment number: 2897 (relative ack number)
Acknowledgment number (raw): 616561264
1000 = Header Length: 32 bytes (8)

Flags: 0x011 (FIN, ACK)

- 000. = Reserved: Not set
- ...0 = Nonce: Not set
-0.... = Congestion Window Reduced (CWR): Not set
-0.. = ECN-Echo: Not set
-0... = Urgent: Not set
-0. = Acknowledgment: Set
-0....0.. = Push: Not set
-0....0.. = Reset: Not set
-0....0.. = Syn: Not set
-0....0....1 = Fin: Set

[TCP Flags: A...F]

Window size value: 272
[Calculated window size: 34816]
[Window size scaling factor: 128]
Checksum: 0x5043 [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0

Onions: (12 bytes). No-Operation (NOP). No-Operation (NOP). Timestamps.....

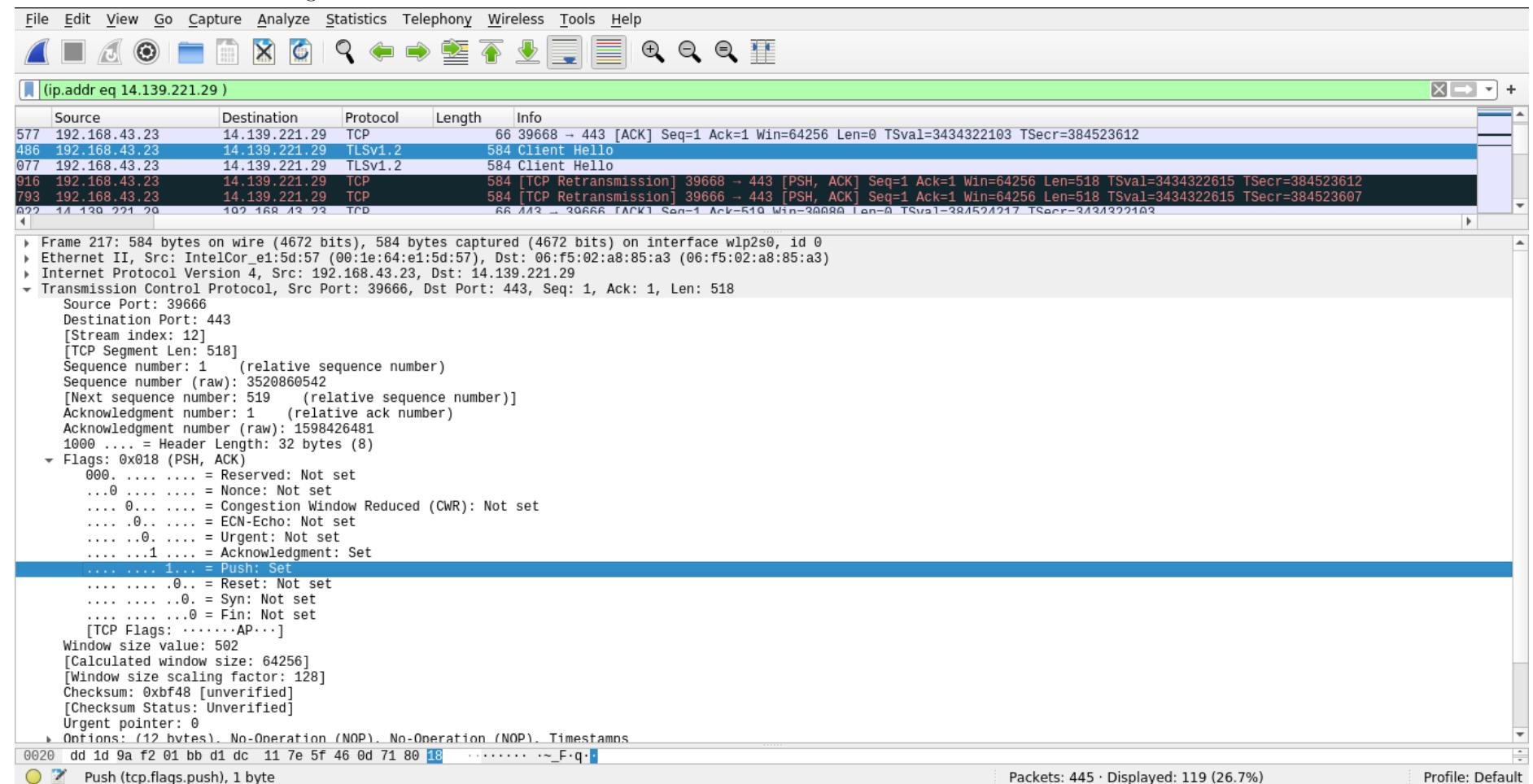
0000 00 1e 64 e1 5d 57 06 f5 02 a8 85 a3 08 00 45 00 ..d.]W.....E.

Ethernet (eth), 14 bytes

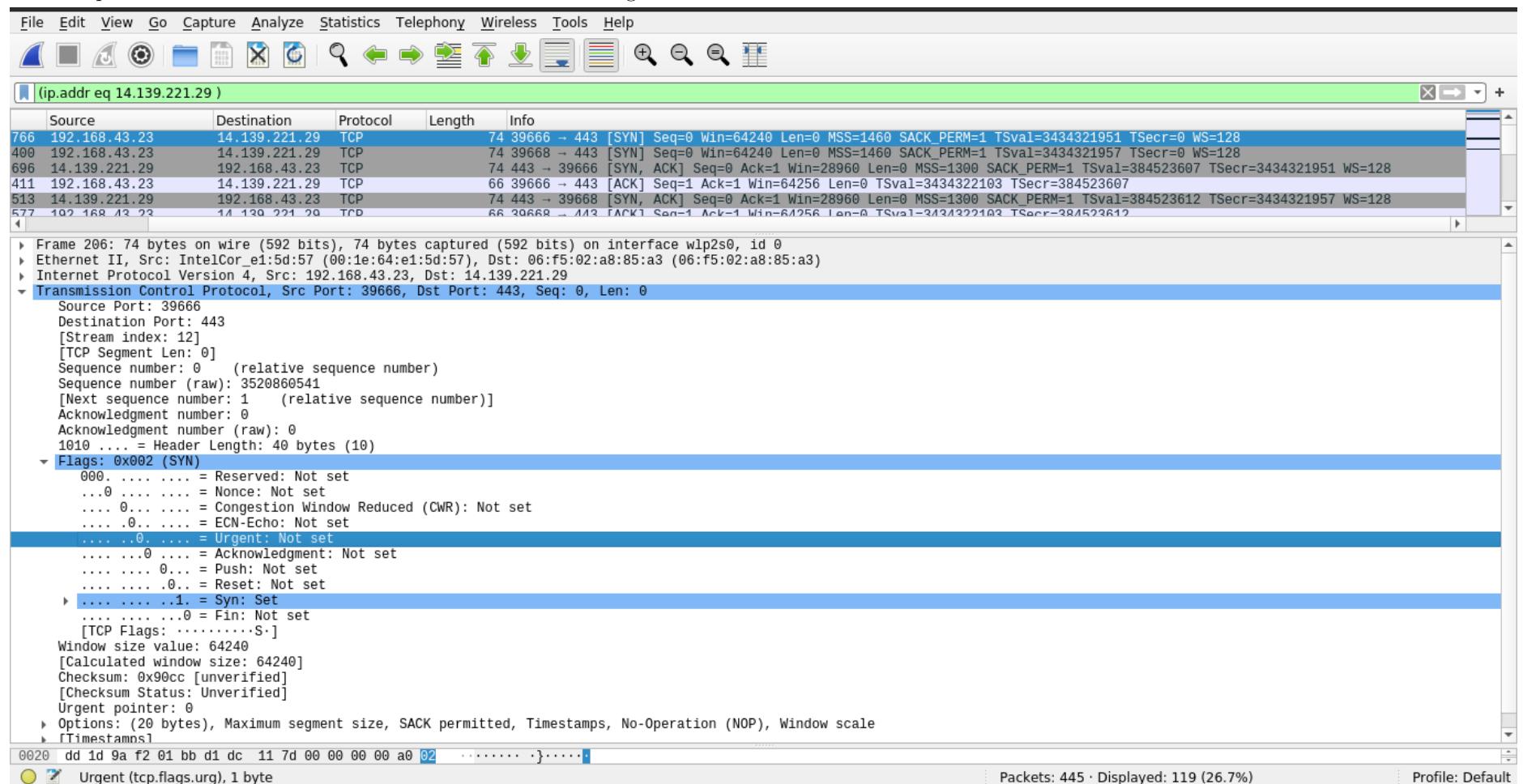
Packets: 445 · Displayed: 119 (26.7%)

Profile: Default

PSH – do not buffer – push the packet to the application The push flag is somewhat similar to the URG flag and tells the receiver to process these packets as they are received instead of buffering them.



URG – urgent byte in packet at location indicated in the “urgent” field The urgent flag is used to notify the receiver to process the urgent packets before processing all other packets. The receiver will be notified when all known urgent data has been received.



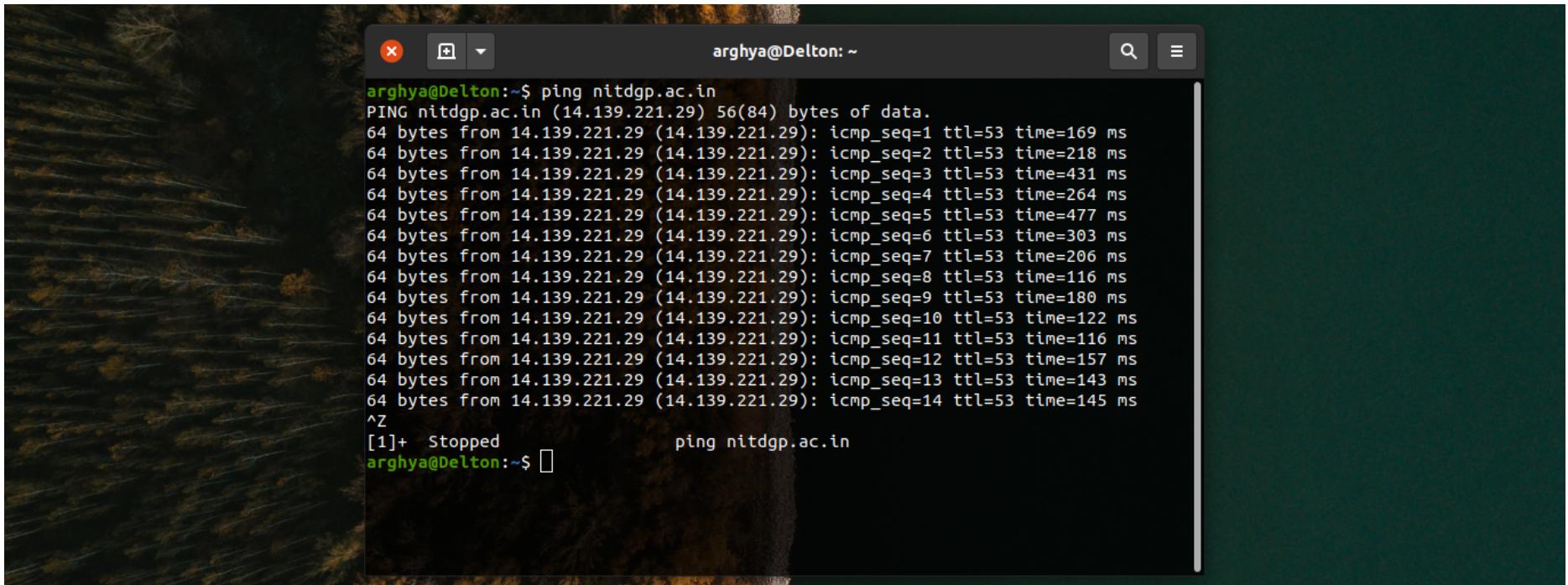
RST – rejection of TCP packet (denoted red in wireshark) The reset flag gets sent from the receiver to the sender when a packet is sent to a particular host that was not expecting it.

The screenshot shows a Wireshark capture window with the following details:

- File Bar:** File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help.
- Toolbar:** Includes icons for file operations, search, and zoom.
- Search Bar:** (ip.addr eq 14.139.221.29)
- Table:** Shows network traffic with columns: No., Time, Source, Destination, Protocol, Length, Info.
- Selected Packet:** The last packet (No. 3957) is highlighted in blue. Its details are expanded:
 - Ethernet II:** Src: IntelCor_e1:5d:57 (00:1e:64:e1:5d:57), Dst: 06:89:57:45:a7:ca (06:89:57:45:a7:ca)
 - Internet Protocol Version 4:** Src: 192.168.43.23, Dst: 14.139.221.29
 - TCP:** Src Port: 40468, Dst Port: 443, Seq: 2011, Ack: 768067, Len: 0
 - Flags:** 0x014 (RST, ACK)
 - 000. = Reserved: Not set
 -0 = Nonce: Not set
 -0.... = Congestion Window Reduced (CWR): Not set
 -0. = ECN-Echo: Not set
 -0. = Urgent: Not set
 -1 = Acknowledgment: Set
 -0.... = Push: Not set
 -1.. = Reset: Set
 -0. = Syn: Not set
 -0 = Fin: Not set
 - Window size value:** 3525
 - Checksum:** 0x8702 [unverified]
 - Checksum Status:** Unverified
 - Urgent pointer:** 0
 - Options:** (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
 - SEQ/ACK analysis:**
 - Timestamps:**
- Hex View:** Shows the raw hex and ASCII data of the selected packet.
- Status Bar:** wlp2s0: <live capture in progress>, Packets: 4342 · Displayed: 3457 (79.6%), Profile: Default.

- (b) What is the IP address of nitdgp.ac.in? On what port number is it sending and receiving TCP segments for this connection?

Answer. The IP address of nitdgp.ac.in is **14.139.221.29** and the tcp port number for sending and receiving TCP segments for the connections are **443** respectively.



The screenshot shows a terminal window titled "arghya@Delton:~". The user has run the command "ping nitdgp.ac.in". The output shows 15 ICMP echo requests being sent to the host at 14.139.221.29. Each request is 64 bytes long, has an ICMP sequence number (icmp_seq), and a TTL of 53. The time taken for each response varies from 116 ms to 477 ms. After the 15th request, the user presses ^Z to stop the ping command. The terminal then displays "[1]+ Stopped ping nitdgp.ac.in".

```
arghya@Delton:~$ ping nitdgp.ac.in
PING nitdgp.ac.in (14.139.221.29) 56(84) bytes of data.
64 bytes from 14.139.221.29 (14.139.221.29): icmp_seq=1 ttl=53 time=169 ms
64 bytes from 14.139.221.29 (14.139.221.29): icmp_seq=2 ttl=53 time=218 ms
64 bytes from 14.139.221.29 (14.139.221.29): icmp_seq=3 ttl=53 time=431 ms
64 bytes from 14.139.221.29 (14.139.221.29): icmp_seq=4 ttl=53 time=264 ms
64 bytes from 14.139.221.29 (14.139.221.29): icmp_seq=5 ttl=53 time=477 ms
64 bytes from 14.139.221.29 (14.139.221.29): icmp_seq=6 ttl=53 time=303 ms
64 bytes from 14.139.221.29 (14.139.221.29): icmp_seq=7 ttl=53 time=206 ms
64 bytes from 14.139.221.29 (14.139.221.29): icmp_seq=8 ttl=53 time=116 ms
64 bytes from 14.139.221.29 (14.139.221.29): icmp_seq=9 ttl=53 time=180 ms
64 bytes from 14.139.221.29 (14.139.221.29): icmp_seq=10 ttl=53 time=122 ms
64 bytes from 14.139.221.29 (14.139.221.29): icmp_seq=11 ttl=53 time=116 ms
64 bytes from 14.139.221.29 (14.139.221.29): icmp_seq=12 ttl=53 time=157 ms
64 bytes from 14.139.221.29 (14.139.221.29): icmp_seq=13 ttl=53 time=143 ms
64 bytes from 14.139.221.29 (14.139.221.29): icmp_seq=14 ttl=53 time=145 ms
^Z
[1]+  Stopped                  ping nitdgp.ac.in
arghya@Delton:~$
```

- (c) Write a small socket program for the URGENT pointer and urgent flag?

Answer.

The following code is the header file for **Binding server and connection of clients**.

```
/* bindacpt.h */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
```

```

#include <ctype.h>

int confd;
int mkaddr(void *addr,
           int *addrlen,
           char *str_addr,
           char *protocol) {

    char *inp_addr = strdup(str_addr);
    char *host_part = strtok(inp_addr, ":" );
    char *port_part = strtok(NULL, "\n" );
    struct sockaddr_in *ap =
        (struct sockaddr_in *) addr;
    struct hostent *hp = NULL;
    struct servent *sp = NULL;
    char *cp;
    long lv;

    /*
     * Set input defaults:
     */
    if ( !host_part ) {
        host_part = "*";
    }
    if ( !port_part ) {
        port_part = "*";
    }
    if ( !protocol ) {
        protocol = "tcp";
    }

    /*
     * Initialize the address structure:
     */
    memset(ap, 0, *addrlen);
    ap->sin_family = AF_INET;
    ap->sin_port = 0;
    ap->sin_addr.s_addr = INADDR_ANY;

    /*
     * Fill in the host address:
     */

```

```

if ( strcmp(host_part, "*" ) == 0 ) {
    /* Leave as INADDR_ANY */
}
else if ( isdigit(*host_part) ) {
    /*
     * Numeric IP address:
     */
    ap->sin_addr.s_addr =
        inet_addr(host_part);
    // if ( ap->sin_addr.s_addr == INADDR_NONE ) {
    if ( !inet_aton(host_part,&ap->sin_addr) ) {
        return -1;
    }
}
else {
    /*
     * Assume a hostname:
     */
    hp = gethostbyname(host_part);
    if ( !hp ) {
        return -1;
    }
    if ( hp->h_addrtype != AF_INET ) {
        return -1;
    }
    ap->sin_addr = * ( struct_in_addr *)
        hp->h_addr_list[0];
}

/*
 * Process an optional port #:
 */
if ( !strcmp(port_part, "*" ) ) {
    /* Leave as wild (zero) */
}
else if ( isdigit(*port_part) ) {
    /*
     * Process numeric port #:
     */
    lv = strtoul(port_part,&cp,10);
    if ( cp != NULL && *cp ) {
        return -2;
    }
}

```

```

        }
        if ( lv < 0L || lv >= 32768 ) {
            return -2;
        }
        ap->sin_port = htons( (short)lv );
    }
    else {
        /*
         * Lookup the service:
         */
        sp = getservbyname( port_part , protocol );
        if ( !sp ) {
            return -2;
        }
        ap->sin_port = (short) sp->s_port ;
    }

    /*
     * Return address length
     */
    *addrlen = sizeof *ap;

    free(inp_addr);
    return 0;
}

void sigurg(int signo)
{
    char buf[256];
    int n;
    printf("SIGURG received \n");
    n = recv(confd , buf, sizeof(buf)-1, MSG_OOB);
    buf[n] = 0;
    fprintf(stdout , "read %d OOB bytes :%s\n" , n, buf);
}

void bail(const char *on_what) {
    if ( errno != 0 ) {
        fputs(strerror(errno), stderr);
        fputs(": ", stderr);
    }
    fputs(on_what, stderr);
}

```

```

        fputc( '\n' , stderr );
        exit( 1 );
    }

int BindAccept( char *addr ) {
    int z;
    int s;
    struct sockaddr_in adr_srvr;
    struct sockaddr_in adr_clnt;
    int len_inet;

    /*
     * Create a TCP/IP socket to use:
     */
    s = socket( PF_INET, SOCK_STREAM, 0 );
    if ( s == -1 )
        bail( "socket()");

    /*
     * Bind the server address:
     */
    len_inet = sizeof adr_srvr;
    z = mkaddr( &adr_srvr, &len_inet,
                addr, "tcp" );

    if ( z != 0 ) {
        puts( "Bad address/port" );
        exit( 1 );
    }

    /*
     * Bind server address:
     */
    z = bind( s, ( struct sockaddr * ) &adr_srvr,
              len_inet );
    if ( z == -1 )
        bail( "bind(2)" );

    /*
     * Set listen mode:
     */
    if ( listen( s, 10 ) == -1 )

```

```

        bail(" listen (2)");

/*
 * Wait for a connect:
 */
len_inet = sizeof adr_clnt;

z = accept(s,
           (struct sockaddr *)&adr_clnt,
           &len_inet);

if ( z == -1 )
    bail("accept(2)");

close(s); /* No longer needed */
return z; /* Connected socket */
}

int Connect(char *addr) {
    int z;
    int s;
    struct sockaddr_in adr_srvr;
    int len_inet;
    /*
     * Create a TDP/IP socket to use:
     */
    s = socket(PF_INET,SOCK_STREAM,0);
    if ( s == -1 )
        bail("socket ()");

    /*
     * Bind the server address:
     */
    len_inet = sizeof adr_srvr;
    z = mkaddr(&adr_srvr,&len_inet,
               addr,"tcp");

    if ( z != 0 ) {
        puts("Bad address/port");
        exit(1);
    }
}

```

```

    * Connect to server:
    */
len_inet = sizeof adr_srvr;

z = connect(s,
    (struct sockaddr *)&adr_srvr,
    len_inet);

if ( z == -1 )
    bail("connect(2)");

return s; /* Connected socket */
}

```

The following code is the implementation for **TCP client side programming**.

```

/* tcpclient01.c */

#include "bindacpt.h"
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <cctype.h>

static void iband(int s,char *str) {
    int z;

    z = send(s,str,strlen(str),0);
    if ( z == -1 )
        bail("send(2)");

    printf("ib: '%s' (%d)\n",str,z);
}

```

```

/*
 * Send out-of-band data:
 */
static void
oband(int s, char *str) {
    int z;

    z = send(s, str, strlen(str), MSG_OOB);
    if (z == -1)
        bail("send(2)");

    printf("OOB '%s' (%d)\n", str, z);
}

int
main(int argc, char **argv) {
    int s = -1; /* Socket */

    s = Connect(argc >= 2
                ? argv[1]
                : "127.0.0.1:9011");

    iband(s, "In the beginning");
    sleep(1);
    iband(s, "Linus begat Linux,");
    sleep(1);

    iband(s, "and the Penguins");
    sleep(1);

    oband(s, "rejoiced");
    sleep(1);

    iband(s, "exceedingly.");
    close(s);

    return 0;
}

```

The following code is the implementation for **TCP Server side programming**.

```
/* tcpserver01.c */
```

```

#include "bindacpt.h"
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <ctype.h>

int listenfd;

int main( int argc , char **argv )
{
    int      n;
    int z;
    char buf[256];
    listenfd = BindAccept( argc >= 2? argv[1]: "127.0.0.1:9011");
    z = fcntl(listenfd ,F_SETOWN, getpid ());
    if ( z == -1 ) bail("fcntl(2)");
    signal(SIGURG, sigurg );
    for (;;) {
        z = recv(listenfd ,buf , sizeof buf ,0);
        if ( z == -1 )
            bail("recv(2)");
        if ( z == 0 )
            break;
        buf[z] = 0;
        printf("rcv '%s' (%d)\n",
               buf , z);
    }
    close(listenfd );
    return 0;
    exit(0);
}

```

```
arghya@Delton:~/My Work/Networks/Simple Socket Programming/Code for urgent bits$ ./tcpserver01
rcv 'In the beginning' (16)
rcv 'Linus begat Linux,' (18)
rcv 'and the Penguins' (16)
SIGURG received
read 16777215 00B bytes :oo)Ko
rcv 'rejoice' (7)
rcv 'exceedingly.' (12)
arghya@Delton:~/My Work/Networks/Simple Socket Programming/Code for urgent bits$
```



```
arghya@Delton:~/My Work/Networks/Simple Socket Programming/Code for urgent bits$ ./tcpclient01
ib: 'In the beginning' (16)
ib: 'Linus begat Linux,' (18)
ib: 'and the Penguins' (16)
00B 'rejoiced' (8)
ib: 'exceedingly.' (12)
arghya@Delton:~/My Work/Networks/Simple Socket Programming/Code for urgent bits$
```

Activities Wireshark ▾ Fri May 28 20:41:54 en: 77% ▾

Capturing from Loopback: lo

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
18	26.139445172	127.0.0.1	127.0.0.1	TCP	84	54170 → 9011 [PSH, ACK] Seq=17 Ack=1 Win=65536 Len=18 TSval=3372088451 TSecr=3372087451
19	26.139465116	127.0.0.1	127.0.0.1	TCP	66	9011 → 54170 [ACK] Seq=1 Ack=35 Win=65536 Len=0 TSval=3372088451 TSecr=3372088451
20	27.139761908	127.0.0.1	127.0.0.1	TCP	82	54170 → 9011 [PSH, ACK] Seq=35 Ack=1 Win=65536 Len=16 TSval=3372089452 TSecr=3372088451
21	27.139783198	127.0.0.1	127.0.0.1	TCP	66	9011 → 54170 [ACK] Seq=1 Ack=51 Win=65536 Len=0 TSval=3372089452 TSecr=3372089452
22	28.140023421	127.0.0.1	127.0.0.1	TCP	74	54170 → 9011 [PSH, ACK, URG] Seq=51 Ack=1 Win=65536 Urg=8 Len=8 TSval=3372090452 TSecr=3372089452
23	28.140064388	127.0.0.1	127.0.0.1	TCP	66	9011 → 54170 [ACK] Seq=1 Ack=59 Win=65536 Len=0 TSval=3372090452 TSecr=3372090452
24	29.140262569	127.0.0.1	127.0.0.1	TCP	78	54170 → 9011 [PSH, ACK] Seq=59 Ack=1 Win=65536 Len=12 TSval=3372091452 TSecr=3372090452

Frame 22: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface lo, id 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 54170, Dst Port: 9011, Seq: 51, Ack: 1, Len: 8

Source Port: 54170
Destination Port: 9011
[Stream index: 0]
[TCP Segment Len: 8]
Sequence number: 51 (relative sequence number)
Sequence number (raw): 1198778451
[Next sequence number: 59 (relative sequence number)]
Acknowledgment number: 1 (relative ack number)
Acknowledgment number (raw): 724443332
1000 = Header Length: 32 bytes (8)
Flags: 0x038 (PSH, ACK, URG)
000. = Reserved: Not set
....0 = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .0. = ECN-Echo: Not set
.... .1. = Urgent: Set
.... .1 = Acknowledgment: Set
.... 1... = Push: Set
.... 0.. = Reset: Not set
.... 0. = Syn: Not set
.... 0 = Fin: Not set
[TCP Flags:UAP....]
Window size value: 512
[Calculated window size: 65536]
[Window size scaling factor: 128]
Checksum: 0xfe30 [unverified]
[Checksum Status: Unverified]
Urgent pointer: 8

0020 00 01 d3 9a 23 33 47 73 e8 53 2b 2e 20 c4 80 38 ...#3Gs ·S+. ..8

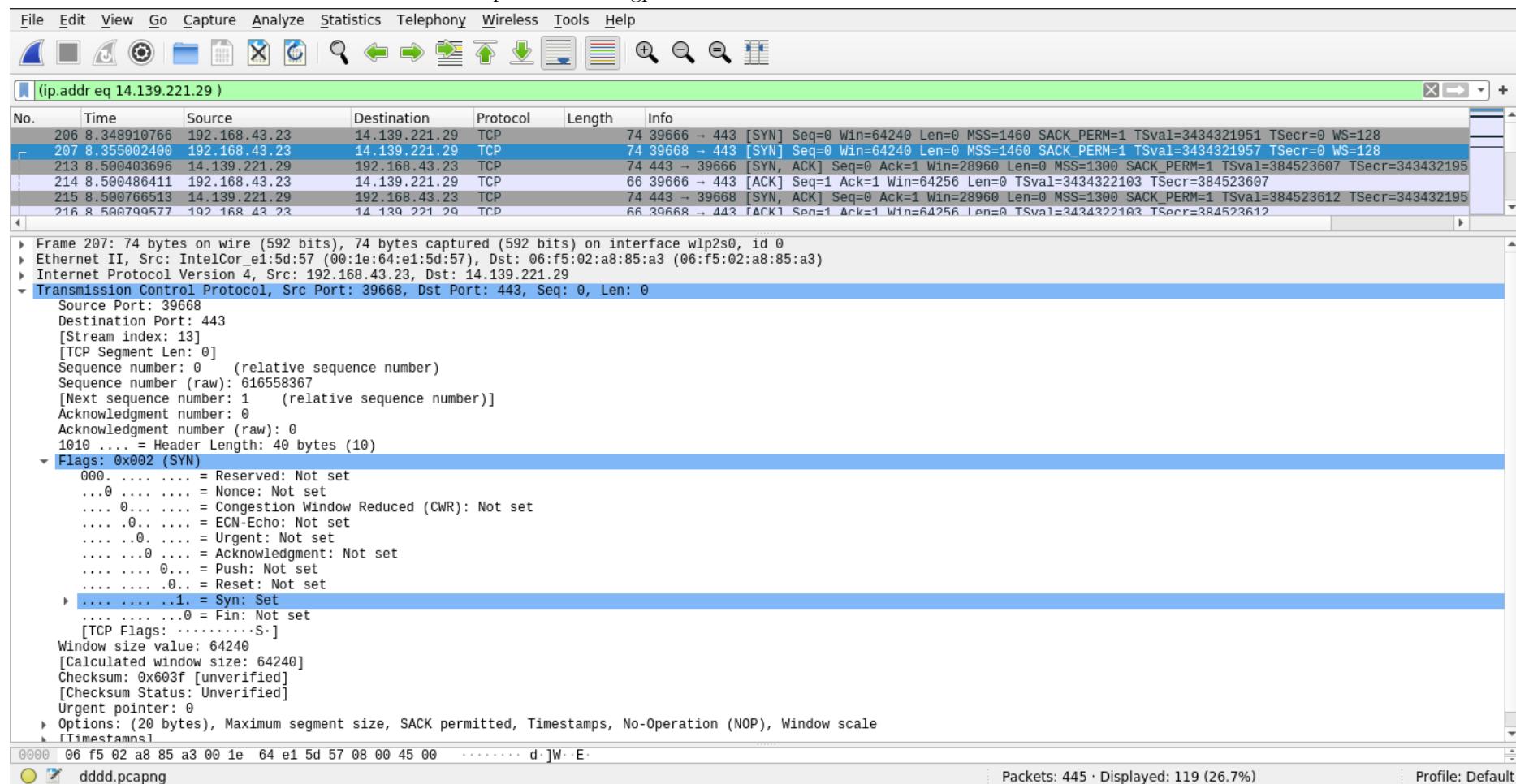
Urgent (tcp.flags.urg), 1 byte

Packets: 66 · Displayed: 66 (100.0%)

Profile: Default

- (d) What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and nitdgp.ac.in?

Answer. Relative sequence number is 0. And actual sequence number is **3520860541**. The sequence number of the TCP SYN segment is 0 since it is used to initiate the TCP connection between the client computer and nitdgp.ac.in.

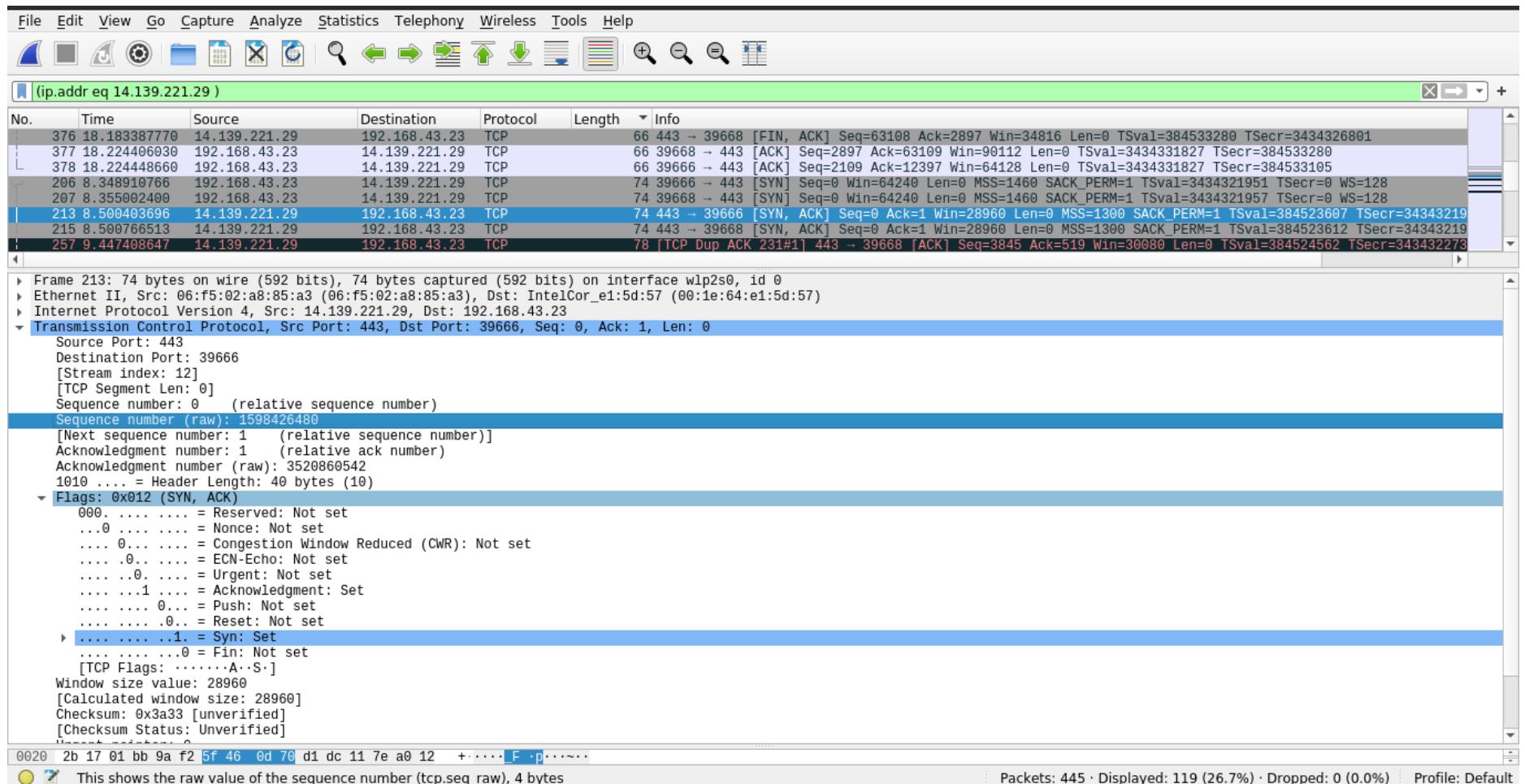


- (e) What is it in the segment that identifies the segment as a SYN segment?

Answer. In the above figure, It is indicated by the Syn flag in the Flags section, which is set to 1.

- (f) What is the sequence number of the SYN-ACK segment sent by nitdgp.ac.in to the client computer in reply to the SYN?

Answer. Relative sequence number is 0. And actual sequence number is **1598426480**. The sequence number of the TCP SYN segment is 0 since it is used to initiate the TCP connection between the client computer and nitdgp.ac.in.



(g) What is the value of the Acknowledgement field in the SYN-ACK segment?

Answer. According to the above figure, the value of the acknowledgement field in the SYNACK segment is 1.

(h) How did nitdgp.ac.in determine that value?

Answer. According to the above figure, the value of the ACK acknowledgement field in the SYNACK segment is determined by the server nitdgp.ac.in. The server adds 1 to the initial sequence number of SYN segment from the client computer. For this case, the initial sequence number of SYN segment from the client computer is 0, thus the value of the ACK acknowledgement field in the SYNACK segment is 1.

(i) What is it in the segment that identifies the segment as a SYN-ACK segment?

Answer. A segment will be identified as a SYNACK segment if both SYN flag and Acknowledgement in the segment are set to 1.