LAB NO: 3

Aim: Socket Programming in 'C' using UDP and Network Monitoring and Analysis with Wireshark

```
// udp client driver program
#include <stdio.h>         //Standard I/O functions like printf, puts, etc.
#include <strings.h>       //String functions like bzero()
#include <sys/types.h>     //Data types used in system calls.
#include <arpa/inet.h>     //Definitions for internet operations,like inet_addr() and htons().
#include <sys/socket.h>    //Definitions for socket operations, like socket(), connect(), etc.
#include<netinet/in.h>     //Contains constants and structures needed for internet domain addresses.
#include<unistd.h>         // Provides access to operating system API, including close() to close sockets.
#include<stdlib.h>         // Includes functions for memory allocation, process control, and conversions.

#define PORT 5000
#define MAXLINE 1000       //maximum size of the buffer

// Driver code
int main()
{
    char buffer[100];
    char *message = "Hello Server";
    int sockfd, n;
    struct sockaddr_in servaddr;

    // clear servaddr
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);
    servaddr.sin_family = AF_INET;

    // create datagram socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    // connect to server
    if(connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
    {
        printf("\n Error : Connect Failed \n");
        exit(0);
    }
      // request to send datagram to server
      // no need to specify server address in sendto
      // connect stores the peers IP and port
      sendto(sockfd, message, MAXLINE, 0, (struct sockaddr*)NULL, sizeof(servaddr));

      // waiting for response (receive data from the server)
      recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr*)NULL, NULL);
      puts(buffer);

      // close the socket, releasing the associated resources
      close(sockfd);
 }
```

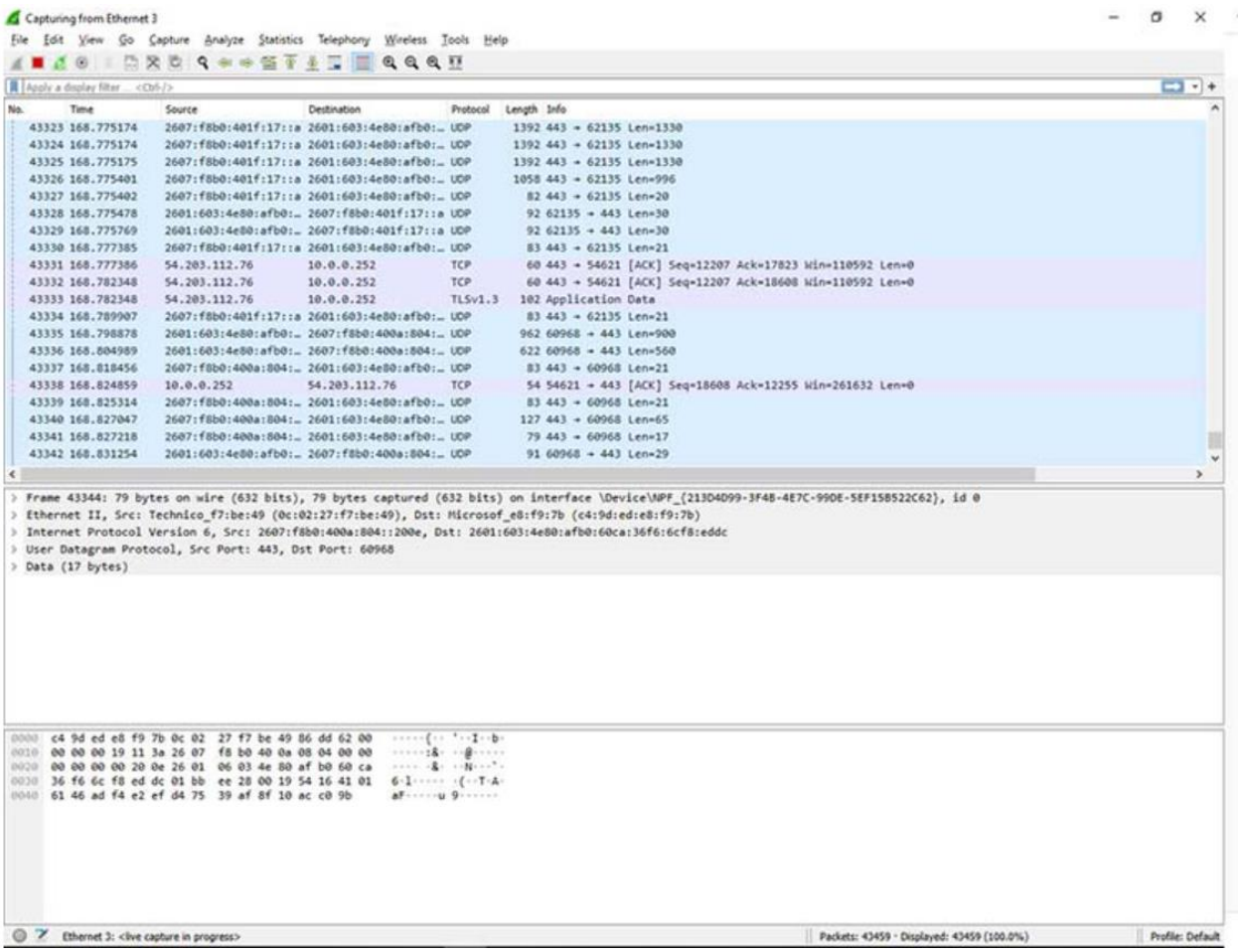
```
// server program for udp connection
#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include<netinet/in.h>

#define PORT 5000
#define MAXLINE 1000

// Driver code
int main()
{
    char buffer[100];
    char *message = "Hello Client";
    int listenfd, len;
    struct sockaddr_in servaddr, cliaddr;
    bzero(&servaddr, sizeof(servaddr));

    // Create a UDP Socket
    listenfd = socket(AF_INET, SOCK_DGRAM, 0);
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);
    servaddr.sin_family = AF_INET;

    // bind server address to socket descriptor
    bind(listenfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
    //receive the datagram
    len = sizeof(cliaddr);
    int n = recvfrom(listenfd, buffer, sizeof(buffer),
            0, (struct sockaddr*)&cliaddr,&len); //receive message from server
    buffer[n] = '\0';
    puts(buffer);

    // send the response
    sendto(listenfd, message, MAXLINE, 0,
          (struct sockaddr*)&cliaddr, sizeof(cliaddr));
}
```

Network Monitoring and Analysis with Wireshark

Wireshark is a network protocol analyzer, or an application that captures packets from a network connection, such as from your computer to your home office or the internet. Packet is the name given to a discrete unit of data in a typical Ethernet network.

Wireshark is the most often-used packet sniffer in the world. Like any other packet sniffer, Wireshark does three things:

1. **Packet Capture:** Wireshark listens to a network connection in real time and then grabs entire streams of traffic – quite possibly tens of thousands of packets at a time.
2. **Filtering:** Wireshark is capable of slicing and dicing all of this random live data using filters. By applying a filter, you can obtain just the information you need to see.
3. **Visualization:** Wireshark, like any good packet sniffer, allows you to dive right into the very middle of a network packet. It also allows you to visualize entire conversations and network streams.



Common Wireshark Use Cases

Here's a common example of how a Wireshark capture can assist in identifying a problem. The figure below shows an issue on a home network, where the internet connection was very slow. As the figure shows, the router thought a common destination was unreachable. This was discovered by drilling down into the IPv6 Internet Message Control Protocol (ICMP) traffic, which is marked in black. In Wireshark, any packet marked in black is considered to reflect some sort of issue.

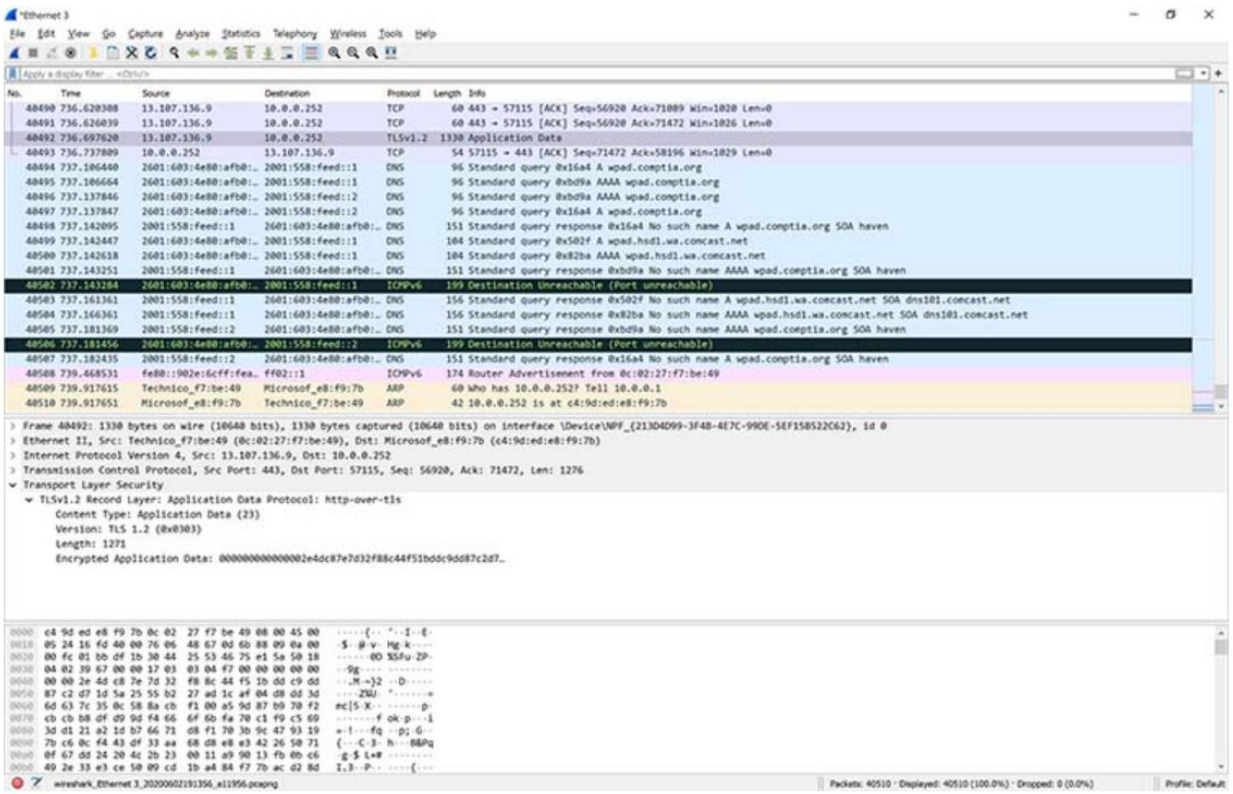


Figure 2: Drilling down into a packet to identify a network problem using Wireshark

In this case, Wireshark helped determine that the router wasn't working properly and couldn't find YouTube very easily. The problem was resolved by restarting the cable modem.

How to Install Wireshark on Linux

If you have a Linux system, you'd install Wireshark using the following sequence (notice that you'll need to have root permissions):

$ sudo apt-get install wireshark

$ sudo dpkg-reconfigure wireshark-common

$ sudo usermod -a -G wireshark $USER

$ newgrp wireshark

Once you have completed the above steps, you then log out and log back in, and then start Wireshark:

$ wireshark &

How to Capture Packets Using Wireshark

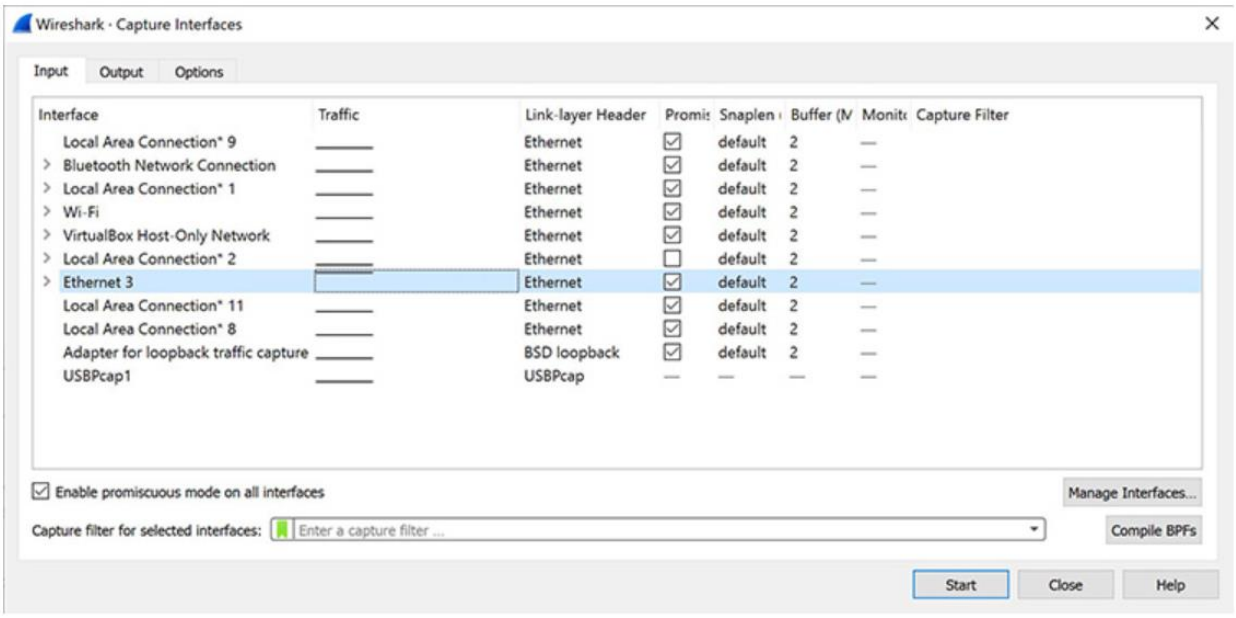


Figure 4: The Capture Interfaces dialog in Wireshark

This window will list all available interfaces. In this case, Wireshark provides several to choose from.

For this example, we'll select the Ethernet 3 interface, which is the most active interface. Wireshark visualizes the traffic by showing a moving line, which represents the packets on the network.

Once the network interface is selected, you simply click the Start button to begin your capture. As the capture begins, it's possible to view the packets that appear on the screen, as shown in Figure 5, below. Once you have captured all the packets that you want, simply click the red, square button at the top. Now you have a static packet capture to investigate.

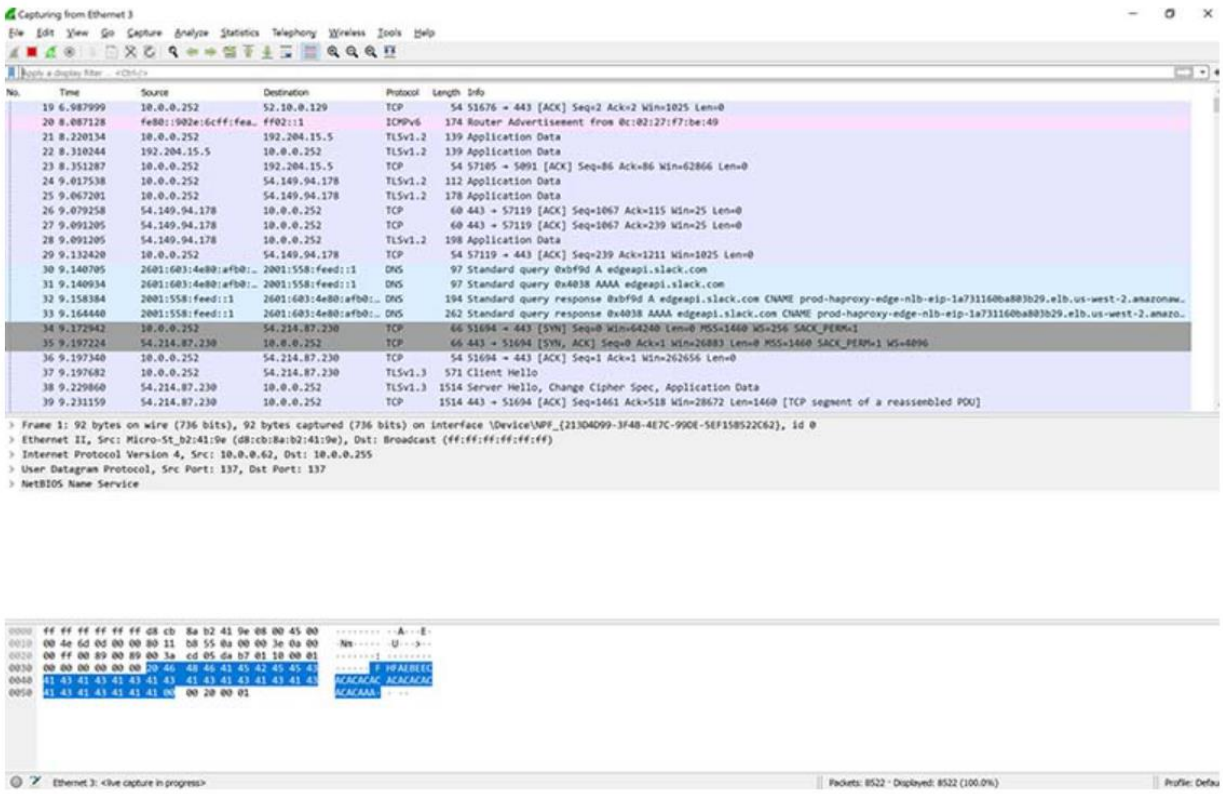


Figure 5: Wireshark capturing packets

What the Color Coding Means in Wireshark

Now that you have some packets, it's time to figure out what they mean. Wireshark tries to help you identify packet types by applying common-sense color coding. The table below describes the default colors given to major packet types.

Color in Wireshark	Packet Type
Light purple	TCP
Light blue	UDP
Black	Packets with errors
Light green	HTTP traffic
Light yellow	Windows-specific traffic, including Server Message Blocks (SMB) and NetBIOS
Dark yellow	Routing
Dark gray	TCP SYN, FIN and ACK traffic

The default coloring scheme is shown below in Figure 6. You can view this by going to View >> Coloring Rules.

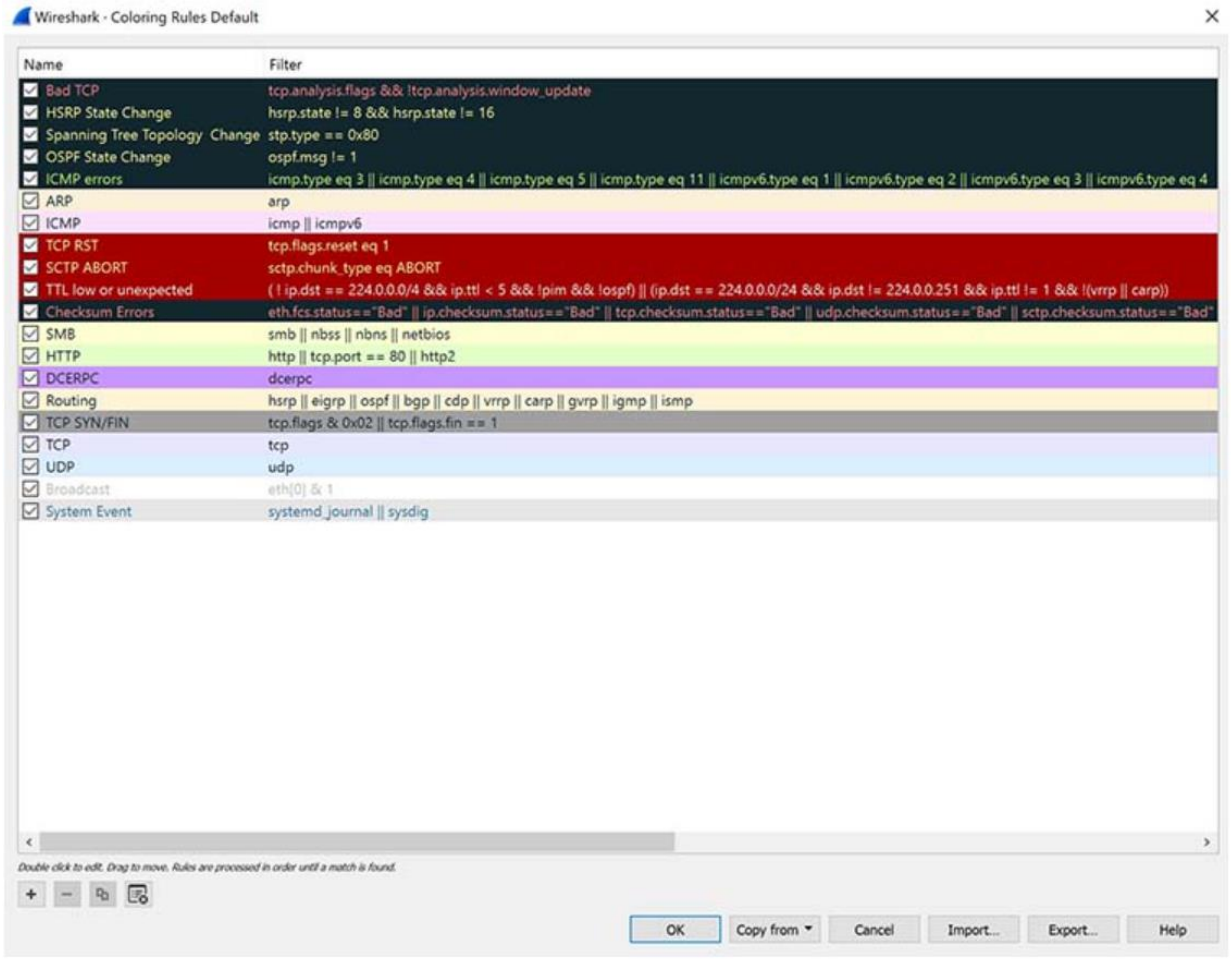


Figure 6: Default coloring rules

You can even change the defaults or apply a custom rule. If you don't want any coloring at all, go to View, then click Colorize Packet List. It's a toggle, so if you want the coloring back, simply go back and click Colorize Packet List again. It's possible, even, to colorize specific conversations between computers.

In Figure 7 below, you can see standard UDP (light blue), TCP (light purple), TCP handshake (dark gray) and routing traffic (yellow).

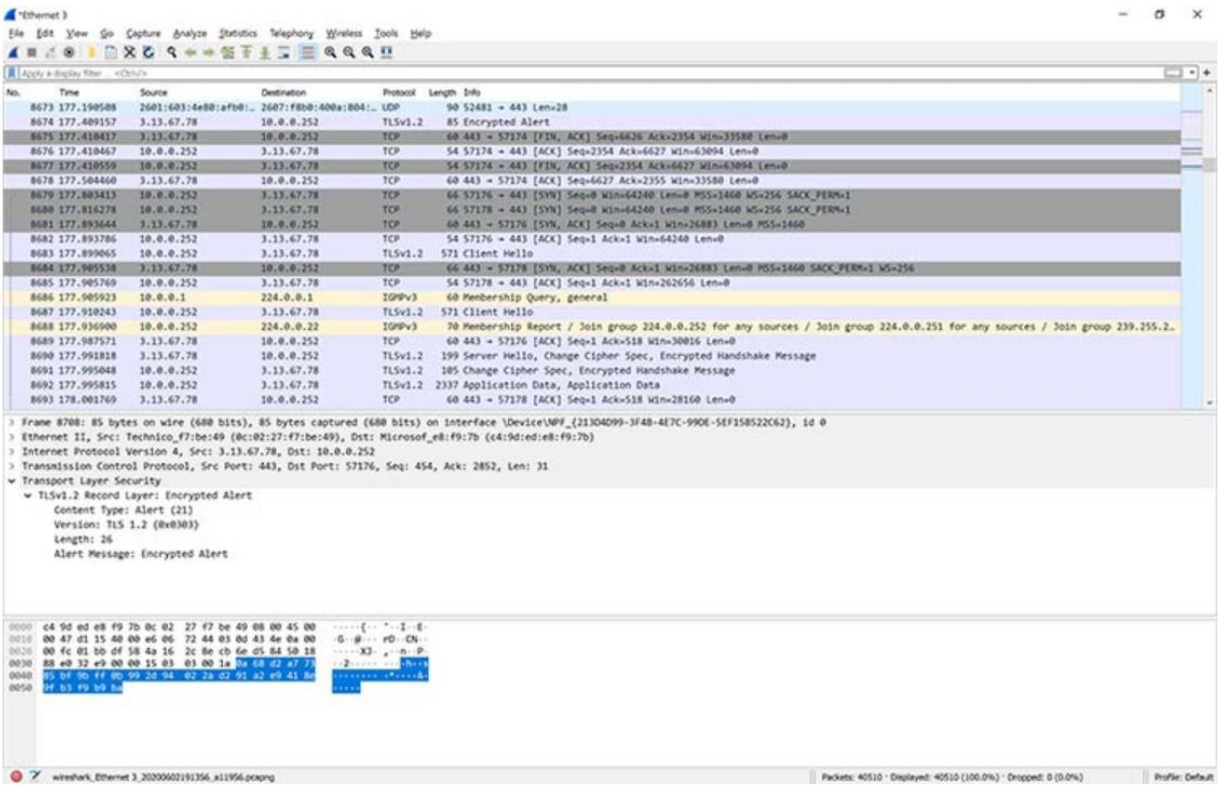


Figure 7: Viewing colorized packets in Wireshark