



Name: Marco Kamal Saad Naoum

ID: 17p6018

Group 1 Section 2

Table of Contents

1.Introduction :	3
2.UDP Client Code:	4
3. How Data transfer across layer:	7
4. Steps to capture your packets:	9
5. Packet Analyzing:	12
5.1 Sent Packet Analysis:	12
5.2 Received Packet Analysis:	15
6.Conclusion:	18

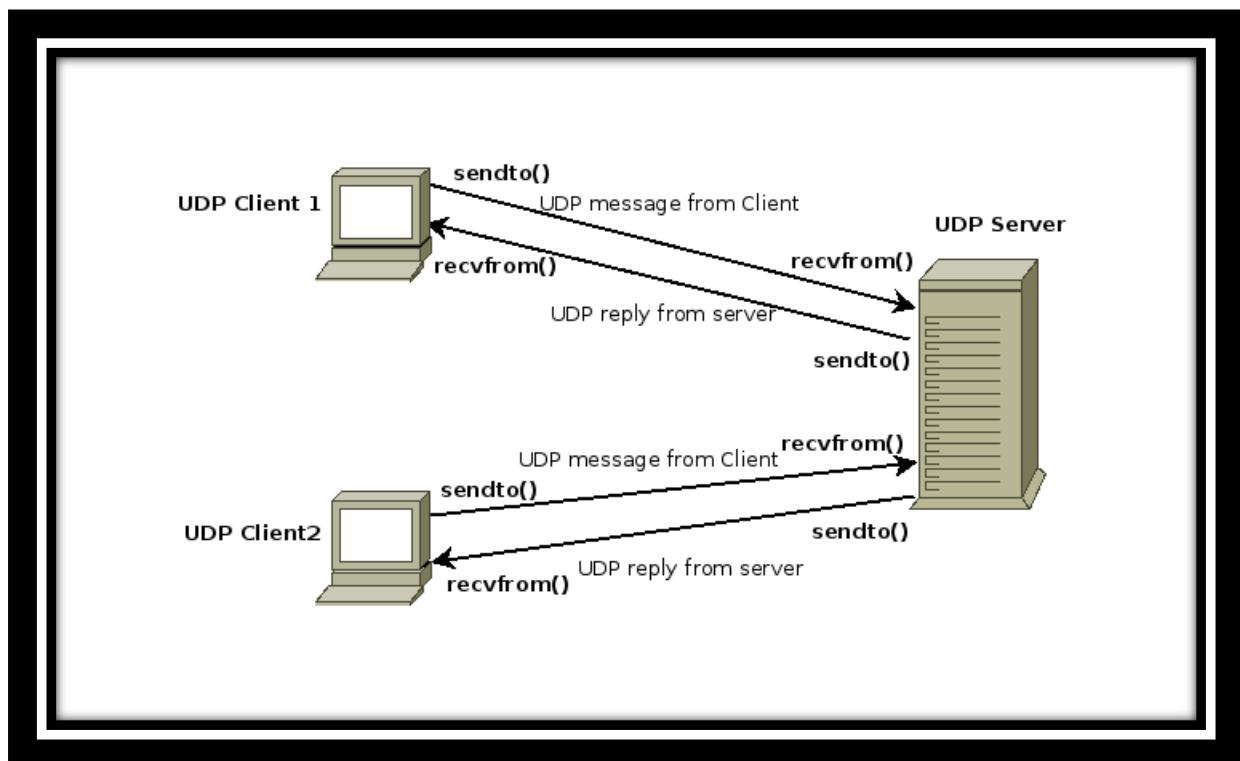
1.Introduction :

User Datagram Protocol (**UDP**) is transport layer protocol. It is also known for being it is unreliable and connectionless protocol So, there is no need to establish connection prior to data transfer, so **UDP** mostly used when:

1. Reduce the requirement of computer resources.
2. When using the Multicast or Broadcast to transfer.
3. mainly in multimedia applications.

UDP used in real-time computer application like voice or video communication and gaming

As it requires High performance, that is why **UDP** allow packet drop instead of processing delayed packet, so **UDP** is more efficient in term of latency and bandwidth.



2.UDP Client Code:

The purpose of the UDP client application is to send data to the server and receives echoed messages.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.Net.Sockets;
using System.IO;

namespace MyUDP_server
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            UdpClient myClient = new UdpClient();
            int count = 1;
            myClient.Client.SendTimeout = 1000;
            myClient.Client.ReceiveTimeout = 1000;
            DateTime t1, t2;
            while (true && count <=10)
            {
                String msg = "ping";
                Console.WriteLine("Client msg: {0} ", msg);

                Byte[] msgdata = Encoding.ASCII.GetBytes(msg);
                IPEndPoint ipserver = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 7777);
                myClient.Send(msgdata, msgdata.Length, ipserver);
                t1 = DateTime.Now;
                try
                {
                    Byte[] recieveData = myClient.Receive(ref ipserver);
                    t2 = DateTime.Now;
                    string servermsg = Encoding.ASCII.GetString(recieveData);
                    Console.WriteLine("From server: {0} {1}", servermsg, count);
                    Console.WriteLine("RTT : {0}", (t2 - t1).TotalMilliseconds);
                    Console.WriteLine("-----");
                }
                catch (Exception e)
                {
                    t2 = DateTime.Now;
                    Console.WriteLine("Request Timeout");
                    Console.WriteLine("-----");
                }
                count++;
            }
        }
    }
}
```

This Code was written in C# language

In this Block we created:

```
0 references
static void Main(string[] args)
{
    UdpClient myClient = new UdpClient();
    int count = 1;
    myClient.Client.SendTimeout = 1000;
    myClient.Client.ReceiveTimeout = 1000;
    DateTime t1, t2;
```

- An instance of UDP Client class
- Declared variable count (**int**) so we can use to send specific number of messages
- Set our Connection time for specific time = 1 sec
- Created variable (**DateTime**) t1,t2

The next block contains how Client work:

```
while (true && count <=10)
{
    String msg = "ping";
    Console.WriteLine("Client msg: {0} ", msg);

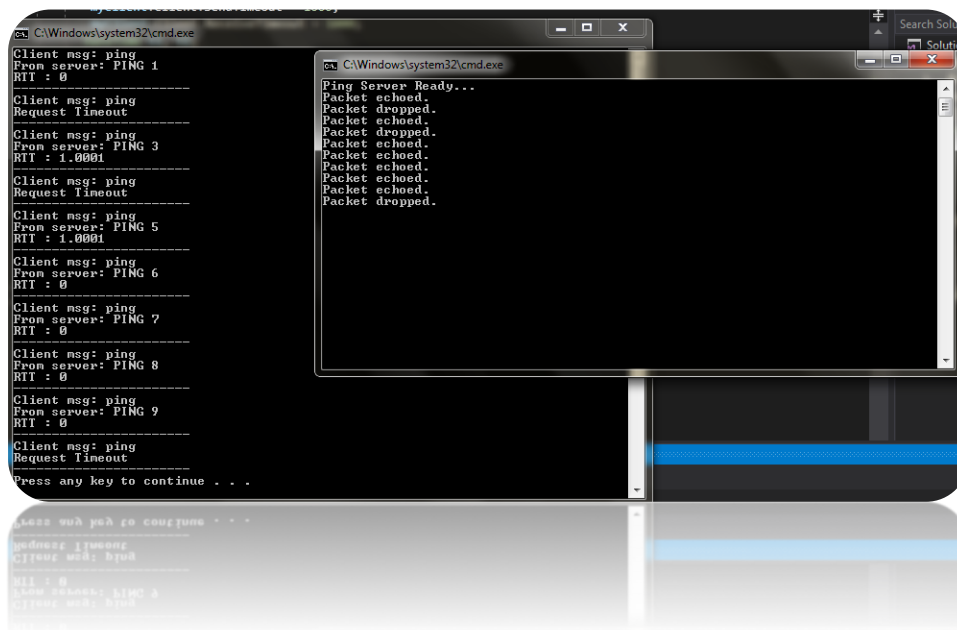
    Byte[] msgdata = Encoding.ASCII.GetBytes(msg);
    IPEndPoint ipserver = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 7777);
    myClient.Send(msgdata, msgdata.Length, ipserver);
    t1 = DateTime.Now;
    try
    {
        Byte[] recieveData = myClient.Receive(ref ipserver);
        t2 = DateTime.Now;
        string servermsg = Encoding.ASCII.GetString(recieveData);
        Console.WriteLine("From server: {0} {1}", servermsg, count);
        Console.WriteLine("RTT : {0}", (t2 - t1).TotalMilliseconds);
        Console.WriteLine("-----");
    }
    catch (Exception e)
    {
        t2 = DateTime.Now;
        Console.WriteLine("Request Timeout");
        Console.WriteLine("-----");
    }
    count++;
}
```

The Client starts with sending a String msg **ping** in form of stream of bytes to the server whose IP = **"127.0.0.1"** and Port = **7777** then we save the time we send msg in **t1**, inside the **Try and Catch** block the Client will wait for respond for specific time. If a **servermsg** was received, the Client will Decode

the received message and save time when it got received t_2 and print the message on the screen with the RTT.

If it failed to receive any message it will print out "Request Timeout".

The Output screen:



```
Client msg: ping
From server: PING 1
RTT : 0

Client msg: ping
Request Timeout

Client msg: ping
From server: PING 3
RTT : 1.0001

Client msg: ping
Request Timeout

Client msg: ping
From server: PING 5
RTT : 1.0001

Client msg: ping
From server: PING 6
RTT : 0

Client msg: ping
From server: PING 7
RTT : 0

Client msg: ping
From server: PING 8
RTT : 0

Client msg: ping
From server: PING 9
RTT : 0

Client msg: ping
Request Timeout

Press any key to continue . . .
```

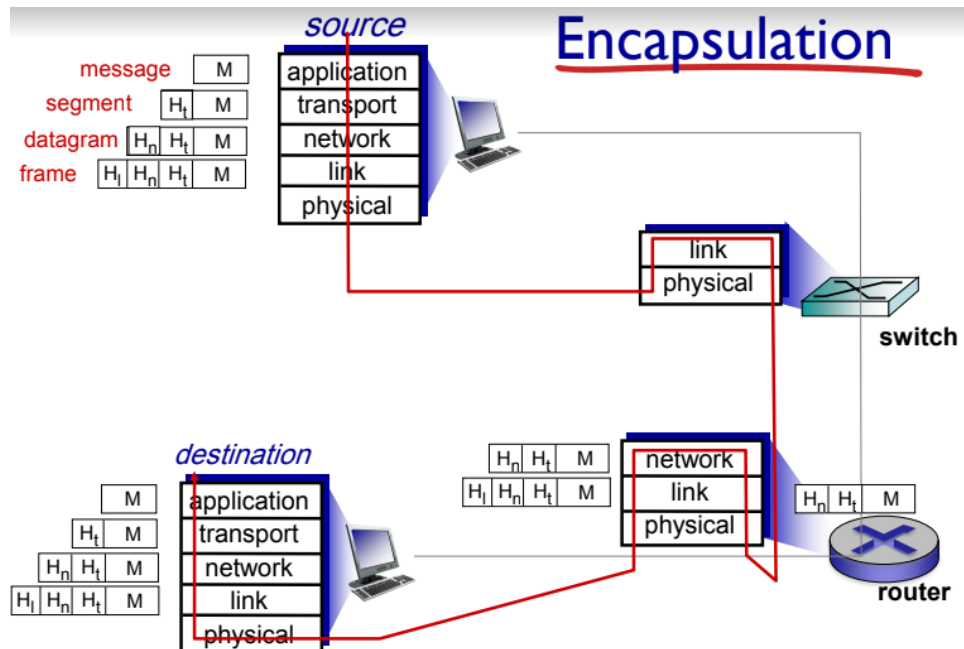
```
Ping Server Ready...
Packet echoed.
Packet dropped.
Packet echoed.
Packet dropped.
Packet echoed.
Packet dropped.
Packet echoed.
Packet dropped.
Packet echoed.
Packet dropped.
```

10 messages were sent from the Client, but server respond for 7 only and 3 were dropped.

3. How Data transfer across layer:

First of all, we will discuss how Data transfer through layers of Network

The message pass through 5 layers:



- **Application Layer:** known as **Desktop layer**, layer which implemented by network applications which produce Data to be transferred over the network, also to provide some useful function:
 1. Ensures that the receiving device is identified, can be reached and is ready to accept data.
 2. Presents the data on the receiving end to the user application
 3. Makes sure necessary communication interfaces exist.

Some of the commonly used protocols in this layer:

- TELNET
 - FTP
 - NFS
- **Transport Layer:** it provides services to Application layer and take services from Network layer. Data is in form of **Segments** and responsible to acknowledge of successful data transmission and re-sent Error:

1. At sender's side: Transport Layer receive Data from upper Layer and start Segmentation process adding Destination Port number in the header and forward it to Network Layer.
2. At receiver's side: Transport Layer read Port number from the Data header and perform sequencing and reassembling of the segmented data and forward it to Application layer

So, it provides some useful function:

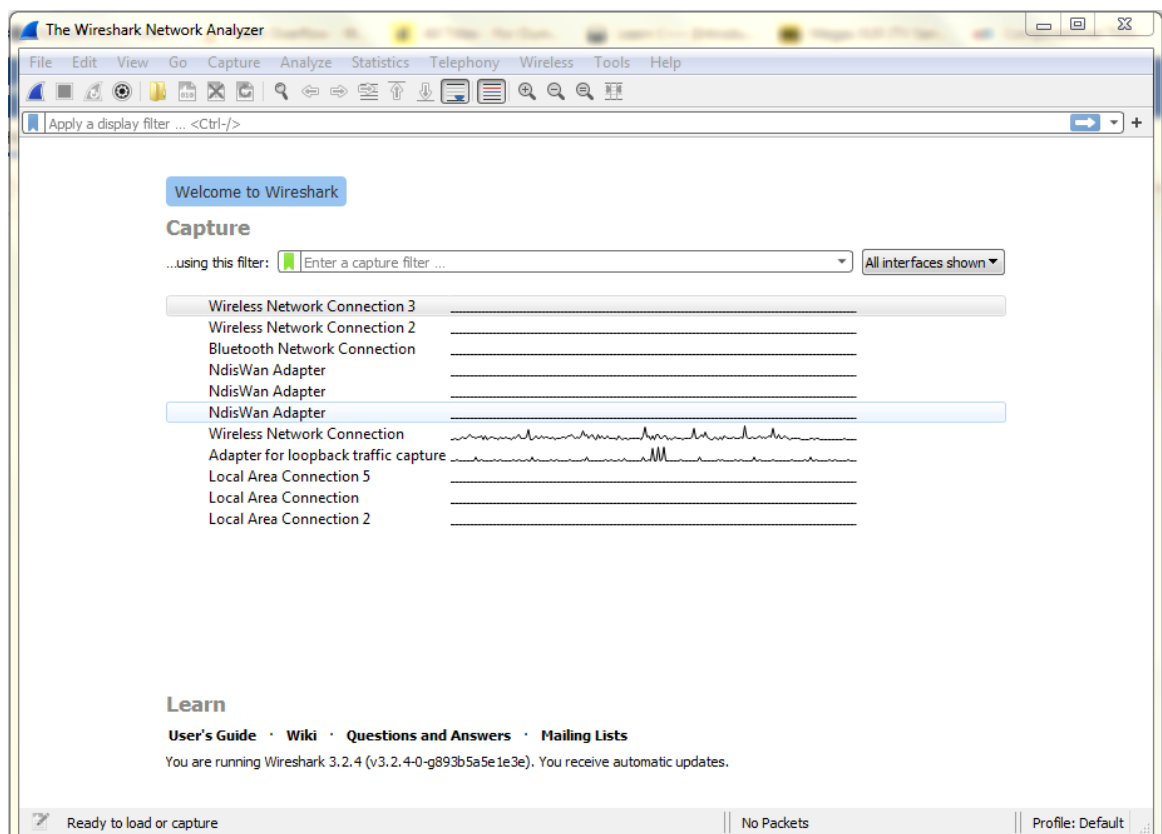
- i. Maintain Flow Control and Error Control in order to ensure that each message should reach its destination completely.
 - ii. uses some standard protocols to enhance its functionalities (TCP/UDP).
 - iii. establishes a connection between two end ports. A connection is a single logical path from source to destination which is associated with all the packets in a message
- **Network Layer:** works for transmission **Packets** from host to another and deciding the shortest available path and adding sender & receiver IP's address to the header, so it provides the following function:
 - i. It translates logical network address into physical address.
 - ii. Breaks larger packets into small packets.
 - iii. Connection services are provided including network layer flow control, network layer error control and packet sequence control.
 - **Data Link Layer (DLL):** Responsible for the node to node delivery of the message, the main function of this Layer is making sure data is transferred error-free from node to another over physical layer, its responsibility to transmit the packet to the Host using Mac address, **Frame** process takes place in this layer where segment breaks into set of bits meaningful to the receiver, It makes sure the appropriate physical protocol is assigned to the data. The main function of this layer:
 - i. deal with transmission errors.
 - ii. regulate the flow of data.

- iii. provide a well-defined interface to the network layer
- Physical Layer: known as [Hardware Layer](#) responsible for actual physical connection between devices, in this layer message is in form of bits transmitted individually, when received converting data into 0s and 1s and sent it to [DLL](#) which frame them back.

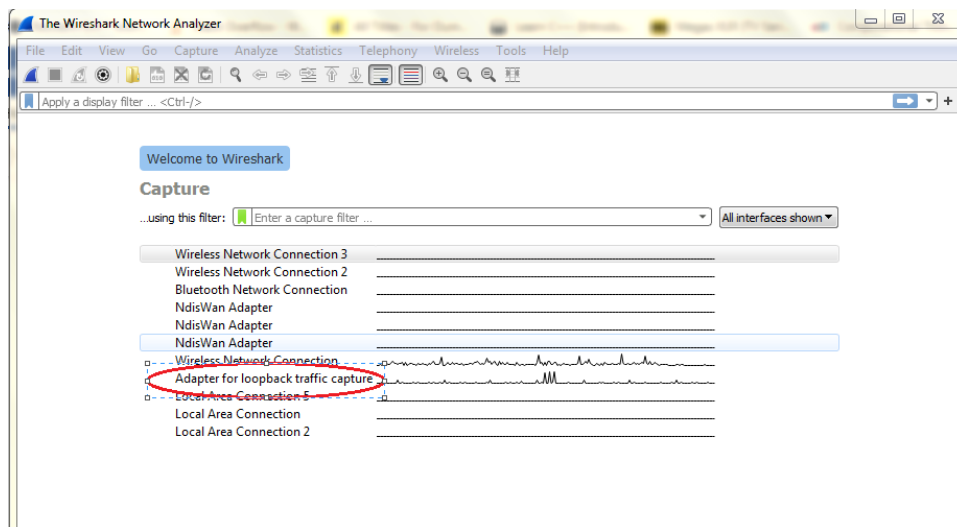
Since now we understand how our Layers work, we will do some application on it using UDP by using our Server and Wireshark.

4. Steps to capture your packets:

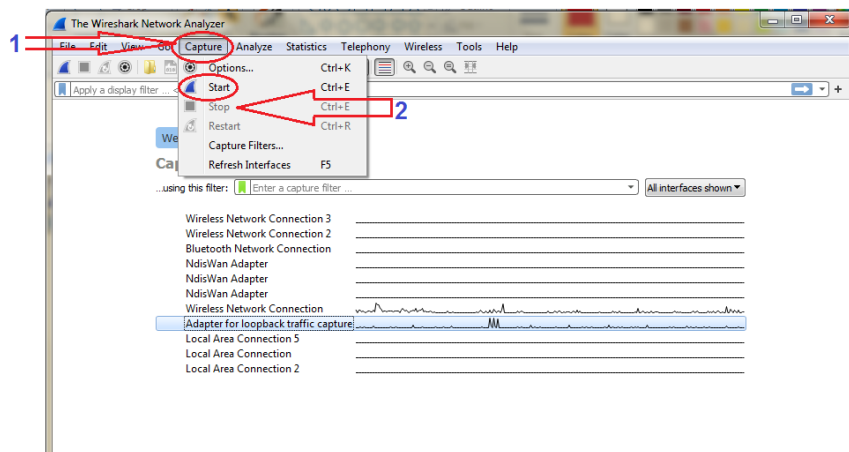
1. Open Wireshark



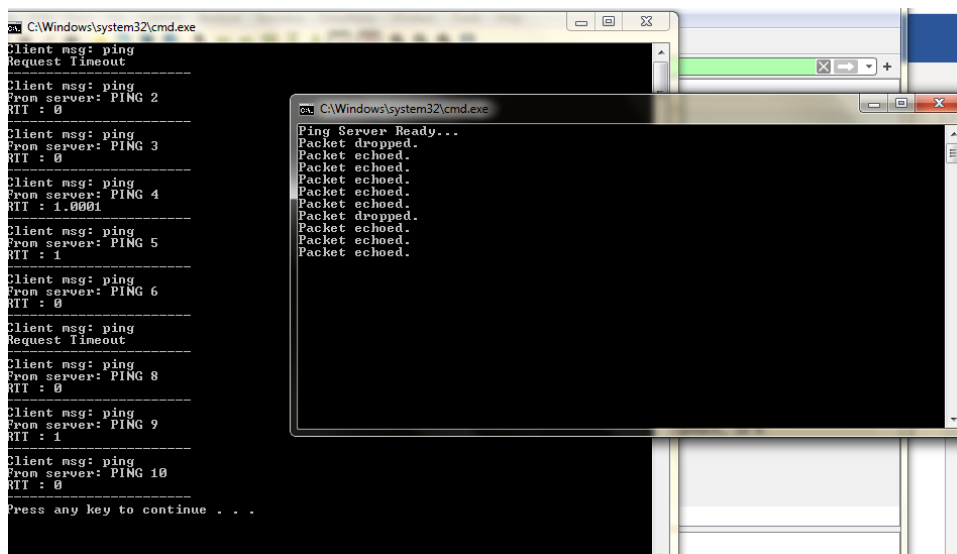
2. Choose “Adapter For loopback traffic Capture”



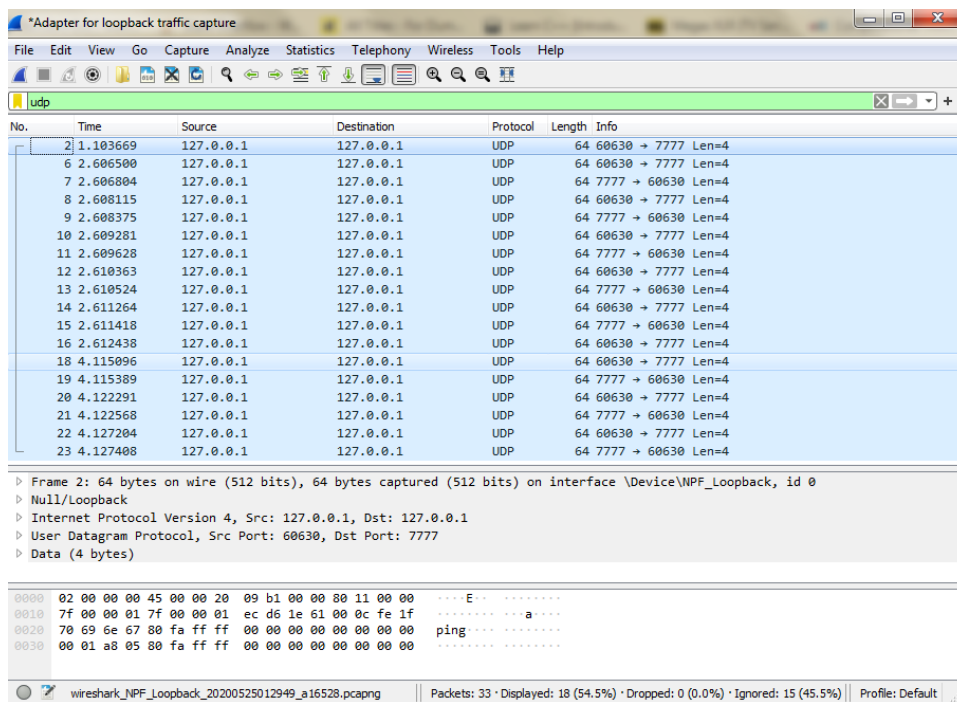
3. Click on “Capture” from toolbar at top and click start



4. Startup your Server and Client .exe file



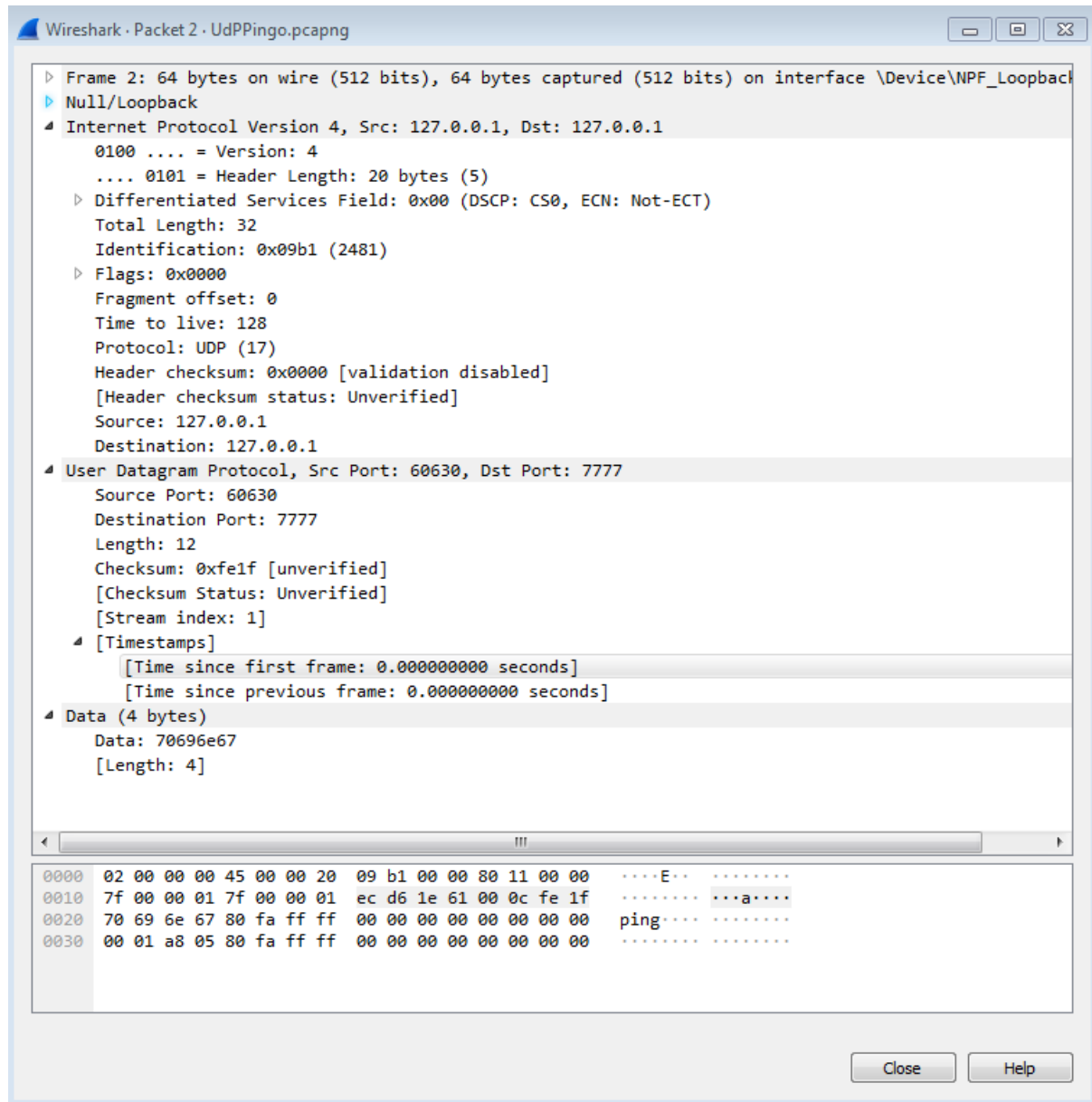
5. Now go back to Wireshark and Filter the unwanted traffic.



Also, we notice that the packets number is doubled as there is packet goes from client to server and server to client (incase the message didn't drop), in this example we have 18 captured packet and 8 successful messages that echoed and 2 dropped packets.

5. Packet Analyzing:

5.1 Sent Packet Analysis:



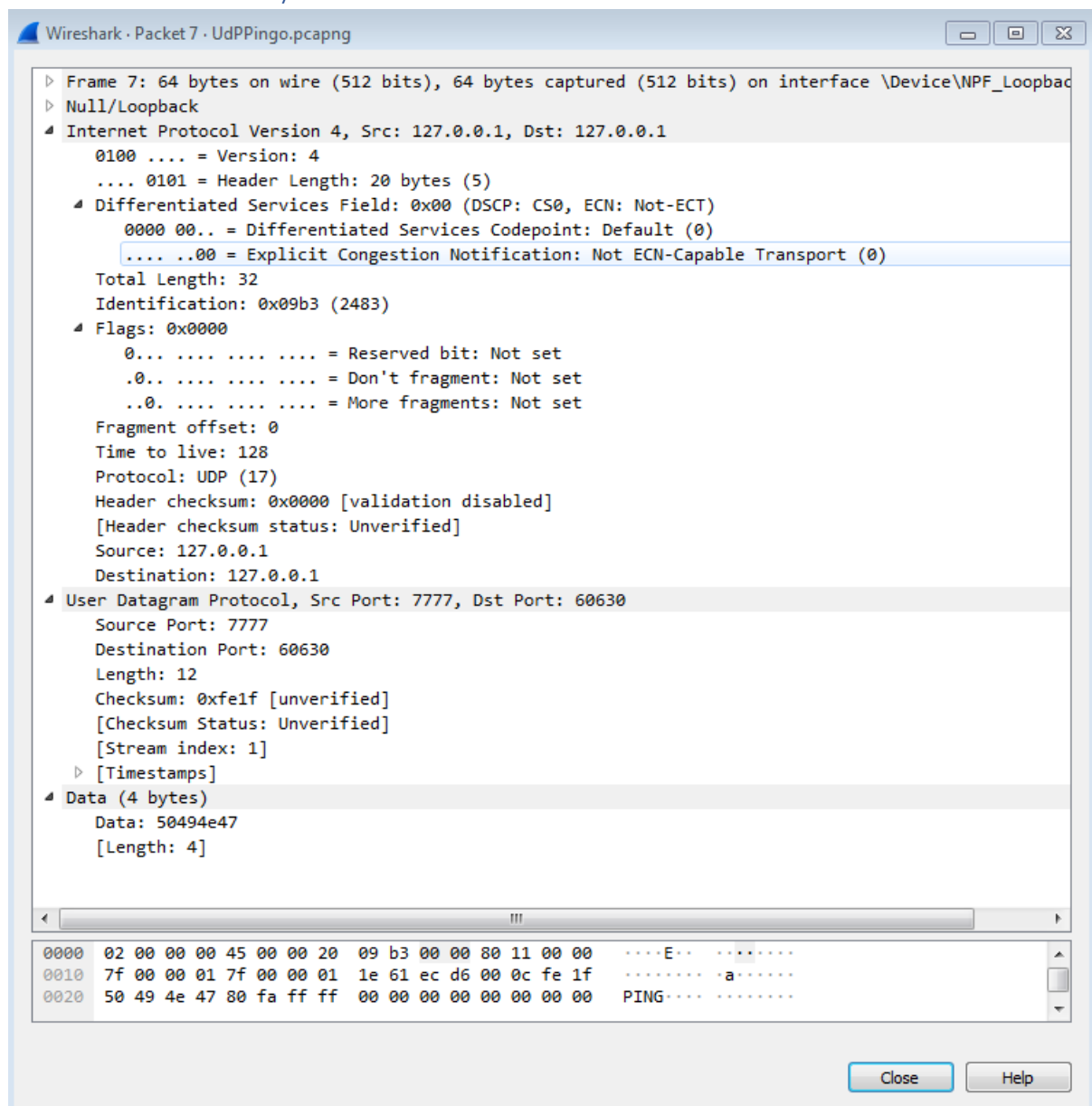
Application Layer		
Header Field	Value	Brief Description
Data	70696e67	Contain message "ping"
Length	4 bytes	Size of the message

Transport Layer

Header Field	Value	Brief Description
Source port	60630	Client port number from where the Segment was sent
Destination Port	7777	Server port number where the Segment will be received
Length (header + Data)	12	Length of Segment after adding the destination port to header of segment
Checksum	0xfelf [unverified]	Convert the Segment to a certain sum of number used check if data reached the destination successfully by checking the sum of message before then and after receive (usually disabled in UDP)
Stream index	1	Stream index here is constructed by Wireshark, and not a field in the actual frame crossing the network
timestamps	0	Time between frames
Network Layer		
Header Field	Value	Brief Description
Protocol version	v.4	IPv4 one of cores of standards-based internetworking methods in the Internet and other packet-switched networks.

Header Length	20 bytes	Contain the IP address of Destination
Total Length	32	Total size of packet
Identification	0x09b2	<ul style="list-style-type: none"> Used to Identifies fragmented Identifies the individual packets that the sender transmits
Flags	0x0000	<p>Used to define which Flag data will use, allowing the sender or receiver to specify which flags should be used so the segment is handled correctly by the other end.</p> <p>Most famous flags "SYN", "ACK ", " FIN" (Mostly in TCP)</p>
Fragments offset	0	Actual offset divided by 8
Time to Live	128	Used to prevent Packet that been sent to address that doesn't exist from looping in the network forever
Protocol	User Datagram Protocol (UDP)	The used protocol for the connection
Header checksum	0x0000	Used to check if data been sent successfully or not
Source	127.0.0.1	IP address of Sender
Destination	127.0.0.1	IP address of receiver

5.2 Received Packet Analysis:



Application Layer		
Header Field	Value	Brief Description
Data	50494e47	Contain message from the server "PING"
Length	4 bytes	Size of the message

Transport Layer		
Header Field	Value	Brief Description
Source port	7777	Server port number from where the Segment was sent
Destination Port	60630	Client port number where the Segment will be received
Length (header + Data)	12	Length of Segment after adding the destination port to header of segment
Checksum	0xfelf [unverified]	Convert the Segment to a certain sum of number used check if data reached the destination successfully by checking the sum of message before then and after receive (usually disabled in UDP)
Stream index	1	Stream index here is constructed by Wireshark, and not a field in the actual frame crossing the network
timestamps	0	Time between frames
Network Layer		
Header Field	Value	Brief Description
Protocol version	v.4	IPv4 one of cores of standards-based internetworking methods

		in the Internet and other packet-switched networks.
Header Length	20 bytes	Contain the IP address of Destination
Total Length	32	Total size of packet
Identification	0x09b3	<ul style="list-style-type: none"> Used to Identifies fragmented Identifies the individual packets that the sender transmits
Flags	0x0000	<p>Used to define which Flag data will use, allowing the sender or receiver to specify which flags should be used so the segment is handled correctly by the other end.</p> <p>Most famous flags "SYN", "ACK ", " FIN" (Mostly in TCP)</p>
Fragments offset	0	Actual offset divided by 8
Time to Live	128	Used to prevent Packet that been sent to address that doesn't exist from looping in the network forever
Protocol	User Datagram Protocol (UDP)	The used protocol for the connection
Header checksum	0x0000	Used to check if data been sent successfully or not

Source	127.0.0.1	IP address of Sender
Destination	127.0.0.1	IP address of receiver

6.Conclusion:

So, we concluded from our example that UDP is best option for real-time application in terms of speed compared to TCP such as voice or video chat, gaming, live conferences, watching online videos, although it loses some packets but it doesn't cause problems to most of application