Assignment - VII

problem statement :-

Write a C/C++ program to analyze following packet format captured through wireshark for wired network
1) FTP 2) IP 3) TCP 4) UDP

Objective :- To understand packet format captured through wireshark for wired network.

Outcome :- students will be able to understand captured packet format through wireshark.

S/W & H/W :- C/C++ compiler, wireshark, monitor, keyboard.

Theory :-

Packet sniffer

A packet sniffer is a computer program or a piece of computer hardware that can intercept & log traffic passing over a digital network or part of network. A data streams flow across the network the sniffer captures each packet & if required decodes packet's raw data, showing the values of various fields in the packet & analyze its content. A packet sniffer a wire-tap device that plays into computer network & eavesdrops on the network traffic.

FTP:-

File Transfer Protocol is standard network protocol used for the transfer of computer files between a client & server on a computer network.

FTP is built on a client-server model architecture using separate control & data connections between client & server.

IP:-

Internet protocol (IP) is principal communication protocol in internet protocol suite for relaying datagrams across network boundaries. IP has task to deliver packets from source to destination solely based on IP addresses in the packet headers. IP defines packets structures that encapsulate the data to be delivered. It also defines addressing methods that are used to label the diagram with source & destination information

TCP :-

TCP segments are sent as internet data-grams. The internet protocol header carries several information fields, including the source & destination host addresses. A TCP header follows the internet header, supplying information specific to TCP. This allows for the existence for the host level protocol other than TCP.

UDP :-

UDP is a connectionless & unreliable transport protocol. The two ports serve to identify the end points within the source & destination machines. User Datagram Protocol is used, in place of TCP, when reliable delivery is not required. However UDP is never used to send important data such as web pages, database information, streaming data/media such as video, audio & others use UDP because it offers speed.

Algorithm :-

1] Start wireshark.
2] Start capturing packets.
3] Stop capturing packets
4] Export as csv file.
5] ~~Ex~~ Open the csv file in cpp program
6] Ask "which protocol packets"
7] Display the count.
8] Exit.

Test cases :-

I/P	Expected O/P	Actual O/P	Result
FTP	count: 4	Count: 4	Success
TCP	count: 17	count: 17	Success
IP	Count: 7	Count: 7	Success
UDP	Count: 979	Count: 979	Success

TCP header

<table>
<tr><td colspan="3">Source Port
Number 2bytes</td><td>Destination Port
Number 2 bytes</td></tr>
<tr><td colspan="4">Sequence Number
4 bytes</td></tr>
<tr><td>data
offset
4bits</td><td>reserved
3 bits</td><td>control
flags
9bits</td><td>Window Size
2bytes.</td></tr>
<tr><td colspan="3">Checksum
2bytes</td><td>Urgent Pointer
2bytes</td></tr>
<tr><td colspan="4">Optional Data
0-40 bytes</td></tr>
</table>

conclusion:-

We learnt how to analyze packet format using wireshark.

```
#include <iostream>
#include<fstream>
#include <iomanip>
#include<string>
using namespace std;

int main() {
 cout << "***** PACKET ANALYZER *****" << endl; // prints !!!Hello World!!!
 string value, sr_no,time,source,destination,info,protocol,len;
 int count=-1,i=0;



 int choice;
 do
 {
  ifstream file("data.csv");
  //Reinitialize Counters
  count=-1;
  i=0;
 cout<<"\nEnter which protocol packets you want to see"<<endl;
 cout<<"1.IP\n2.UDP\n3.TCP\n4.Ethernet\n0Exit!!!\nChoice:"<<endl;
 cin>>choice;
 string protocolChoice; //sting to hold user packet choice
 switch(choice){
 case 1: protocolChoice="ICMPv6";
 break;
 case 2: protocolChoice="UDP";
 break;
 case 3: protocolChoice="TCP";
 break;
 case 4: protocolChoice="ARP";
 break;
 default: protocolChoice="ARP";
 break;
 }
 while(file.good()) //LOOP UNTIL FILE HAS CONTENT
 {
  getline(file,sr_no,','); //GET STRING TILL ,
  getline(file,time,',');
  getline(file,source,',');
  getline(file,destination,',');
  getline(file,protocol,',');
  getline(file,len,',');
  getline(file,info,'\n');

  protocol=string(protocol,1,protocol.length()-2);

  if(protocol=="Protocol"||protocol==protocolChoice)
  {
   cout <<setw(4)<<left<<i++;
   cout <<setw(12)<<left<< string( time, 1, time.length()-2 );
   cout << setw(30)<<left<<string( source, 1, source.length()-2 );
   cout << setw(30)<<left<<string( destination, 1, destination.length()-2 );
   cout <<setw(8)<<left<<protocol;
   cout <<setw(8)<<left<< string( len, 1, len.length()-2 );
   cout << string( info, 1, info.length()-2 )<<"\n";
   count++;
  }
 }
 file.close();
 cout<<"\nTotal Packet Count: "<<count;
 }while(choice!=0);
 return 0;
}
/* output:
 * ***** PACKET ANALYZER *****

Enter which protocol packets you want to see
1.IP
2.UDP
3.TCP
4.Ethernet
0Exit!!!
Choice:
1
0   Time        Source                        Destination                   ProtocolLength  Info
1   0.000000000 fe80::f68e:38ff:fe87:a57e     ff02::1:ff02:21a              ICMPv6  86      Neighbor Solicitation for fe80
::726d:ecff:fe02:21a from f4:8e:38:87:a5:7e
2   0.151808000 fe80::175:6553:3c34:d4f0      ff02::1:ff02:21a              ICMPv6  86      Neighbor Solicitation for fe8
0::726d:ecff:fe02:21a from c8:1f:66:06:4a:84
3   0.245234000 fe80::208:a1ff:fe43:c3c2      ff02::1:ff02:21a              ICMPv6  86      Neighbor Solicitation for fe80
::726d:ecff:fe02:21a from 00:08:a1:43:c3:c2
4   0.301527000 fe80::4046:d001:d60a:e934     ff02::1:ff00:1                ICMPv6  86      Neighbor Solicitation for fe8
0::1 from 00:25:64:92:4d:81
5   0.310878000 fe80::80a7:7d55:7ecf:5582     ff02::1:ff02:21a              ICMPv6  86      Neighbor Solicitation for fe8
0::726d:ecff:fe02:21a from 34:17:eb:9e:8e:45
6   0.382715000 fe80::104b:adee:75e6:c425     ff02::1:ff2f:e430             ICMPv6  86      Neighbor Solicitation for fe
80::a490:6a6c:d52f:e430 from 00:19:d1:45:e9:4b
7   0.486747000 fe80::8e2:220e:db99:187f      ff02::2                       ICMPv6  70      Router Solicitation from c8:e0:e
b:9e:44:9e
8   0.619047000 fe80::adb7:4c35:7a64:621e     ff02::1:ff18:d425             ICMPv6  86      Neighbor Solicitation for fe
80::899f:4a1b:518:d425 from b8:ac:6f:68:65:68
9   0.621767000 fe80::25e2:1c6e:545d:d5ca     ff02::1:ff00:1                ICMPv6  86      Neighbor Solicitation for fe8
0::1 from f0:4d:a2:fd:b3:b3
10  0.879948000 fe80::6600:6aff:fe37:40d9     ff02::1:ff02:22f              ICMPv6  86      Neighbor Solicitation for fe8
0::726d:ecff:fe02:22f from 64:00:6a:37:40:d9
11  0.943252000 fe80::4a4d:7eff:fec6:fe57     ff02::1:ff02:21a              ICMPv6  86      Neighbor Solicitation for fe8
0::726d:ecff:fe02:21a from 48:4d:7e:c6:fe:57
12  0.973236000 fe80::ad92:4946:c11e:bff0     ff02::1:ff00:1                ICMPv6  86      Neighbor Solicitation for fe8
0::1 from f4:8e:38:9d:86:5c
13  1.001717000 fe80::f68e:38ff:fe87:a57e     ff02::1:ff02:21a              ICMPv6  86      Neighbor Solicitation for fe8
0::726d:ecff:fe02:21a from f4:8e:38:87:a5:7e
14  1.158015000 fe80::175:6553:3c34:d4f0      ff02::1:ff02:21a              ICMPv6  86      Neighbor Solicitation for fe
80::726d:ecff:fe02:21a from c8:1f:66:06:4a:84
15  1.164756000 fe80::90c7:9c8e:4162:743a     ff02::16                      ICMPv6  110     Multicast Listener Report Me
```

```
ssage v2
16  1.247232000 fe80::208:a1ff:fe43:c3c2      ff02::1:ff02:21a            ICMPv6  86     Neighbor Solicitation for fe8
0::726d:ecff:fe02:21a from 00:08:a1:43:c3:c2
17  1.299874000 fe80::4046:d001:d60a:e934     ff02::1:ff00:1              ICMPv6  86     Neighbor Solicitation for fe8
0::1 from 00:25:64:92:4d:81
18  1.334884000 fe80::80a7:7d55:7ecf:5582     ff02::1:ff02:21a            ICMPv6  86     Neighbor Solicitation for fe
80::726d:ecff:fe02:21a from 34:17:eb:9e:8e:45
19  1.381157000 fe80::104b:adee:75e6:c425     ff02::1:ff2f:e430           ICMPv6  86     Neighbor Solicitation for fe
80::a490:6a6c:d52f:e430 from 00:19:d1:45:e9:4b
20  1.410771000 fe80::adb7:4c35:7a64:621e     ff02::1:ff11:4e6f           ICMPv6  86     Neighbor Solicitation for fe
80::5058:2741:6f11:4e6f from b8:ac:6f:68:65:68
21  1.422139000 fe80::ec3b:be3b:a1cf:b8dc     ff02::1:ff64:621e           ICMPv6  86     Neighbor Solicitation for fe
80::adb7:4c35:7a64:621e from 28:d2:44:f6:d0:71
22  1.464011000 fe80::c2c9:76ff:fe50:72f9     ff02::2                     ICMPv6  70     Router Solicitation from c0:c9:7
6:50:72:f9
23  1.472534000 fe80::adb7:4c35:7a64:621e     ff02::1:ff1c:b39b           ICMPv6  86     Neighbor Solicitation for f
e80::d107:c499:311c:b39b from b8:ac:6f:68:65:68
24  1.502391000 fe80::4a4d:7eff:feca:8004     ff02::1:ff02:21a            ICMPv6  86     Neighbor Solicitation for fe8
0::726d:ecff:fe02:21a from 48:4d:7e:ca:80:04
25  1.614264000 fe80::4a4d:7eff:fec6:ff33     ff02::1:ff02:21a            ICMPv6  86     Neighbor Solicitation for fe80
::726d:ecff:fe02:21a from 48:4d:7e:c6:ff:33
26  1.639345000 fe80::221:9bff:fe6e:4b01      ff02::1:ff02:21a            ICMPv6  86     Neighbor Solicitation for fe8
0::726d:ecff:fe02:21a from 00:21:9b:6e:4b:01
27  1.880789000 fe80::6600:6aff:fe37:40d9     ff02::1:ff02:22f            ICMPv6  86     Neighbor Solicitation for fe8
0::726d:ecff:fe02:22f from 64:00:6a:37:40:d9
28  1.998620000 fe80::adb7:4c35:7a64:621e     ff02::1:ffa7:7fb2           ICMPv6  86     Neighbor Solicitation for fe
80::a1fb:332b:83a7:7fb2 from b8:ac:6f:68:65:68
29  2.003773000 fe80::f68e:38ff:fe87:a57e     ff02::1:ff02:21a            ICMPv6  86     Neighbor Solicitation for fe8
0::726d:ecff:fe02:21a from f4:8e:38:87:a5:7e
30  2.028027000 fe80::e298:61ff:fe35:9a26     ff02::1:ff64:621e           ICMPv6  86     Neighbor Solicitation for fe
80::adb7:4c35:7a64:621e from e0:98:61:35:9a:26
31  2.040149000 fe80::f68e:38ff:fe87:a56a     ff02::1:ff02:21a            ICMPv6  86     Neighbor Solicitation for fe8
0::726d:ecff:fe02:21a from f4:8e:38:87:a5:6a
32  2.107577000 fe80::b283:feff:fe4d:f1c9     ff02::1:ff02:21a            ICMPv6  86     Neighbor Solicitation for fe8
0::726d:ecff:fe02:21a from b0:83:fe:4d:f1:c9
33  2.162415000 fe80::90c7:9c8e:4162:743a     ff02::16                    ICMPv6  110    Multicast Listener Report Me
ssage v2
34  2.181982000 fe80::175:6553:3c34:d4f0      ff02::1:ff02:21a            ICMPv6  86     Neighbor Solicitation for fe
80::726d:ecff:fe02:21a from c8:1f:66:06:4a:84

Total Packet Count: 34
Enter which protocol packets you want to see
1.IP
2.UDP
3.TCP
4.Ethernet
0Exit!!!
Choice:
2
0   Time        Source                      Destination                 ProtocolLength  Info

Total Packet Count: 0
Enter which protocol packets you want to see
1.IP
```

```
2.UDP
3.TCP
4.Ethernet
0Exit!!!
Choice:
3
0   Time        Source                      Destination                 ProtocolLength  Info
1   0.243260000 216.58.197.68               10.10.14.151                TCP     66      https > 51709 [FIN, ACK] Seq=1 A
ck=1 Win=175 Len=0 TSval=2559300079 TSecr=23747257
2   0.438095000 108.168.177.14              10.10.13.238                TCP     103     [TCP segment of a reassembled P
DU]
3   0.746828000 192.168.16.254              10.10.10.28                 TCP     60      57777 > etftp [RST] Seq=1 Win=58
40 Len=0
4   0.855756000 64.233.188.188              10.10.15.48                 TCP     97      hpvroom > 39687 [PSH, ACK] Seq
=1 Ack=1 Win=175 Len=31 TSval=2933171628 TSecr=49981356
5   1.839024000 118.214.135.85              10.10.12.0                  TCP     60      https > 50976 [FIN, ACK] Seq=32
Ack=1 Win=980 Len=0
6   1.839028000 118.214.135.85              10.10.12.0                  TCP     60      https > 50977 [FIN, ACK] Seq=32
Ack=1 Win=980 Len=0
7   1.886438000 192.168.3.254               192.168.3.211               TCP     62      ndl-aas > fnet-remote-ui [SYN, AC
K] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1
8   1.888346000 192.168.3.254               192.168.3.211               TCP     60      ndl-aas > fnet-remote-ui [ACK] Se
q=1 Ack=211 Win=30016 Len=0




Total Packet Count: 8
 */
```