Ethernet an Overview

Before discussing automotive special use cases we can have a look on normal Ethernet based networking with respect to OSI model. Robert Metcalfe and his coworkers at Xerox designed Ethernet for creating small networks for computers (basically LANs) . In 1980 The first Ethernet standard was published by Digital Equipment Corporation, Intel, and Xerox (DIX) . In 1985, the Institute of Electrical and Electronics Engineers (IEEE) standards committee for Local and Metropolitan Networks published standards for LANs. The standard for Ethernet is 802.3 and the other similar standards was starting with 802. The IEEE standards were compatible with those of the International Standards Organization (ISO) and OSI model. Ethernet operates in the lower two layers of the OSI model , the Data Link layer and the Physical layer.

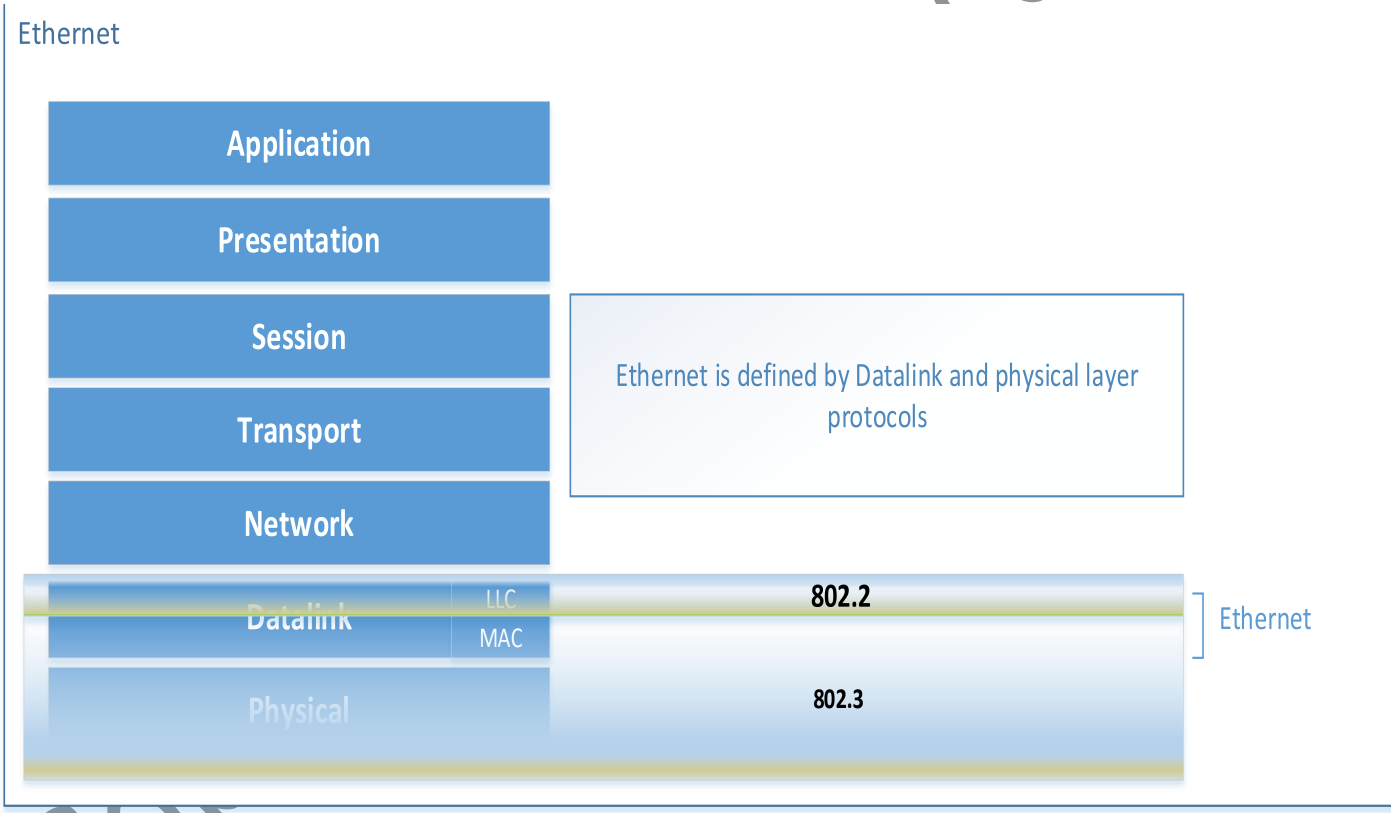


Fig -3.

Physical Layer

Physical layer is the lowest layer of the OSI model. It will be responsible for sending data on the actual transport medium in the bits level and deals with the setup of physical connection to the network and with transmission and reception of signals. This layer may be implemented by a PHY chip. The physical layer consists of the electronic circuit transmission technologies of a network The physical layer provides an electrical, mechanical, and procedural interface to the transmission medium. Main functionalities of physical layer include the following

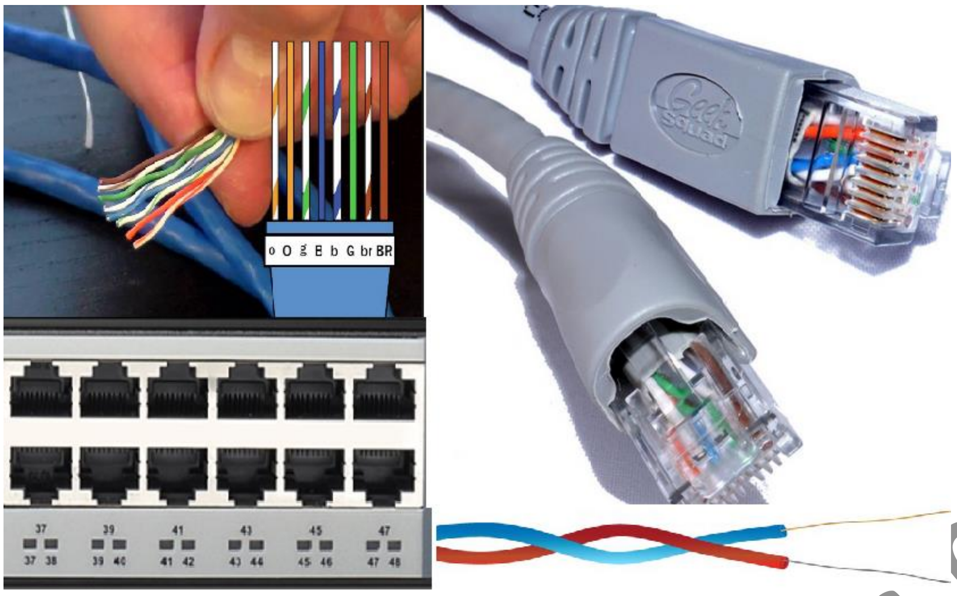


Fig-4.

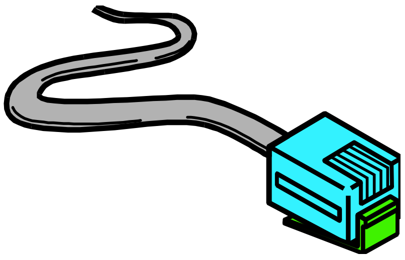
1. **Transfer of Bits:** Data in this layer consists of stream of bits. The bits must be encoded into signals for transmission. It defines the type of encoding i.e. how 0's and 1's are changed to signal.
2. **Data Rate:** This layer defines the rate of transmission which is the number of bits per second.
3. **Synchronization:** It deals with the synchronization of the transmitter and receiver. The sender and receiver are synchronized at bit level.
4. **Interface:** The physical layer defines the transmission interface between devices and transmission medium.
5. **Line Configuration:** This layer connects devices with the medium: Point to Point configuration and Multipoint configuration.
6. **Topologies:** Devices must be connected using the following topologies: Mesh, Star, Ring and Bus.
7. **Transmission Modes:** Physical Layer defines the direction of transmission between two devices: Simplex, Half Duplex, Full Duplex.
8. Deals with baseband and broadband transmission.

Physical layer will be deciding what will be the signaling frequency used, at what SNR and at which voltage level ‘0’ and ‘1’ to be transmitted , which connector and cabling should use like that.

The differences between standard Ethernet, Fast Ethernet, Gigabit Ethernet, and 10 Gigabit Ethernet occur at the Physical layer, often referred to as the Ethernet PHY.

- 10 Mbps - 10Base-T Ethernet

- 100 Mbps - Fast Ethernet
- 1000 Mbps - Gigabit Ethernet
- 10 Gbps - 10 Gigabit Ethernet



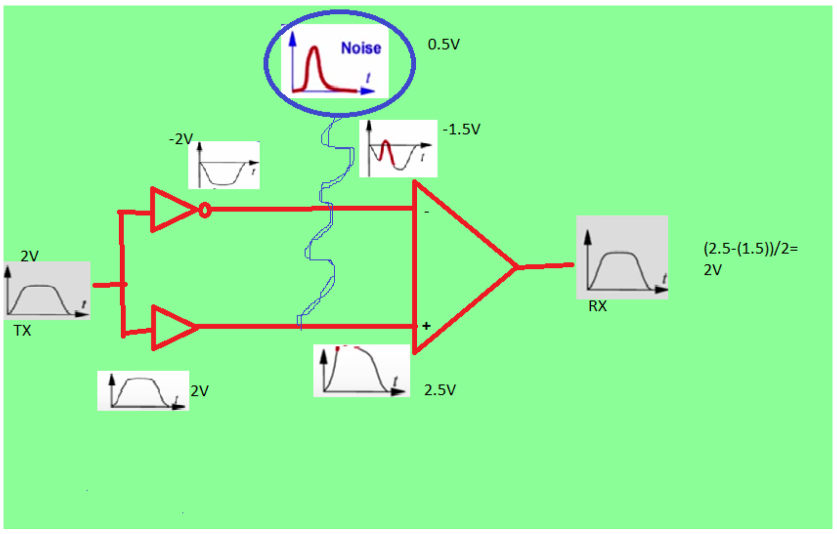
Generally, PHY are named by their specifications eg 10BaseT

- 10, 100, 1000, 10G, ... – the nominal, usable speed at the top of the physical layer (no suffix = megabit/s, G = gigabit/s), excluding line codes but including other physical layer overhead (preamble, SFD, IPG); some WAN PHYs (W) run at slightly reduced bitrates for compatibility reasons; encoded PHY sublayers usually run at higher bitrates
- BASE, BROAD, PASS – indicates baseband, broadband, or passband signaling respectively
- -T, -S, -L, -E, -Z, -C, -K, -H ... – medium: T = twisted pair, S = 850 nm short wavelength (multi-mode fiber), L = 1300 nm long wavelength (mostly single-mode fiber), E or Z = 1500 nm extra long wavelength (single-mode), B = bidirectional fiber (mostly single-mode) using WDM, P = passive optical (PON), C = copper/twinax, K = backplane, 2 or 5 or 36 = coax with 185/500/3600 m reach (obsolete), F = fiber, various wavelengths, H = plastic optical fiber
- X, R – PCS encoding method (varying with the generation): X for 8b/10b block encoding (4B5B for Fast Ethernet), R for large block encoding (64b/66b)
- 1, 2, 4, 10 – for LAN PHYs indicates number of lanes used per link.

For 10 Mbit/s, no encoding is indicated as all variants use Manchester code. Most twisted pair layers use unique encoding, so most often just -T is used.

Many Ethernet adapters and switch ports support multiple speeds by using auto negotiation to set the speed and duplex for the best values supported by both connected devices. While this can practically be relied on for Ethernet over twisted pair, few optical-fiber ports support multiple speeds .Automotive domain will be using BroadR Reach as physical layer which will be carrying data in a single twisted pair cable (BR+ and BR-) .We will be discussing more about automotive physical layer in the upcoming chapter.

How differential pair reduce common noise?



As mentioned in the above figure Ethernet will be using differential signaling. Differential is method of electrically transmitting information through two complementary signals. Means data is send over the one line and exact opposite on the other. Even though we are using two lines for sending a data it help us to reduce common mode noise introduced as you can see from the figure. For example if the 2v data we are sending over differential pair as 2V and (-2V) in the receiving end the data received will be 2V ((2- -2)/2) without any noise. Consider a 0.5 v noise introduced on both the lines ,now also at the receiving end overall effect will be same ((2.5- -1.5)/2) 2V.

Data Link Layer

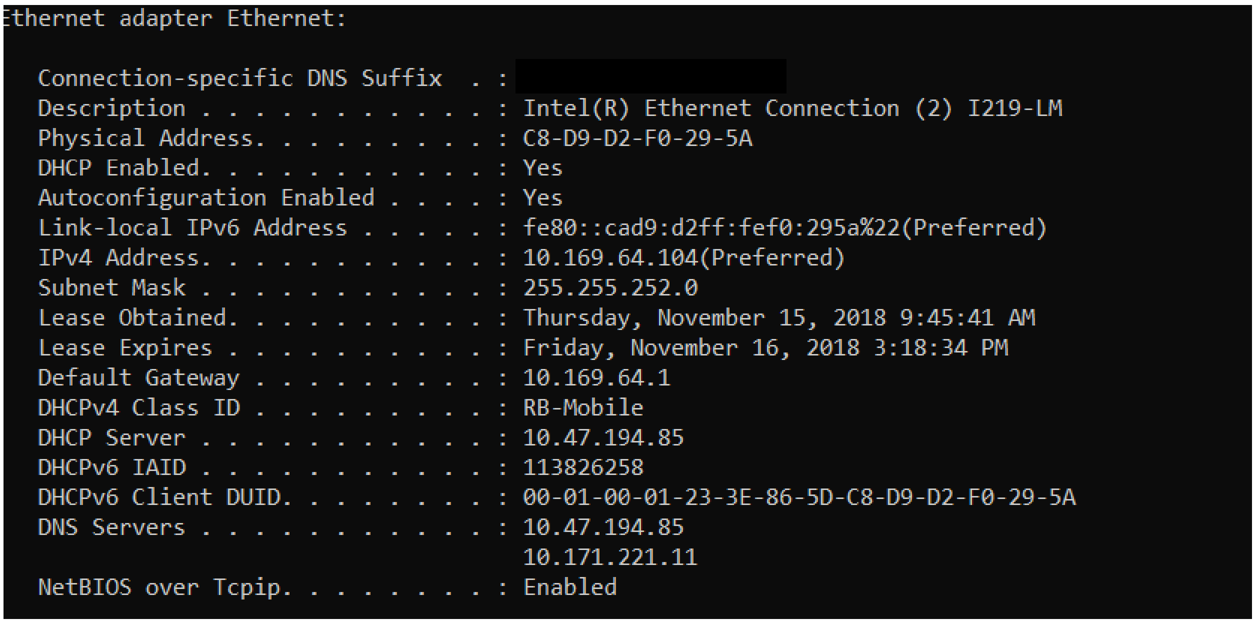
Data Link Layer is the second layer in OSI model. Data link layer comprised of LLC and MAC .It is the layer between network and physical layer. MAC layer types include Ethernet and 802.11 wireless specifications. The Data Link sublayers contribute significantly to technological compatibility and computer communications. The MAC sublayer is concerned with the physical components that will be used to communicate the information and prepares the data for transmission over the media. The Logical Link Control (LLC) sublayer remains relatively independent of the physical equipment that will be used for the communication process. The data link layer provides the functional and procedural means to transfer data between network entities and might provide the means to detect and possibly correct errors that may occur in the physical layer. The uppermost sublayer, LLC, multiplexes protocols running at the top of data link layer, and optionally provides flow control, acknowledgment, and error notification. The LLC provides addressing and control of the data link. It specifies which mechanisms are to be used for addressing stations over the transmission medium and for controlling the data exchanged between the originator and recipient machines.When a packet arrives in a network, it is the responsibility of DLL to transmit it to the Host using its MAC address.

Data Link Layer is divided into two sub layers :

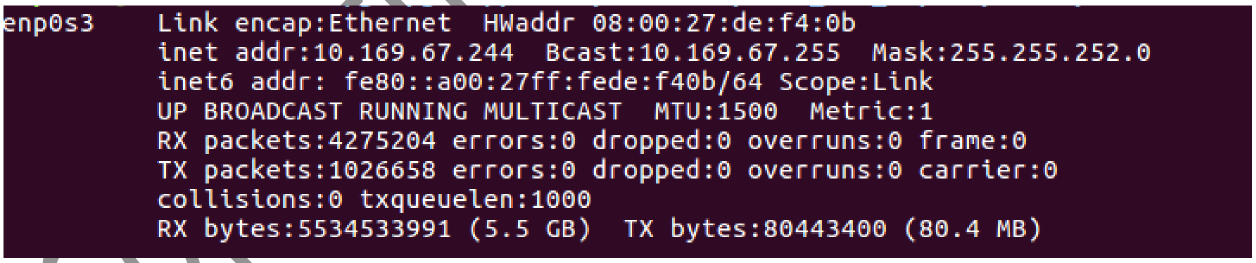
1. Logical Link Control (LLC)
2. Media Access Control (MAC)

The packet received from Network layer is further divided into frames depending on the frame size of NIC(Network Interface Card). DLL also encapsulates Sender and Receiver's MAC address in the header.

The Receiver's MAC address is obtained by placing an ARP(Address Resolution Protocol) request onto the wire asking "Who has that IP address?" and the destination host will reply with its MAC address.



```
Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : [redacted]
   Description . . . . . . . . . . . : Intel(R) Ethernet Connection (2) I219-LM
   Physical Address. . . . . . . . . : C8-D9-D2-F0-29-5A
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::cad9:d2ff:fef0:295a%22(Preferred)
   IPv4 Address. . . . . . . . . . . : 10.169.64.104(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.252.0
   Lease Obtained. . . . . . . . . . : Thursday, November 15, 2018 9:45:41 AM
   Lease Expires . . . . . . . . . . : Friday, November 16, 2018 3:18:34 PM
   Default Gateway . . . . . . . . . : 10.169.64.1
   DHCPv4 Class ID . . . . . . . . . : RB-Mobile
   DHCP Server . . . . . . . . . . . : 10.47.194.85
   DHCPv6 IAID . . . . . . . . . . . : 113826258
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-23-3E-86-5D-C8-D9-D2-F0-29-5A
   DNS Servers . . . . . . . . . . . : 10.47.194.85
                                       10.171.221.11
   NetBIOS over Tcpip. . . . . . . . : Enabled
```



```
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:de:f4:0b
          inet addr:10.169.67.244  Bcast:10.169.67.255  Mask:255.255.252.0
          inet6 addr: fe80::a00:27ff:fede:f40b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4275204 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1026658 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5534533991 (5.5 GB)  TX bytes:80443400 (80.4 MB)
```

The functions of the data Link layer are :

1. **Framing:** Framing is a function of the data link layer. It provides a way for a sender to transmit a set of bits that are meaningful to the receiver. This can be accomplished by attaching special bit patterns to the beginning and end of the frame.
2. **Physical addressing:** After creating frames, Data link layer adds physical addresses (MAC address) of sender and/or receiver in the header of each frame.
3. **Error control:** Data link layer provides the mechanism of error control in which it detects and retransmits damaged or lost frames.

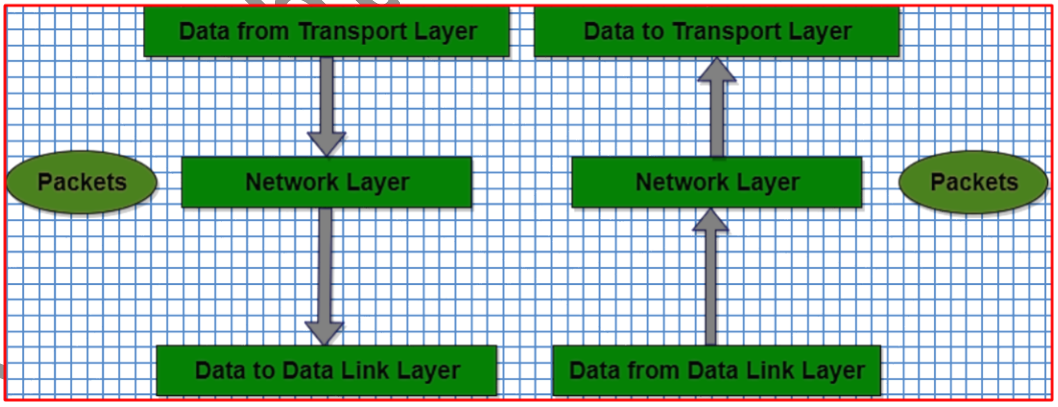
4. **Flow Control:** The data rate must be constant on both sides else the data may get corrupted thus , flow control coordinates that amount of data that can be sent before receiving acknowledgement.
5. **Access control:** When a single communication channel is shared by multiple devices, MAC sub-layer of data link layer helps to determine which device has control over the channel at a given time.

Network Layer

Network layer is responsible for the transmission of data from one host to the other located in different networks. Network layer plays an important role in packet routing, which will be finding the fastest and shortest route between hosts. Network layer will be adding Source and destination IP address on the packet.

The functions of the Network layer are :

1. **Routing:** The network layer protocols determine which route is suitable from source to destination. This function of network layer is known as routing.
2. **Logical Addressing:** In order to identify each device on internetwork uniquely, network layer defines an addressing scheme. The sender & receiver's IP address are placed in the header by network layer. Such an address distinguishes each device uniquely and universally. It translates logical network address into physical address. Concerned with circuit, message or packet switching.
3. Routers and gateways operate in the network layer. Mechanism is provided by Network Layer for routing the packets to final destination.
4. Connection services are provided including network layer flow control, network layer error control and packet sequence control.



While talking about routing it is important to have an idea about ipv4 and ipv6 addressing.

Like our postal mail system (each house have house number and address) all the computer/network devices connected to internet will have an address for data transfer. IPv4 is the 4th version of Internet Protocol. And it's a connectionless protocol in packet switched network which used for reliable data delivery across the network devices.IPv4 is defined and specified in IETF publication RFC 791. In IPv4 addressing each address is a 32 bit number represented as 4 octets separated by a decimal point eg:192.168.2.7 .Since it uses 32 bit it can address nearly 4 billion (2^32 addresses).Because of the increase in network devices like personal computers and laptops IPv4 address will exhaust easily. So Internet Engineering Task Force (IETF) introduced IPv6 which uses 128 bit addressing. Using IPv6 we can address 2^128 devices which is large enough for present world. Since IPv6 uses 128 bit for the easiest representation we use hexa-colon representation. 128 bit will be represented as 8 groups of 16 bits (2 bytes) .Eg :1234:abcd:5678:def0:90ab:cd12:3478:ffff , in other words 16 bytes. For basic understanding of IPv4 addresses it is necessary have to overview of gateway address, subnet mask and different addresses classes (A,B,C,D,E) and multicast and unicast addresses too. Describing all this basics will be an other book ☺ .So we can brief this discussion by just seeing the basic IPv4 and IPv6 differences .

IPv4	IPv6
Using 32 bit addressing.	Using 128 bit addressing.
It can generate about 4 billion addresses.	It can generate 2^128 addresses ,which is quite large.
Application have to take care of secuirity.	IPSEC is inbuilt security feature in the IPv6 protocol.
It has broadcast Message Transmission Scheme.	In IPv6 multicast and any cast message transmission scheme is available.
No encryption and authentication facility available as part of IPv4 protocol.	Encryption and authentication facility available as part of IPv6 protocol.
Checksum filed is available.	Checksum field is not available.
Fragmentation performed by Sender and forwarding routers.	In IPv6 fragmentation performed only by sender.
Representation in decimal -dot	Representation in hexa-colon representation

Transport Layer

Transport layer plays an important role in Ethernet communication. Transport layer is responsible for data exchange with application layer and network layer. The data in the transport layer referred to as segments. Transport layer is responsible for fragmenting the application data in to segments and sending .It will also take care of flow& error control. It also adds Source and Destination port number in its header and forwards the segmented data to the Network Layer . The sender need to know the port number associated with the receiver's application. For some application the destination port might be predefined (for example 80 for http).

Transport layer mainly provide connection oriented and connection less services. In connection oriented service the hosts communicating will establishing a connection first .Then it will transfer the data , after transferring data the connection will be terminated. For each data transfer there will be corresponding acknowledgement for reliable data transfer.Eg: TCP ,

In connection less service in a single phase I will transfer the data, so it will be faster and there will not be any acknowledge back. But it may not be reliable. Eg: UDP

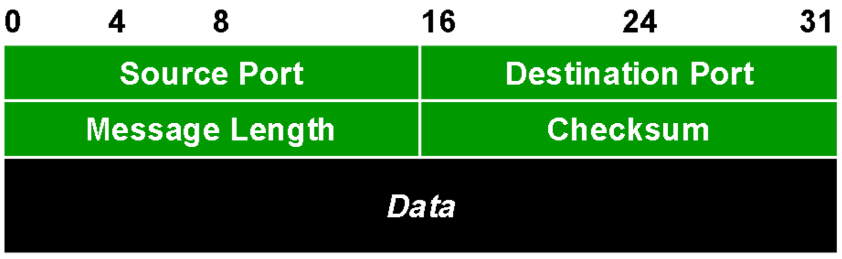
We can have a quick look on UDP and TCP since most of the application protocols are implemented on top of TCP and UDP transport. Understanding the communication in the transport layer using network sniffers like Wireshark is necessary for debugging most of the application communicating over Ethernet.

User Datagram Protocol (UDP)

UDP is a connectionless unreliable transport level protocol. But it is very much useful for low latency data transfer. Packets may be lost or delivered out of order. Users exchange datagrams (not streams).And data will not be buffered before transmission (UDP accepts data and transmits immediately). This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed.

Data coming from the application layer will get added with UDP header before reaching the network layer during the data transmission. UDP header will be containing very minimal data as follows.

Fig. UDP header



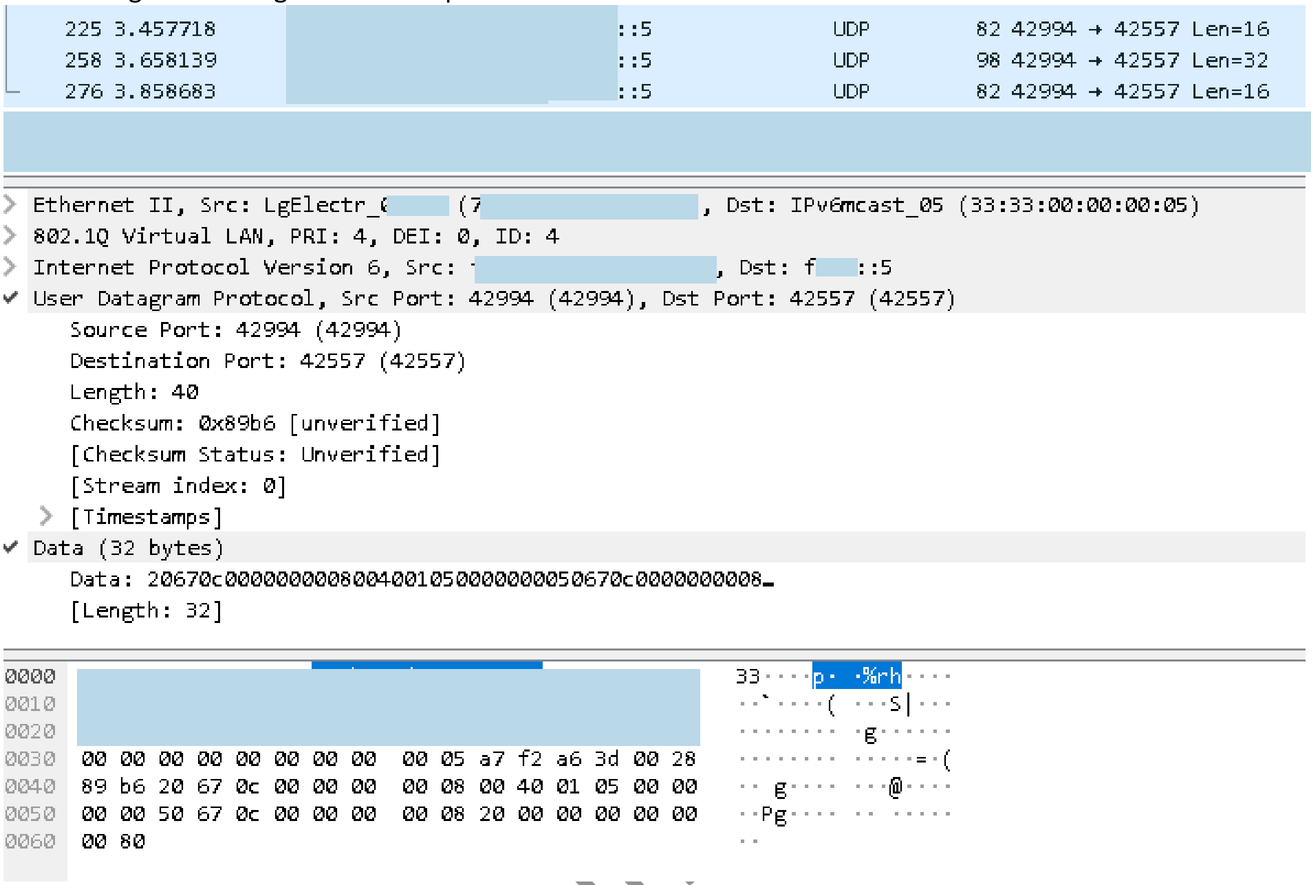
Source Port: Source Port is an optional field, when meaningful, it indicates the port of the sending process, and may be assumed to be the port to which a reply should be addressed in the absence of any other information. If not used, a value of zero is inserted.

Destination Port: has a meaning within the context of a particular internet destination address.

Length:is the length in octets of this user datagram including this header and the data. (This means the minimum value of the length is eight.)

Checksum: is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets. Checksum will help us for error detection.

Below image is showing wireshark capture of UDP header.


```
  225 3.457718   [redacted]::5      UDP    82 42994 → 42557 Len=16
  258 3.658139   [redacted]::5      UDP    98 42994 → 42557 Len=32
  276 3.858683   [redacted]::5      UDP    82 42994 → 42557 Len=16

[redacted]

> Ethernet II, Src: LgElectr_[redacted] (7[redacted], Dst: IPv6mcast_05 (33:33:00:00:00:05)
> 802.1Q Virtual LAN, PRI: 4, DEI: 0, ID: 4
> Internet Protocol Version 6, Src: [redacted], Dst: f[redacted]::5
✓ User Datagram Protocol, Src Port: 42994 (42994), Dst Port: 42557 (42557)
     Source Port: 42994 (42994)
     Destination Port: 42557 (42557)
     Length: 40
     Checksum: 0x89b6 [unverified]
     [Checksum Status: Unverified]
     [Stream index: 0]
   > [Timestamps]
✓ Data (32 bytes)
     Data: 20670c0000000008004001050000000050670c0000000008…
     [Length: 32]

0000  [redacted]                                          33····p· ·%rh····
0010                                                      ··`····( ···S|···
0020                                                      ········ ·g······
0030  00 00 00 00 00 00 00 00  00 05 a7 f2 a6 3d 00 28   ········ ·····=·(
0040  89 b6 20 67 0c 00 00 00  00 08 00 40 01 05 00 00   ·· g···· ···@····
0050  00 00 50 67 0c 00 00 00  00 08 20 00 00 00 00 00   ··Pg···· ·· ·····
0060  00 80                                              ··
```

We can have a quick look on UDP sample server client program , apis used are not describing here , which will be well explained in Linux man pages.

Click Below to follow us ,and stay updated.

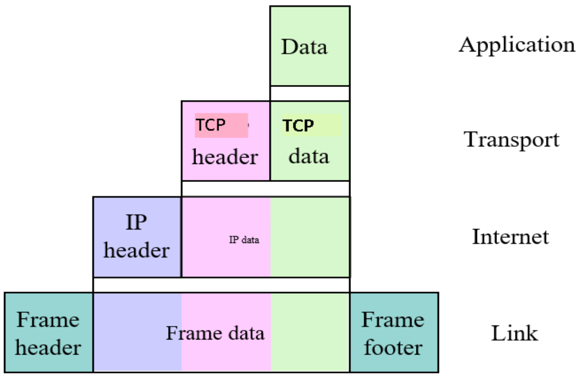
```c
/* UDP Server.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#define PORT     7777
#define MAX 1024
int main() {
    int sockfd;
    char buffer[MAX];
    int len, n;
    char *msg = "msg from server";
    struct sockaddr_in servaddr, cliaddr;
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);// Creating socket
    if ( sockfd< 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }
    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));
    servaddr.sin_family = AF_INET; // Filling server detais
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);
    if ( bind(sockfd, (const struct sockaddr *)&servaddr,sizeof(servaddr)) < 0 )
{// Bind the socket with the server address
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    n = recvfrom(sockfd, (char *)buffer, MAX,MSG_WAITALL, ( struct sockaddr *)
&cliaddr,&len);
    buffer[n] = '\0';
    printf("MSG from Client : %s\n", buffer);
    sendto(sockfd, (const char *)msg, strlen(msg),MSG_CONFIRM, (const struct
sockaddr *) &cliaddr,len);
    printf("msg message sent.\n");
    return 0;
}
```

```c
/* Client,c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#define PORT     7777 /*Macro defining UDP port*/
#define MAX 256
int main() {
    int sockfd;
    char buffer[MAX];
    int n, len;
    char *msg = "Message from client";
    struct sockaddr_in   servaddr;
    // Creating socket file descriptor
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (  0>sockfd ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }
    // Filling server details
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = INADDR_ANY;
    sendto(sockfd, (const char *)msg, strlen(msg),
        MSG_CONFIRM, (const struct sockaddr *) &servaddr,
            sizeof(servaddr));
    printf("msg message sent.\n");
    n = recvfrom(sockfd, (char *)buffer, MAX,
                MSG_WAITALL, (struct sockaddr *) &servaddr,
                &len);
    buffer[n] = '\0';
    printf("Server : %s\n", buffer);
    close(sockfd);
    return 0;
}
```

<u>Transmission Control Protocol (TCP)</u>

<u>(with Example Client Server Program)</u>

Transmission Control Protocol (TCP) is a transport layer protocol .It is reliable protocol and it is a connection oriented protocol too. Today Most of the application layer reliable protocol are implemented on top of TCP transport.



Transmission Control Protocol accepts data from application layer , divides it into chunks, and adds a TCP header creating a TCP segment. The TCP segment is then encapsulated into an Internet Protocol (IP) datagram, and exchanged with peers.

Consider the following TCP sever client program based on Linux platform for understanding TCP socket programming.

Example TCP-Server Program

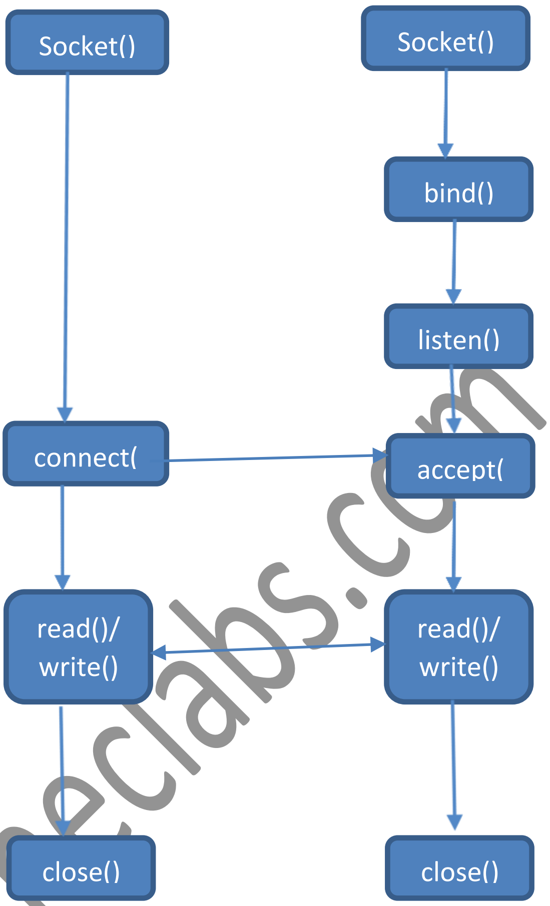
```
/*
TCP_Server.c


*/
#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>
#include <sys/types.h>
#include <time.h>
#define RELIABLE_PORT (5000)


int main(int argc, char *argv[])
{
    int listenfd = 0, connfd = 0;
    struct sockaddr_in serv_addr;
    char msgBuf[1025];
    time_t time_ticks;

    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    memset(&serv_addr, '0', sizeof(serv_addr));
    memset(msgBuf, '0', sizeof(msgBuf));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        serv_addr.sin_port = htons(RELIABLE_PORT);

    bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));

    listen(listenfd, 10);

    while(1)
    {
    printf("[server] Waiting for client...\r\n");
        connfd = accept(listenfd, (struct sockaddr*)NULL, NULL);

    printf("[server] client connection accepted\r\nSending data...\r\n");
        time_ticks = time(NULL);
        snprintf(msgBuf, sizeof(msgBuf), "%.24s\r\n", ctime(&time_ticks));
        write(connfd, msgBuf, strlen(msgBuf));
        if(0>=read(connfd, msgBuf, strlen(msgBuf))){
    printf("[server] Closing client connection\r\n");
    close(connfd);
    }


       sleep(1);
     }
}

/* Output

============================
./server
[server] Waiting for client...
[server] client connection accepted
Sending data...
[server] Closing client connection
[server] Waiting for client...

*/
```

The client will be connecting to the TCP server socket RELIABLE_PORT mentioned in the server code.

The Server-client Program is self-explanatory from the following diagram .For printing errors it is better to use perror() ,instead of printf in the error cases.

All the APIs used are available in Linux man page .Example # man bind ,

will give all the details about the bind call.(It is always better to follow the man page , that is the reason I am not explaining all the



A sample client which will be showing data received from the server is given below.

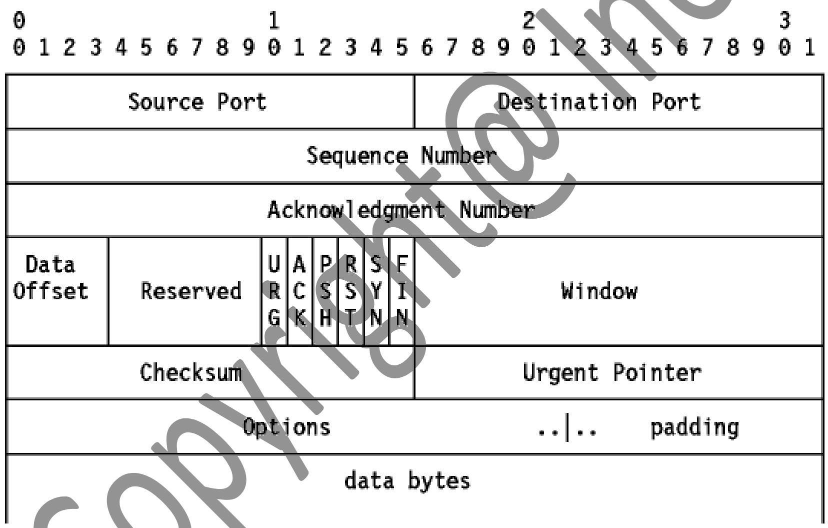
```
/*
TCP_Client.c
*/
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>
```

```c
int main(int argc, char *argv[])
{
    int sockfd = 0, n = 0;
    char msgBuf[1024];
    struct sockaddr_in serv_addr;

    if(argc != 2)
    {
        printf("\n Usage: %s <ip of server> \n",argv[0]);
        return 1;
    }

    memset(msgBuf, '0',sizeof(msgBuf));
    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Error : Could not create socket \n");
        return 1;
    }

    memset(&serv_addr, '0', sizeof(serv_addr));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(5000);

    if(inet_pton(AF_INET, argv[1], &serv_addr.sin_addr)<=0)
    {
        printf("\n inet_pton error occured\n");
        return 1;
    }

    printf("[client] Connecting to server...\r\n");
    if( connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
       printf("\n Error : Connect Failed \n");
       return 1;
    }

    printf("[client] Connected to server...\r\n");

    n = read(sockfd, msgBuf, sizeof(msgBuf)-1);
   if(n>0) {
        msgBuf[n] = 0;
    printf("[client] recieved data from server \r\n%s...\r\n",msgBuf);
    }
  else
    {
        printf("\n Read error \n");
    }
    close(sockfd);
    return 0;
}

/*
Output
==============
./client 127.0.0.1
[client] Connecting to server...
[client] Connected to server...
[client] recieved data from server
Mon Apr  8 11:31:45 2019
...

*/
```

For understanding TCP connection establishment and communication consider the following section which including flow graph captured from wireshark for a sample tcp server client communication.

<u>Understanding TCP , 3 way handshaking using Wireshark</u>

As we all know TCP is a reliable connection oriented protocol .So for establishing connection ,reliable data transfer and disconnecting the connection It using many types of flags (1 bit Boolean).



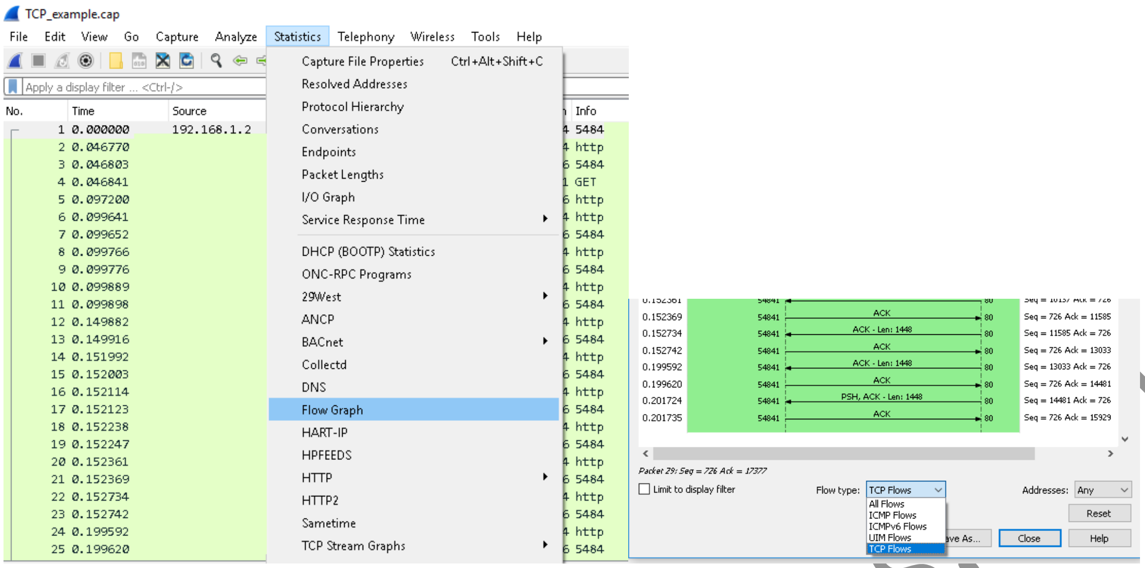
Following are the three important flags.

- **SYN** - (Synchronize) Initiates a connection
- **FIN** - (Final) Cleanly terminates a connection
- **ACK** - Acknowledges received data

All the further explanation are based on the Wireshark packet captured from the previous example.

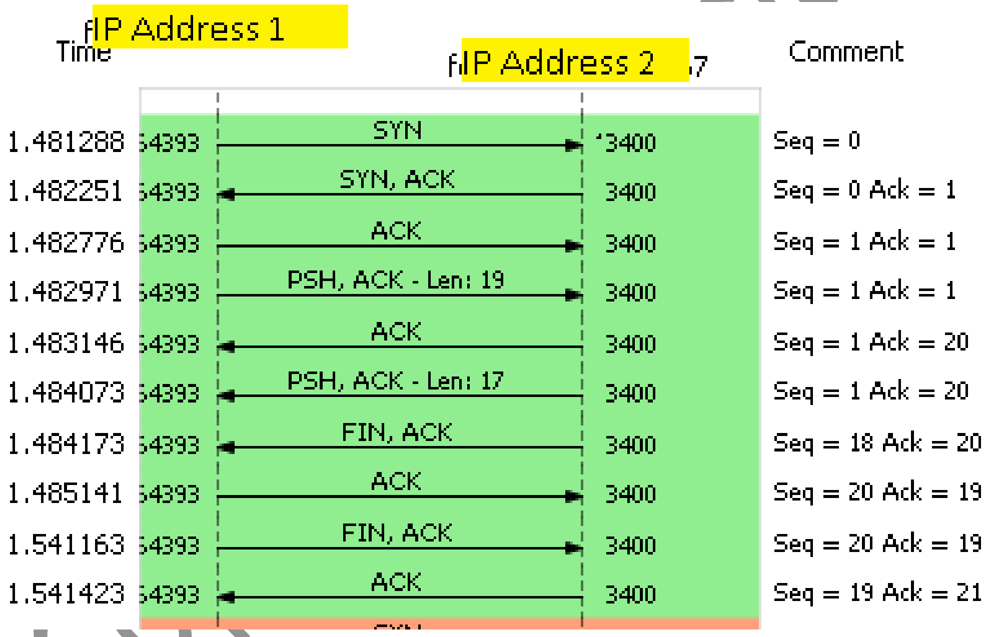
But important fields are included here in this explanation .For better understanding of TCP protocol consider the following diagram obtained from Wireshark.

(Statistics -> Flow Graph) and filter the results with TCP Flows.



```
No.   Time       Source
 1 0.000000  192.168.1.2
 2 0.046770
 3 0.046803
 4 0.046841
 5 0.097200
 6 0.099641
 7 0.099652
 8 0.099766
 9 0.099776
10 0.099889
11 0.099898
12 0.149882
13 0.149916
14 0.151992
15 0.152003
16 0.152114
17 0.152123
18 0.152238
19 0.152247
20 0.152361
21 0.152369
22 0.152734
23 0.152742
24 0.199592
25 0.199620
```

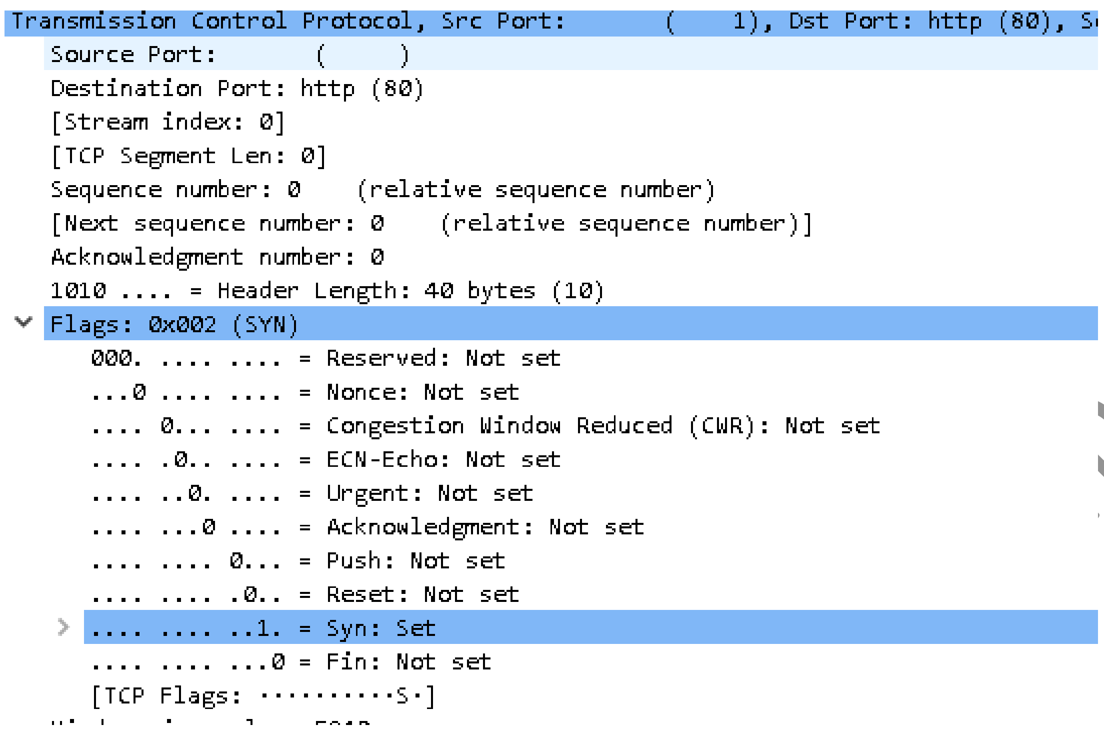
```
0.152369  54841  ACK ->                     80  Seq = 726 Ack = 11585
0.152734  54841  <- ACK - Len: 1448         80  Seq = 11585 Ack = 726
0.152742  54841  ACK ->                     80  Seq = 726 Ack = 13033
0.199592  54841  <- ACK - Len: 1448         80  Seq = 13033 Ack = 726
0.199620  54841  ACK ->                     80  Seq = 726 Ack = 14481
0.201724  54841  <- PSH, ACK - Len: 1448    80  Seq = 14481 Ack = 726
0.201735  54841  ACK ->                     80  Seq = 726 Ack = 15929

Packet 29: Seq = 726 Ack = 17377
Flow type: TCP Flows     Addresses: Any
```



```
1.481288  54393  SYN ->                    3400  Seq = 0
1.482251  54393  <- SYN, ACK               3400  Seq = 0 Ack = 1
1.482776  54393  ACK ->                    3400  Seq = 1 Ack = 1
1.482971  54393  PSH, ACK - Len: 19 ->     3400  Seq = 1 Ack = 1
1.483146  54393  <- ACK                    3400  Seq = 1 Ack = 20
1.484073  54393  <- PSH, ACK - Len: 17     3400  Seq = 1 Ack = 20
1.484173  54393  <- FIN, ACK               3400  Seq = 18 Ack = 20
1.485141  54393  ACK ->                    3400  Seq = 20 Ack = 19
1.541163  54393  FIN, ACK ->               3400  Seq = 20 Ack = 19
1.541423  54393  <- ACK                    3400  Seq = 19 Ack = 21
```

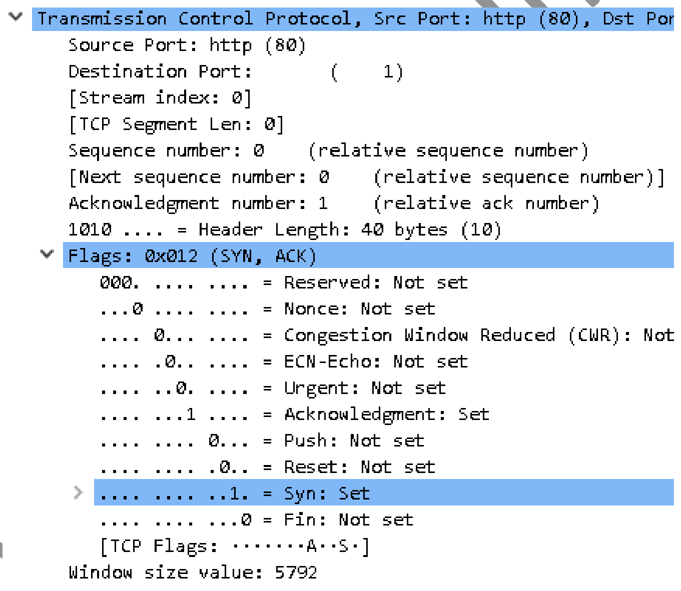
Here each arrow line represents each packet and It will shows the direction of communication .IP and port also will be visible in the flow graph. Here you can click the each arrow line so that it will high light the corresponding packets in Wireshark.

Expand the first packet and notice the SYN flag ,which will be set to 1.



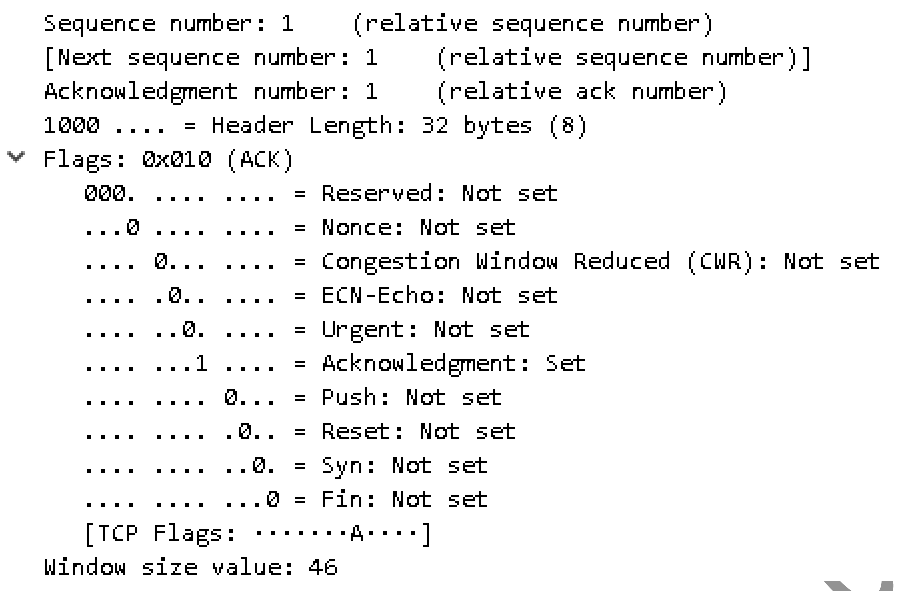
```
Transmission Control Protocol, Src Port:          (    1), Dst Port: http (80), S
    Source Port:         (      )
    Destination Port: http (80)
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 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·]
```

SYN packet will be used for establishing the connection ,so it will be the first TCP packet send by the client to a server. If we are checking the second packet which will be coming from server to client we can see both SYN and ACK bits are set.



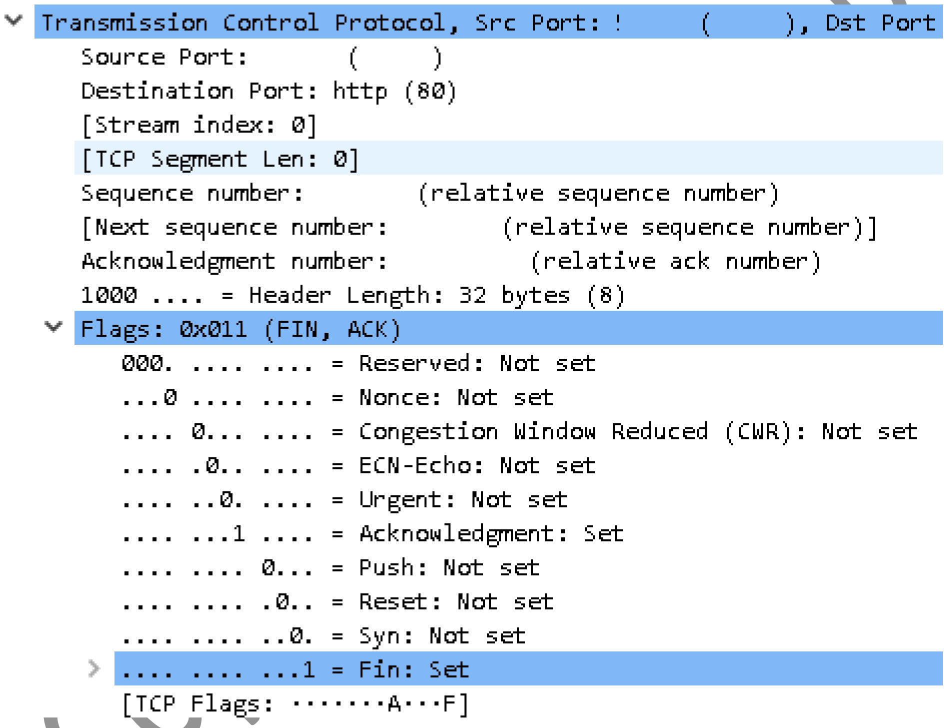
```
Transmission Control Protocol, Src Port: http (80), Dst Por
    Source Port: http (80)
    Destination Port:       (    1)
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    1010 .... = Header Length: 40 bytes (10)
  Flags: 0x012 (SYN, ACK)
       000. .... .... = Reserved: Not set
       ...0 .... .... = Nonce: Not set
       .... 0... .... = Congestion Window Reduced (CWR): Not
       .... .0.. .... = ECN-Echo: Not set
       .... ..0. .... = Urgent: Not set
       .... ...1 .... = Acknowledgment: Set
       .... .... 0... = Push: Not set
       .... .... .0.. = Reset: Not set
       .... .... ..1. = Syn: Set
       .... .... ...0 = Fin: Not set
       [TCP Flags: ·······A··S·]
    Window size value: 5792
```

Here ACK flag represents the acknowledge for the client connection request and SYN flag represents that server also wants to establish a connection with client.So as 3rd packet client wants to send a packet with ACK bit set as shown below,



```
      Sequence number: 1    (relative sequence number)
      [Next sequence number: 1    (relative sequence number)]
      Acknowledgment number: 1    (relative ack number)
      1000 .... = Header Length: 32 bytes (8)
    ⌄ Flags: 0x010 (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
         .... .... .0.. = Reset: Not set
         .... .... ..0. = Syn: Not set
         .... .... ...0 = Fin: Not set
         [TCP Flags: ·······A····]
      Window size value: 46
```

For closing the connection It will be using the FIN flag bit as shown below.



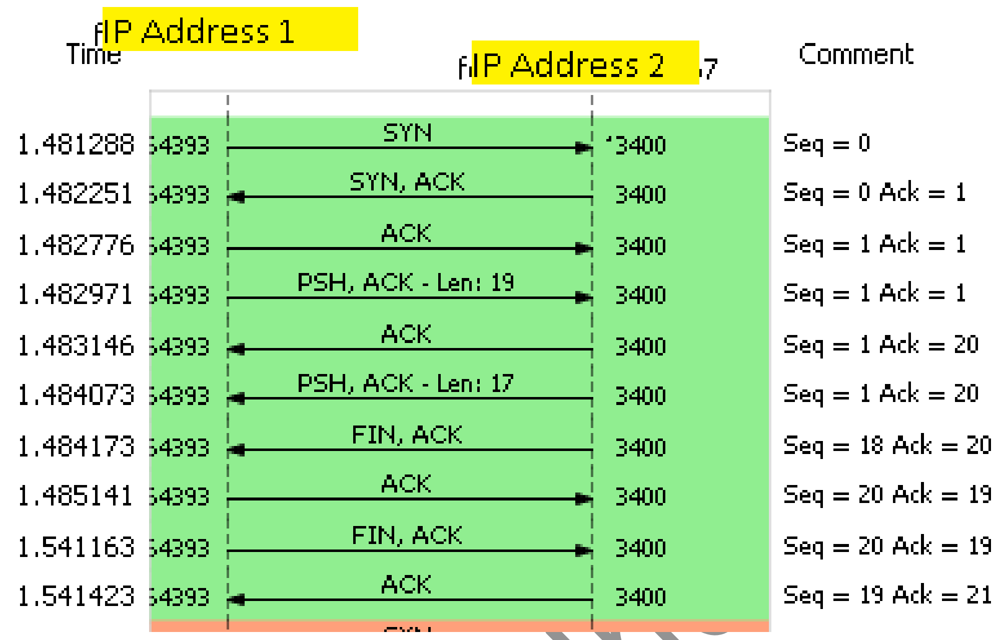
```
⌄ Transmission Control Protocol, Src Port: !      (      ), Dst Port
     Source Port:        (      )
     Destination Port: http (80)
     [Stream index: 0]
     [TCP Segment Len: 0]
     Sequence number:        (relative sequence number)
     [Next sequence number:        (relative sequence number)]
     Acknowledgment number:        (relative ack number)
     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
        .... ...1 .... = Acknowledgment: Set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
      › .... .... ...1 = Fin: Set
        [TCP Flags: ·······A···F]
```

<u>Understanding the Flow Graph</u>

Before jumping in to the flow graph we have to understand two more terms .

- ➢ Sequence number
- ➢ Acknowledge number

For understanding how much data has been transferred ,both sides are maintaining two 32 bit values called sequence number and acknowledge number. This sequence number will be included in each transmit packet and other side will be acknowledged as acknowledge number to inform the transmitter that all the data sent was received properly. Usually the initial sequence number will be random ,but wire shark will show the relative numbers (0 for initial packet) ,so it will be easy for analysis.

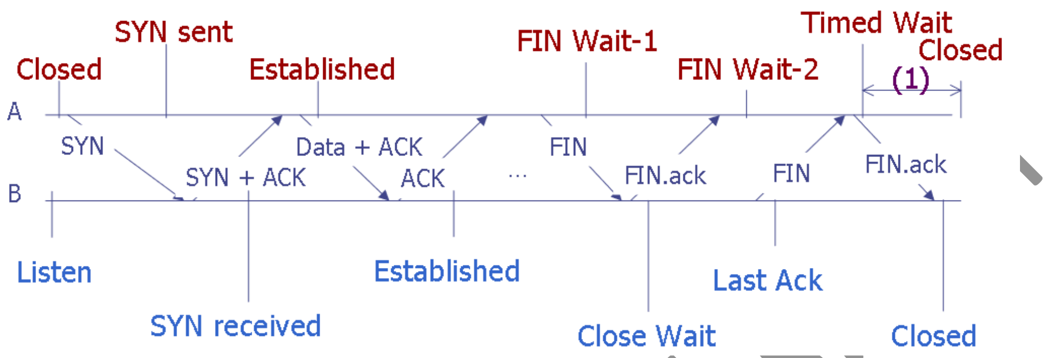


Time	IP Address 1		IP Address 2	Comment
1.481288	54393	SYN →	3400	Seq = 0
1.482251	54393	← SYN, ACK	3400	Seq = 0 Ack = 1
1.482776	54393	ACK →	3400	Seq = 1 Ack = 1
1.482971	54393	PSH, ACK - Len: 19 →	3400	Seq = 1 Ack = 1
1.483146	54393	← ACK	3400	Seq = 1 Ack = 20
1.484073	54393	← PSH, ACK - Len: 17	3400	Seq = 1 Ack = 20
1.484173	54393	← FIN, ACK	3400	Seq = 18 Ack = 20
1.485141	54393	ACK →	3400	Seq = 20 Ack = 19
1.541163	54393	FIN, ACK →	3400	Seq = 20 Ack = 19
1.541423	54393	← ACK	3400	Seq = 19 Ack = 21

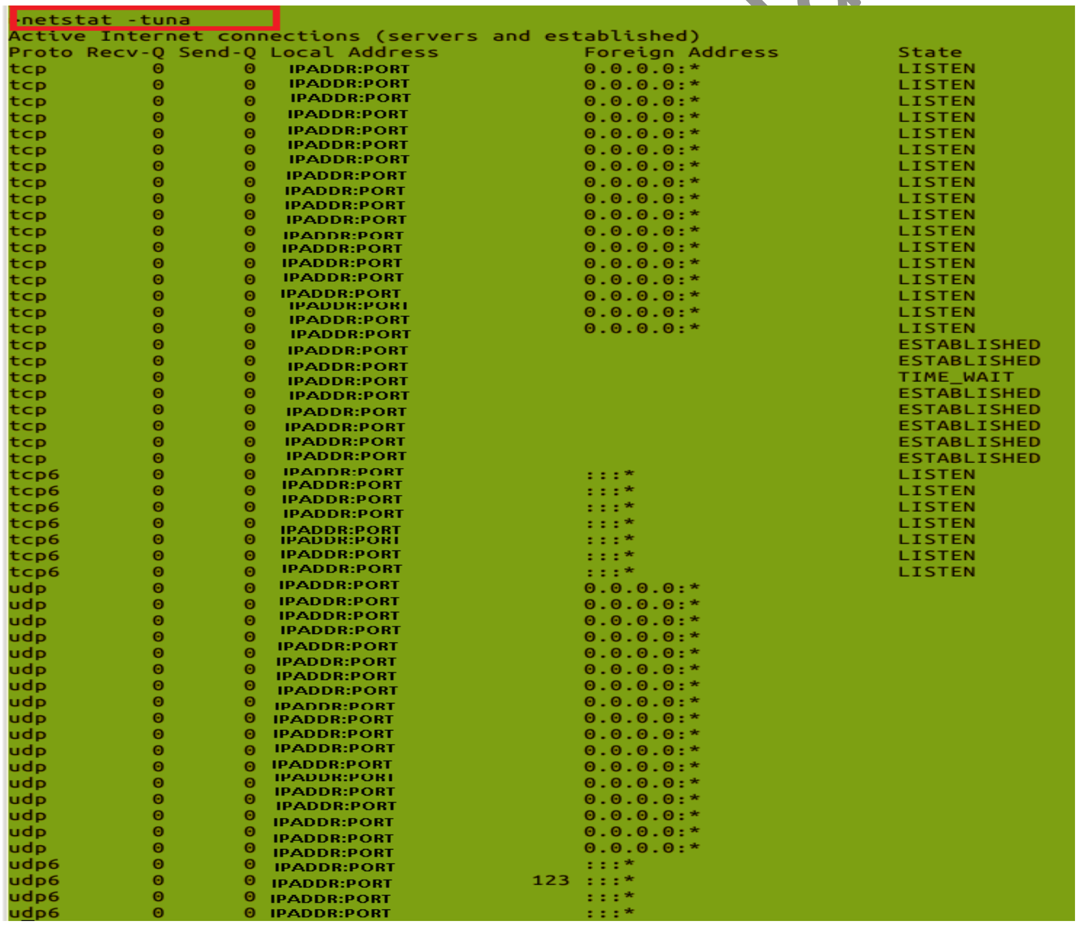
Now consider the flow graph mentioned above and analyze each packets as mentioned below

1. First packet is a SYN packet from the client for the connection , since it is the first packet both sequence number and acknowledge number (relative) will be 0.
2. Server will send the Second packet SYN and ACK with acknowledge number one to inform the client that server received the SYN packet. Since This is the first packet from the server sequence number will be zero.
3. Now Client will acknowledge the connection with 3rd ACK packet .Here ACK will be 1 as mentioned in the case of packet 2 .Sequence number also will be 1 (Incremented).At this point, the sequence number for both hosts is 1. This initial increment of 1 on both hosts' sequence numbers occurs during the establishment of all TCP sessions.
4. Packet 4 will be carrying the actual payload ,length is mentioned as 19.Since there is no data transfer has been happened yet the sequence number (nothing transmitted) and the acknowledge number (No data received) will remain the same (1,1).
5. Server will acknowledge the Packet with the same sequence number and 20 as the acknowledge number .(Saying how many data has been received.)
6. This packet marks the beginning of the server's response. Its sequence number is still 1, since none of its packets prior to this one have carried a payload. This packet carries a payload of 17 bytes.
7. Packet 7 server is sending the FIN for closing the connection because there is no remaining data is to be send. Here Sequence number will be increased from 1 to 18 (payload length of previous transmit was 17). ACK will be remain the same.
8. Here Client is acknowledging with sequence number 20 (payload length of previous transmit was 19) and acknowledging the server 19 bytes .

9. Packet 9 : Client is also sending the FIN packet for closing the connection .Slice there is no payload the sequence and acknowledge number will remain the same.
10. After receiving the FIN server will send the last packet in that communication with ACK flag set.
    Here acknowledge number will be incremented by 1 by the arrival of FIN from client and sequence number in the server side also incremented by 1.
    At this point, both hosts have terminated the session and can release the resources if any.



Netstat is a powerful tool which will helps to monitor the network status. netstat command with –t will list tcp and with –u will list udp ports



```
-netstat -tuna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT            0.0.0.0:*               LISTEN
tcp        0      0  IPADDR:PORT                                    ESTABLISHED
tcp        0      0  IPADDR:PORT                                    ESTABLISHED
tcp        0      0  IPADDR:PORT                                    TIME_WAIT
tcp        0      0  IPADDR:PORT                                    ESTABLISHED
tcp        0      0  IPADDR:PORT                                    ESTABLISHED
tcp        0      0  IPADDR:PORT                                    ESTABLISHED
tcp        0      0  IPADDR:PORT                                    ESTABLISHED
tcp        0      0  IPADDR:PORT                                    ESTABLISHED
tcp6       0      0  IPADDR:PORT            :::*                    LISTEN
tcp6       0      0  IPADDR:PORT            :::*                    LISTEN
tcp6       0      0  IPADDR:PORT            :::*                    LISTEN
tcp6       0      0  IPADDR:PORT            :::*                    LISTEN
tcp6       0      0  IPADDR:PORT            :::*                    LISTEN
tcp6       0      0  IPADDR:PORT            :::*                    LISTEN
tcp6       0      0  IPADDR:PORT            :::*                    LISTEN
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp        0      0  IPADDR:PORT            0.0.0.0:*
udp6       0      0  IPADDR:PORT            :::*
udp6       0      0  IPADDR:PORT        123 :::*
udp6       0      0  IPADDR:PORT            :::*
udp6       0      0  IPADDR:PORT            :::*
```

Session Layer ,Presentation Layer and Application Layer

*All these layers are integrated as a single layer in TCP/IP model as "Application Layer". These are also known as **Upper Layers** or **Software Layers**.* Session layer is responsible for establishment of connection, maintenance of sessions, authentication and also ensures security. Presentation layer is also called the **Translation layer**. The data from the application layer is extracted here and manipulated as per the required format to transmit over the network. Main functionalities include compression encryption decryption , translation(ASCII - EBCDIC) and compression and decompression. Finally application layer is the top layer , which consist of network application running like ,firefox,skype,mailbox etc. in most of the cases we will be following TCP/IP model than OSI model. These will be enough for getting started with automotive Ethernet . Come let's start ….

Click Below to follow us ,and stay updated.