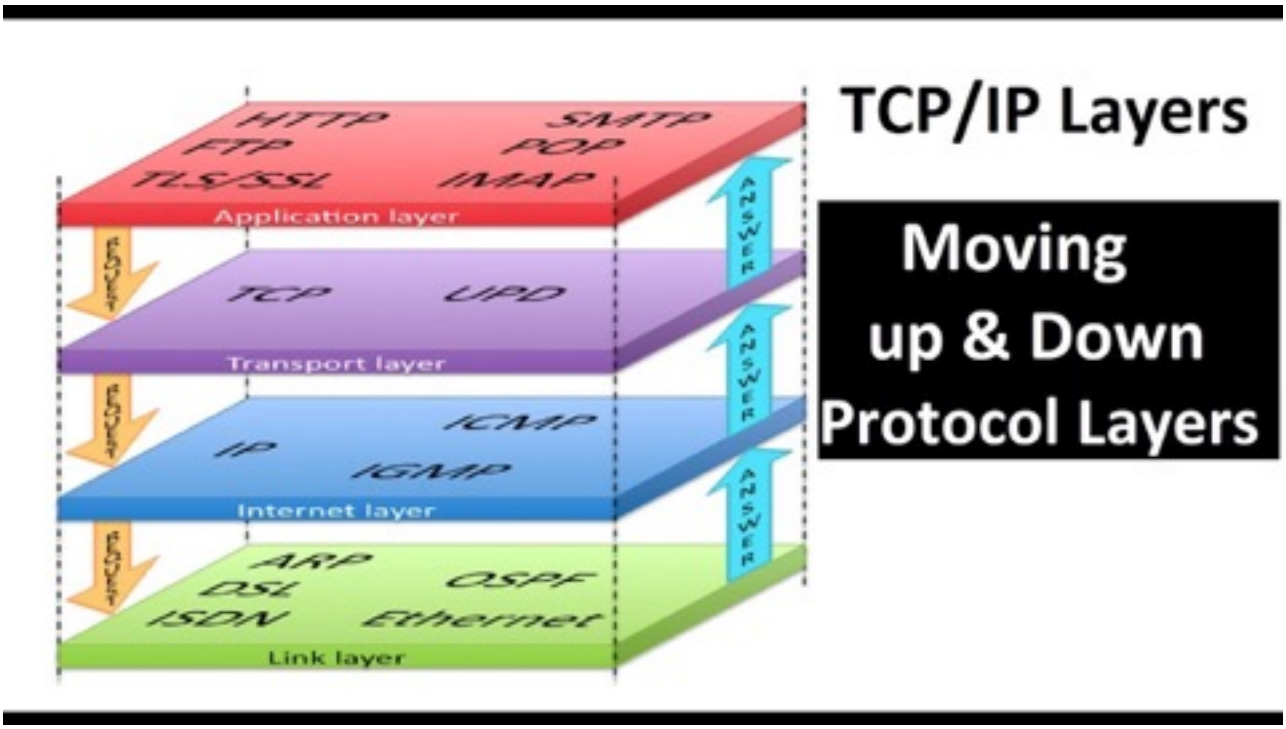
Networks Project

Encapsulation/Decapsulation of TCP,UDP,IP,Ethernet Packets

A Kamil Khan (P191311) routetokamil@gmail.com (9845104104)- 29 June 2020



Introduction

This Project is implemented as part of 2nd Semester Networks Lab Course for M.Sc Computer Science in Department of Computer Science Central University of Tamil Nadu Tiruvarur. It is developed to understand the process of encapsulating ethernet packets, IP packets, TCP packets and UDP packets for the given payload. Decapsulation of the above said packets also implemented. This implementation would give a better idea about various packet formats and header formats to the user. This project is implemented using C as Programming Language to get better hold of Void Pointers and Dynamic Memory Allocation.

The source code is placed here *https://github.com/donkhan/msc/tree/master/Networks/Project*

Features Implemented

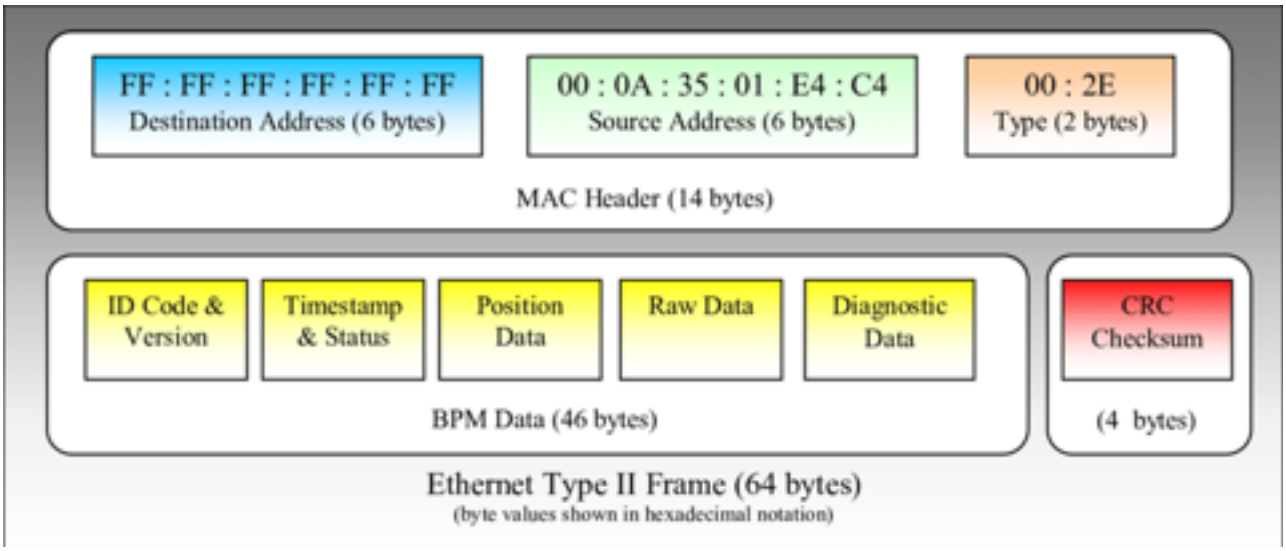
Menu to encapsulate/decapsulate packets
Encapsulation of Ethernet Packet
Decapsulation of Ethernet Packet
Encapsulation of IP Packet
Decapsulation of IP Packet
Encapsulation of TCP Packet
Decapsulation of TCP Packet
Encapsulation of UDP Packet
Decapsulation of UDP Packet
Options to save the packets in a binary file
Read packets from the binary file and decapsulate

Encapsulation and Decapsulation of Ethernet Packet

Ethernet Packet:

In computer networking, an Ethernet frame is a data link layer protocol data unit and uses the underlying Ethernet physical layer transport mechanisms. In other words, a data unit on an Ethernet link transports an Ethernet frame as its payload.[1]

An Ethernet frame is preceded by a preamble and start frame delimiter (SFD), which are both part of the Ethernet packet at the physical layer. Each Ethernet frame starts with an Ethernet header, which contains destination and source MAC addresses as its first two fields. The middle section of the frame is payload data including any headers for other protocols (for example, Internet Protocol) carried in the frame. The frame ends with a frame check sequence (FCS), which is a 32-bit cyclic redundancy check used to detect any in-transit corruption of data.



Ethernet Header:

Destination Address – This is 6-Byte field which contains the MAC address of machine for which data is destined.
Source Address – This is a 6-Byte field which contains the MAC address of source machine. As Source Address is always an individual address (Unicast), the least significant bit of first byte is always 0.

Other Fields:

Length – Length is a 2-Byte field, which indicates the length of entire Ethernet frame. This 16-bit field can hold the length value between 0 to 65534, but length cannot be larger than 1500 because of some own limitations of Ethernet.

Data – This is the place where actual data is inserted, also known as Payload. Both IP header and data will be inserted here if Internet Protocol is used over Ethernet. The maximum data present may be as long as 1500 Bytes. In case data length is less than minimum length i.e. 46 bytes, then padding 0's is added to meet the minimum possible length.
Cyclic Redundancy Check (CRC) – CRC is 4 Byte field. This field contains a 32-bits hash code of data, which is generated over the Destination Address, Source Address, Length, and Data field. If the checksum computed by destination is not the same as sent checksum value, data received is corrupted.

Implementation:

```
struct eth_header {
	char *src_mac;
	char *dst_mac;

};

struct eth{
	struct eth_header* header;
	struct ip* data;
};
```

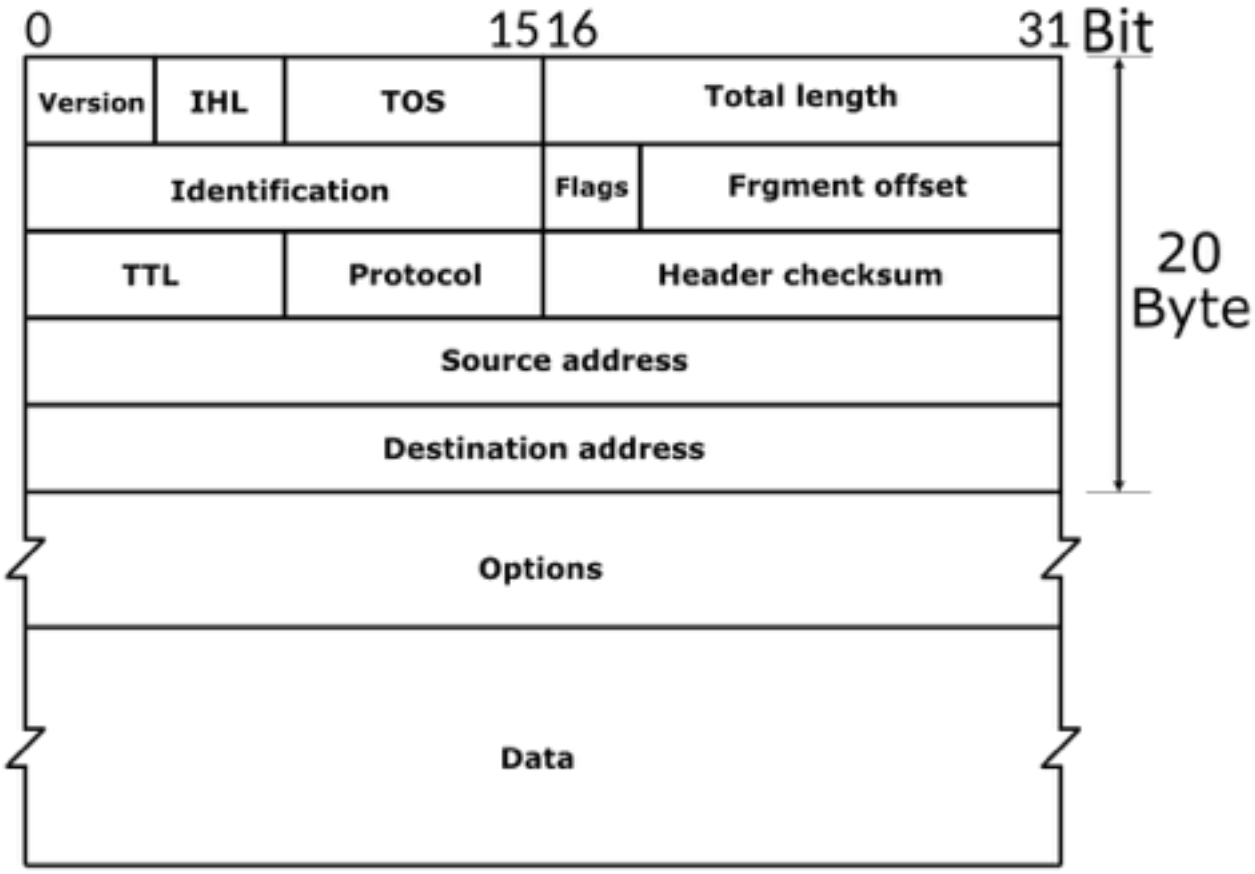
Note: CRC check is not implemented.

Encapsulation and Decapsulation of IP Packet

IP Packet:

Internet Protocol version 4 (IPv4) is the fourth version of the Internet Protocol (IP). It is one of the core protocols of standards-based internetworking methods in the Internet and other packet-switched networks. IPv4 was the first version deployed for production in the ARPANET in 1983. It still routes most Internet traffic today,[1] despite the ongoing deployment of a successor protocol, IPv6. IPv4 is described in IETF publication RFC 791 (September 1981), replacing an earlier definition (RFC 760, January 1980).

IPv4 uses a 32-bit address space which provides 4,294,967,296 (232) unique addresses, but large blocks are reserved for special networking methods.



IP Header:

An IP header is a prefix to an IP packet that contains information about the IP version, length of the packet, source and destination IP addresses, etc. It consists of the following fields:

Here is a description of each field:

Version – the version of the IP protocol. For IPv4, this field has a value of 4. Header length – the length of the header in 32-bit words. The minumum value is 20 bytes, and the maximum value is 60 bytes.

Priority and Type of Service – specifies how the datagram should be handled. The first 3 bits are the priority bits.

Total length – the length of the entire packet (header + data). The minimum length is 20 bytes, and the maximum is 65,535 bytes.

Identification – used to differentiate fragmented packets from different datagrams.

Flags – used to control or identify fragments.

Fragmented offset – used for fragmentation and reassembly if the packet is too large to put in a frame.

Time to live – limits a datagram's lifetime. If the packet doesn't get to its destination before the TTL expires, it is discarded.

Protocol – defines the protocol used in the data portion of the IP datagram. For example, TCP is represented by the number 6 and UDP by 17.

Header checksum – used for error-checking of the header. If a packet arrives at a router and the router calculates a different checksum than the one specified in this field, the packet will be discarded.

Source IP address – the IP address of the host that sent the packet.

Destination IP address – the IP address of the host that should receive the packet.

Other Fields:

Options – used for network testing, debugging, security, and more. This field is usually empty.

Implementation:

```
struct ip_header {
    unsigned int version :4;
    unsigned int ihl :4;
    unsigned int dscp :6;
    unsigned int ecn :2;
    unsigned int total_length :16;
    unsigned int identification :16;
```

```
        unsigned int flags :2;
        unsigned int fragment_offset :14;

        unsigned int ttl :8;
        unsigned int protocol :8;
        unsigned int header_checksum :16;

        char src[15];
        char dst[15];
};

struct ip{
        struct ip_header* header;
        void* data;
        int option;
};
```

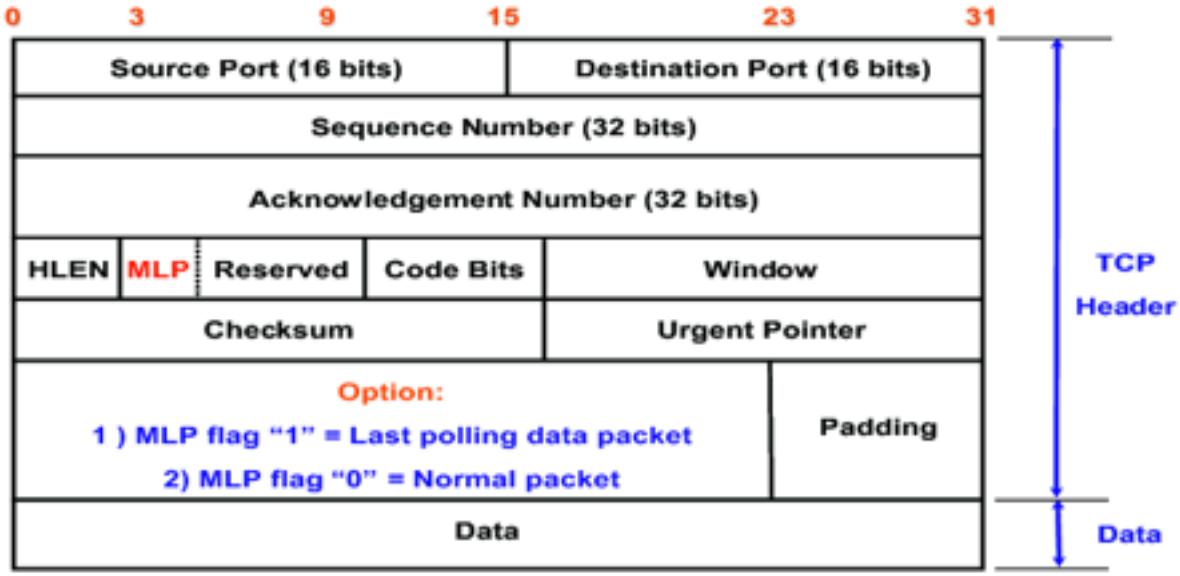
Note: Header CRC check is not implemented.

Encapsulation and Decapsulation of TCP Packet

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating via an IP network. Major internet applications such as the World Wide Web, email, remote administration, and file transfer rely on TCP, which is part of the Transport Layer of the TCP/IP suite. SSL/TLS often runs on top of TCP.

TCP is connection-oriented, and a connection between client and server is established before data can be sent. The server must be listening (passive open) for connection requests from clients before a connection is established. Three-way handshake (active open), retransmission, and error-detection adds to reliability but lengthens latency. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that prioritizes time over reliability. TCP employs network congestion avoidance. However, there are vulnerabilities to TCP including denial of service, connection hijacking, TCP veto, and reset attack. For network security, monitoring, and debugging, TCP traffic can be intercepted and logged with a packet sniffer.

Though TCP is a complex protocol, its basic operation has not changed significantly since its first specification. TCP is still dominantly used for the web, i.e. for the HTTP protocol, and later HTTP/2, while not used by latest standard HTTP/3.



Source and Destination Port Number
Identification of the sending and receiving application. Along with the source and destination IP addresses in the IP - header identify the connection as a socket.

Sequence Number
The sequence number of the first data byte in this segment. If the SYN bit is set, the sequence number is the initial sequence number and the first data byte is initial sequence number + 1.

Acknowledgement Number
If the ACK bit is set, this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

Hlen
The number of 32-bit words in the TCP header. This indicates where the data begins. The length of the TCP header is always a multiple of 32 bits.

Flags
There are six flags in the TCP header. One or more can be turned on at the same time.

URG The URGENT POINTER field contains valid data
ACK The ackowledgement number is valid

PSH The receiver should pass this data to the application as soon as possible
RST Reset the connection
SYN Synchronize sequence numbers to initiate a connection.
FIN The sender is finished sending data.

Window

This is the number of bytes, starting with the one specified by the acknowledgment number field, that the receiver is willing to accept. This is a 16-bit field, limiting the window to 65535 bytes.

Checksum

This covers both the header and the data. It is calculated by prepending a pseudo-header to the TCP segment, this consists of three 32 bit words which contain the source and destination IP addresses, a byte set to 0, a byte set to 6 (the protocol number for TCP in an IP datagram header) and the segment length (in words). The 16-bit one's complement sum of the header is calculated (i.e., the entire pseudo-header is considered a sequence of 16-bit words). The 16-bit one's complement of this sum is stored in the checksum field. This is a mandatory field that must be calculated and stored by the sender, and then verified by the receiver.

Urgent Pointer

The urgent pointer is valid only if the URG flag is set. This pointer is a positive offset that must be added to the sequence number field of the segment to yield the sequence number of the last byte of urgent data. TCP's urgent mode is a way for the sender to transmit emergency data to the other end. This feature is rarely used.

Implementation:

```
struct tcp{
	struct tcp_header* header;
	char* data;
};


struct tcp_header {
	unsigned int src_port :16;
	unsigned int dst_port :16;
	unsigned int seq_no :32;
	unsigned int ack_no :32;
	unsigned int hlen :4;
	unsigned int reserved :6;
	unsigned int urg :1;
	unsigned int ack :1;
	unsigned int psh :1;
```

```
        unsigned int rst :1;
        unsigned int syn :1;
        unsigned int fin :1;
        unsigned int window_size :16;
        unsigned int tcp_checksum :16;
        unsigned int urgent_pointer :16;
        //The number of 32 bit words in the TCP Header.
        unsigned int *options;
};
```

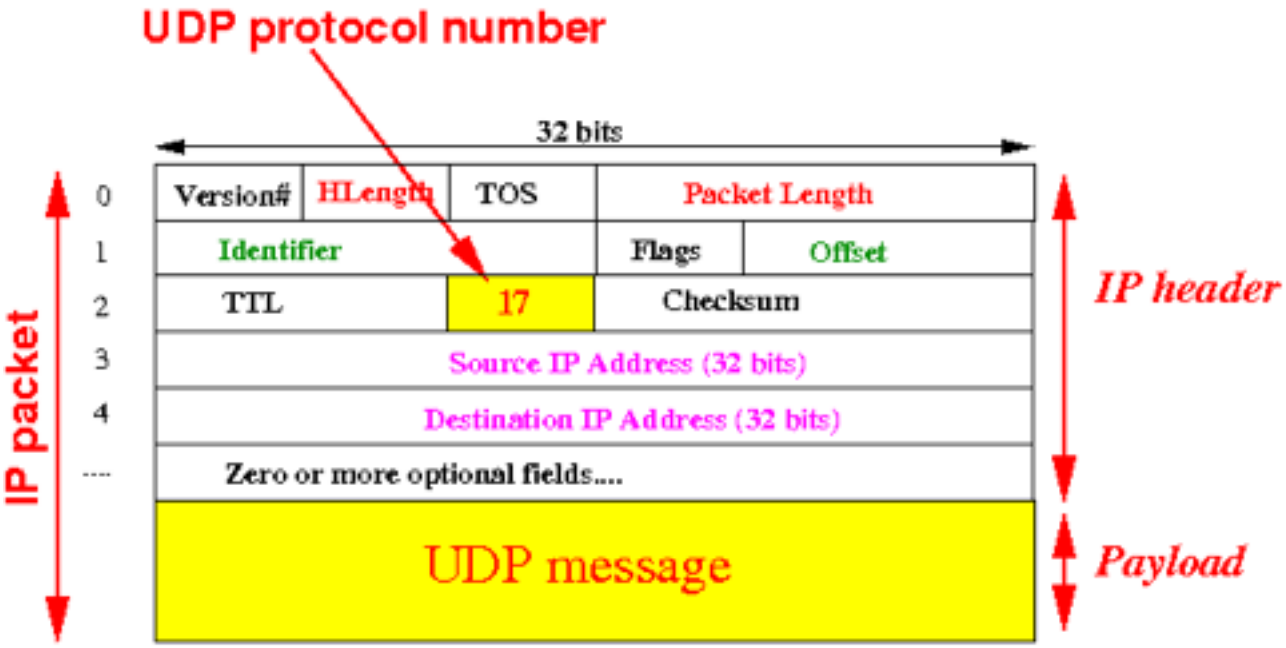
Note: Header CRC check is not implemented.

Encapsulation and Decapsulation of UDP Packet

In computer networking, the User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network. Prior communications are not required in order to set up communication channels or data paths.

UDP uses a simple connectionless communication model with a minimum of protocol mechanisms. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network; there is no guarantee of delivery, ordering, or duplicate protection. If error-correction facilities are needed at the network interface level, an application may use Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

UDP is suitable for purposes where error checking and correction are either not necessary or are performed in the application; UDP avoids the overhead of such processing in the protocol stack. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for packets delayed due to retransmission, which may not be an option in a real-time system.[1]



Implementation:

```
struct udp_header {
      unsigned int src_port : 16;
    unsigned int dst_port : 16;
    unsigned int length : 16;
    unsigned int checksum : 16;
};

struct udp{
      struct udp_header* header;
      char* data;
};
```

Note: Header CRC check is not implemented.

Extra Features

1. Save Option is provided to each and every encapsulated packet into a binary format
2. Read the binary content, decapsulate the packet for every layer.