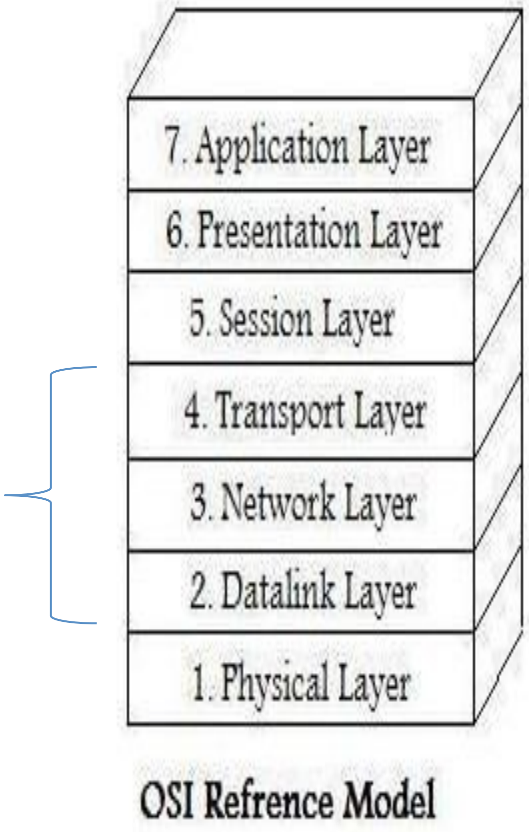
TOS Network Logical Layer Implementation(NLL)

Where is NLL?

- NLL stretches across 3 layers in the OSI network model
- Data Link layer (ARP)
- Network Layer (IP)
- Transportation Layer (UDP)

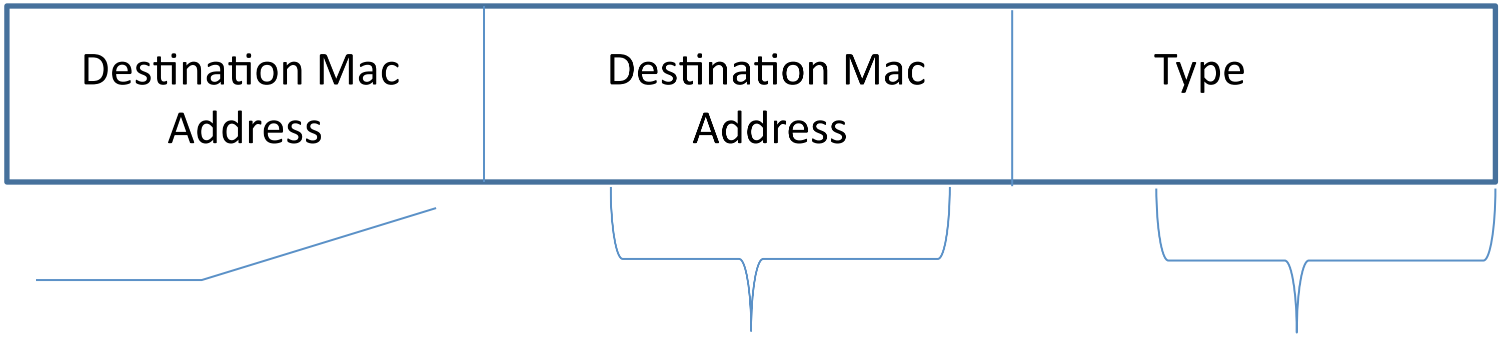


Major tasks done

- API’s interfacing lower physical layer
- API’s interfacing upper application layer
- Parsing of arp packets
- Parsing of UDP packets
- Construction of ARP packets
- Construction of UDP packets

Packet header structures

- Ethernet header



Destination Mac Address

Destination Mac Address

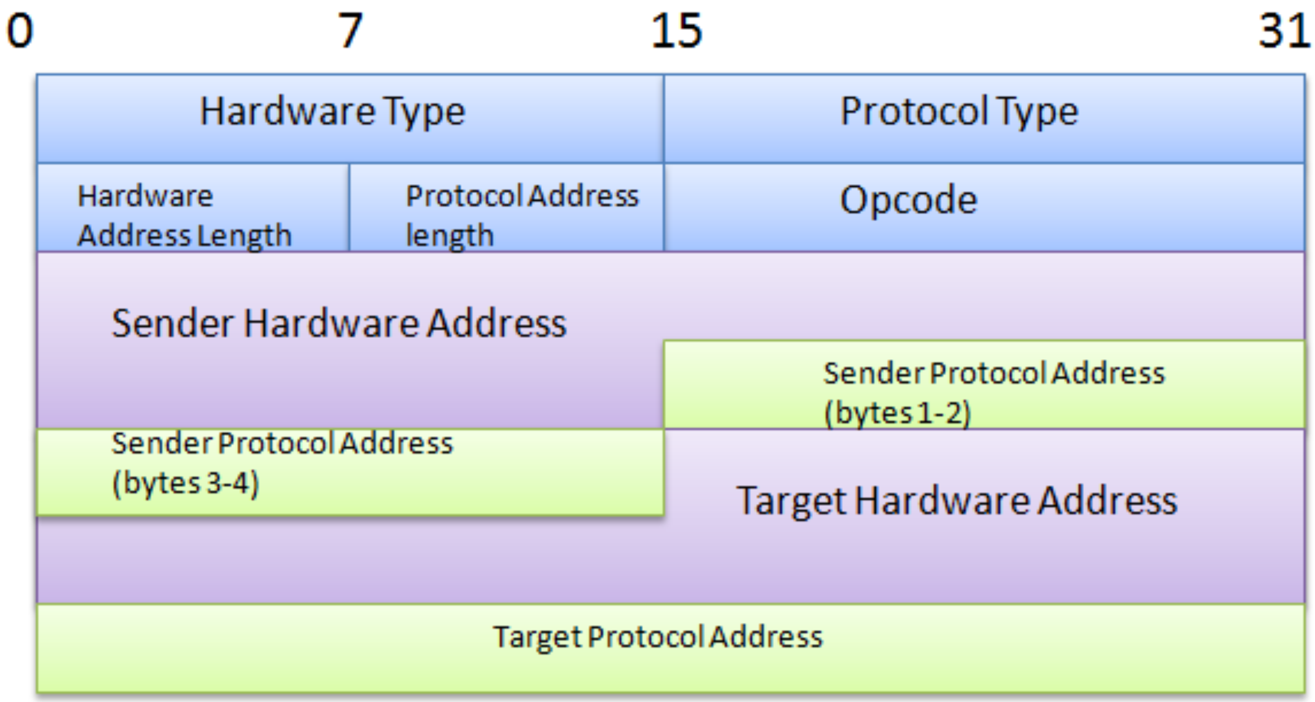
Type

- 6 bytes 6 bytes 2 bytes

Cont. ..

- ARP header:

ARP header



0 7 15 31

Hardware Type

Protocol Type

Hardware Address Length

Protocol Address length

Opcode

Sender Hardware Address

Sender Protocol Address (bytes 1-2)

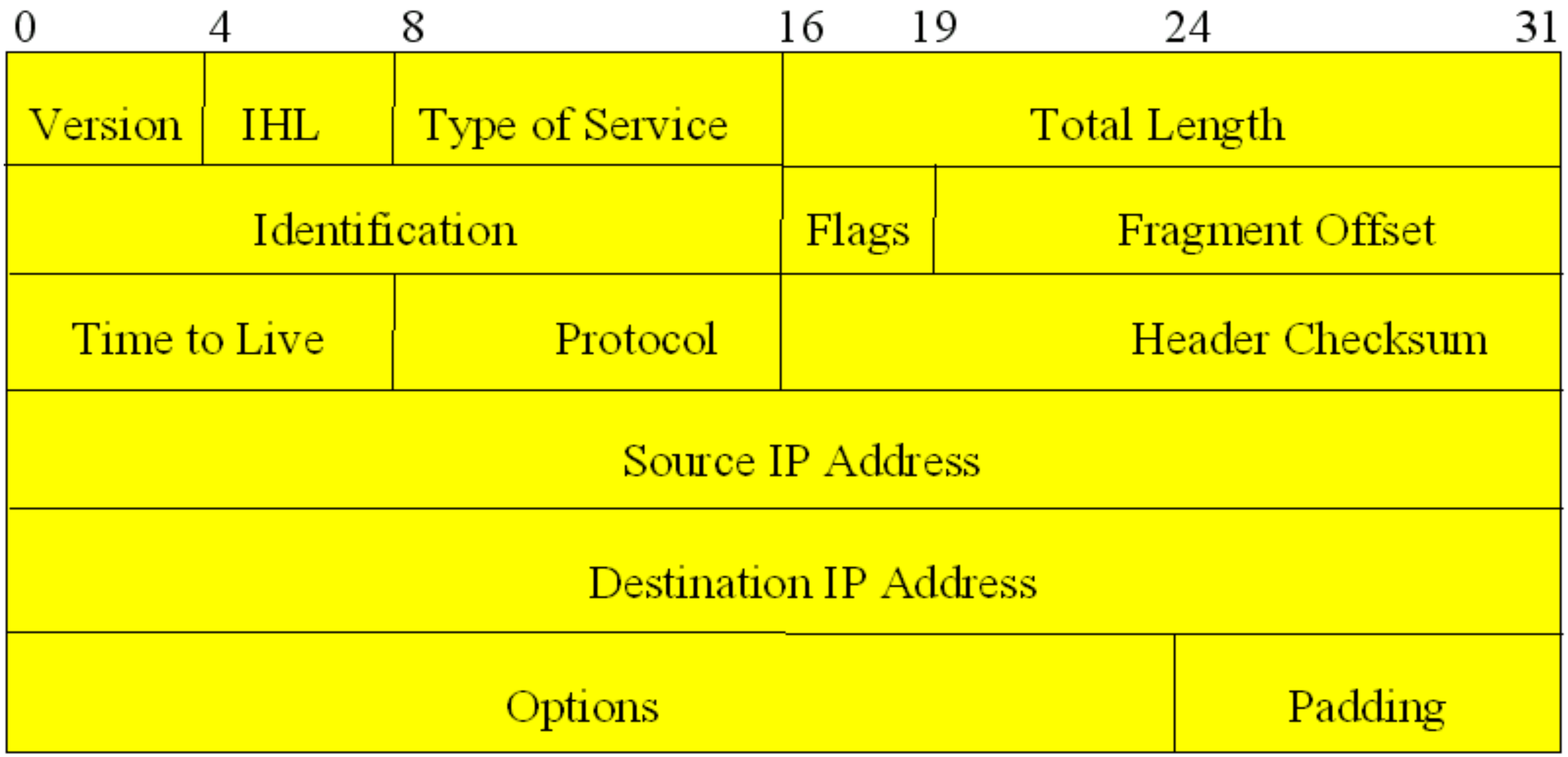
Sender Protocol Address (bytes 3-4)

Target Hardware Address

Target Protocol Address

Cont.

- IP header:



0 4 8 16 19 24 31

Version | IHL | Type of Service | Total Length

Identification | Flags | Fragment Offset

Time to Live | Protocol | Header Checksum

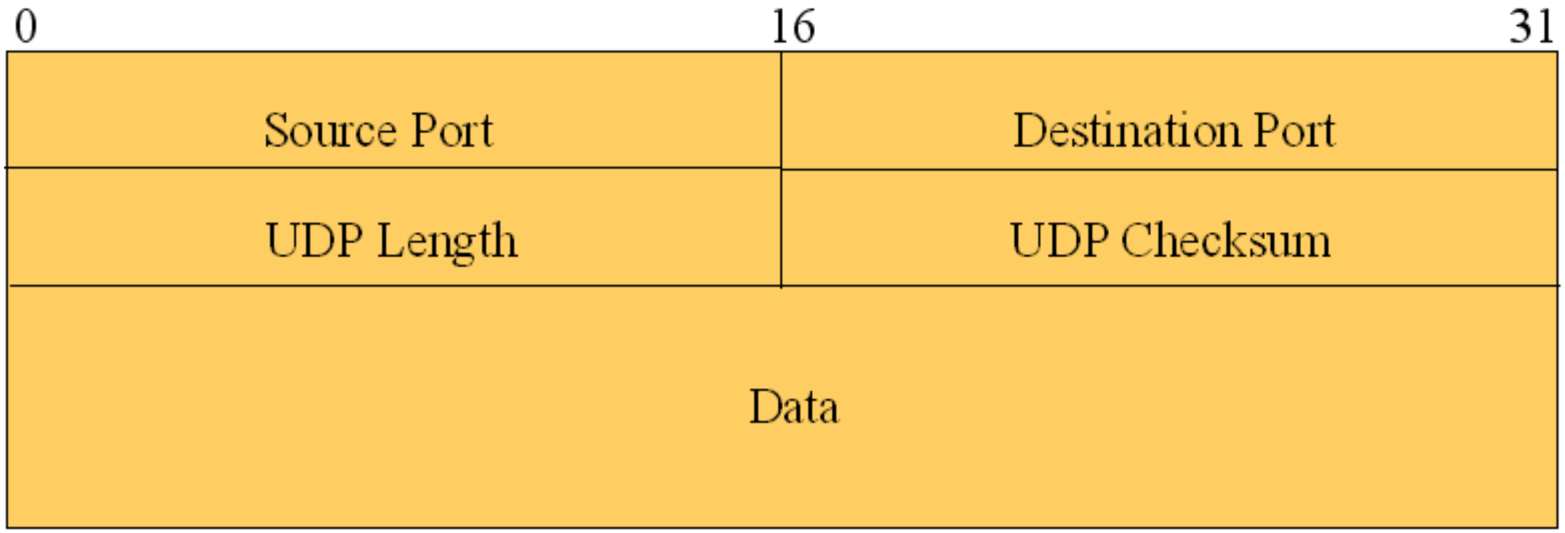
Source IP Address

Destination IP Address

Options | Padding

Cont.

- UDP Header:



0 16 31

Source Port	Destination Port
UDP Length	UDP Checksum
Data	

File organization

- Header file
  - nll.h
- C files
  - arp.c
  - ip.c
  - udp.c
  - eth.c
  - test_print.c

nll.h

- Structure definitions for all the headers
- Function prototype definition
- Inline function
  - tos_ntohs()
  - tos_htons()
- Used for byte order conversion

```
INLINE unsigned int ntohs_tos(unsigned short n){
   #if __BYTE_ORDER__==__LITTLE_ENDIAN__
         return (((n & 0xFF00) >> 8) | ((n & 0x00FF) << 8));
   #else
       return n;
  #endif
  }

 INLINE unsigned short htons_tos(unsigned short n){
    #if __BYTE_ORDER__==__LITTLE_ENDIAN__
          return (((n & 0xFF00) >> 8) | ((n & 0x00FF) << 8));
   #else
        return n;
    #endif
   }
```

arp.c

- Functions parsing arp request and reply
  - BOOL is_arp_request(void *buffer, u_int_t len, ARP *arp_pkt) {

    }
  - BOOL is_arp_reply(void *buffer, u_int_t len, ARP *arp_pkt) {

    }
- Function adding & updating cache entry
- void arp_add_cache(u_char_t *ip, u_char_t *mac) {
  - }
- Function translating ip address to mac address
  - BOOL arp_ip_to_mac(u_char_t *eth_addr, u_char_t *ip) {

    }
- Method constructing arp packet
  - u_int_t create_arp_packet(u_char_t *ip_to, u_char_t *eth_to, u_char_t *host_ip,   u_char_t *host_mac, u_int16_t arp_op, ARP *packet) {
  - }

Ip.c

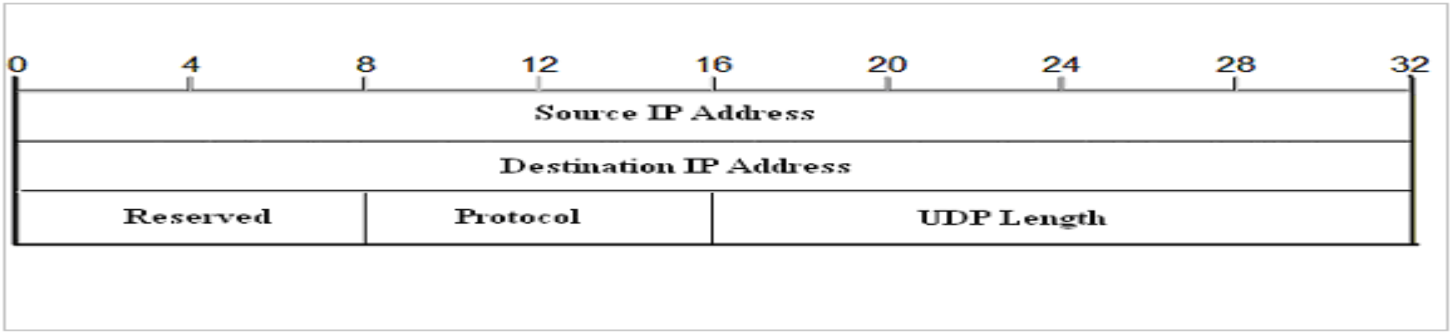
- Construction of ip packet
  - int create_ip_hr(u_char_t *src_ip,u_char_t *dst_ip,u_int_t payload_len,IP *packet) {
  - unsigned short packet_len = (sizeof(IP) + payload_len);
  - packet->version = IP_V4;
  - packet->hdr_len = sizeof(IP) / sizeof(int);
  - packet->tos = IP_TOS_MIN_DELAY;
  - packet->len = htons_tos(packet_len);
  - packet->id = htons_tos(0xFEED);
  - packet->offset = htons_tos(IP_FLAG_DF);
  - packet->ttl = IP_DEFAULT_TTL;
  - packet->protocol = IP_PROTO_UDP;
  - packet->checksum = 0;
  - memcpy_tos(packet->src, src_ip, IP_LEN);
  - memcpy_tos(packet->dst, dst_ip, IP_LEN);
  - packet->checksum = ip_checksum(packet);
  - return (int)packet_len;
  - }

- **Calculating ip header checksum**
  - Used ones' complement of the sum of the header's 16-bit words.
  - u_int16_t ip_checksum(IP *ip){
  - }

Udp.c

- Parsing of udp packets:
  - BOOL is_udp_packet(void *buffer,u_int_t len,UDP *packet){

    }
- Calculation of the udp checksum:
  - udp check sum calculated including a pseudo ip header
  - u_int16_t udp_checksum(UDP *udp,u_char_t *src_ip,u_char_t *dst_ip) {
  - }



Source IP Address		
Destination IP Address		
Reserved	Protocol	UDP Length

Main challenges

- C pointers
  - Hard time understanding c pointers and pointers to structures
- Dangers of bitwise operations
  - Bitwise operations very danger and
  - At times confusing
- Reading and understanding protocols

What I got

- Confidence working with c pointers
- Confidence reading & understanding RFC’s
- Understand what OS processes actually are.