Using Raw Sockets
with Internet Protocols

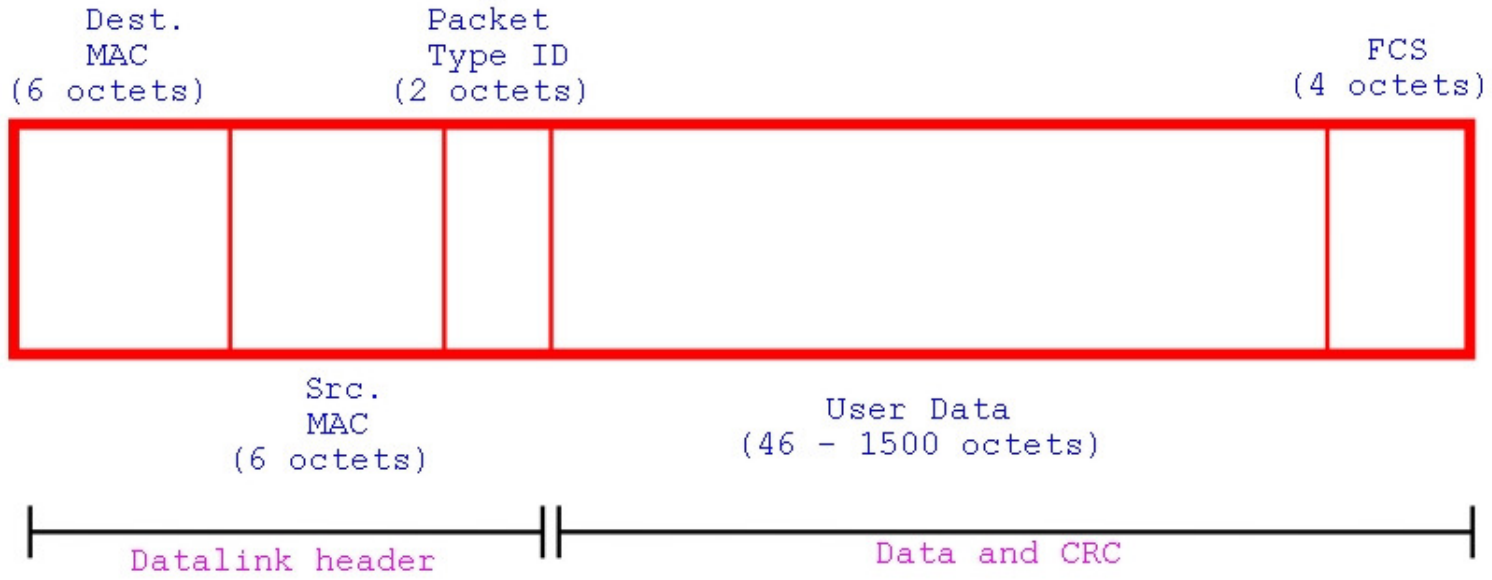
P. Bakowski

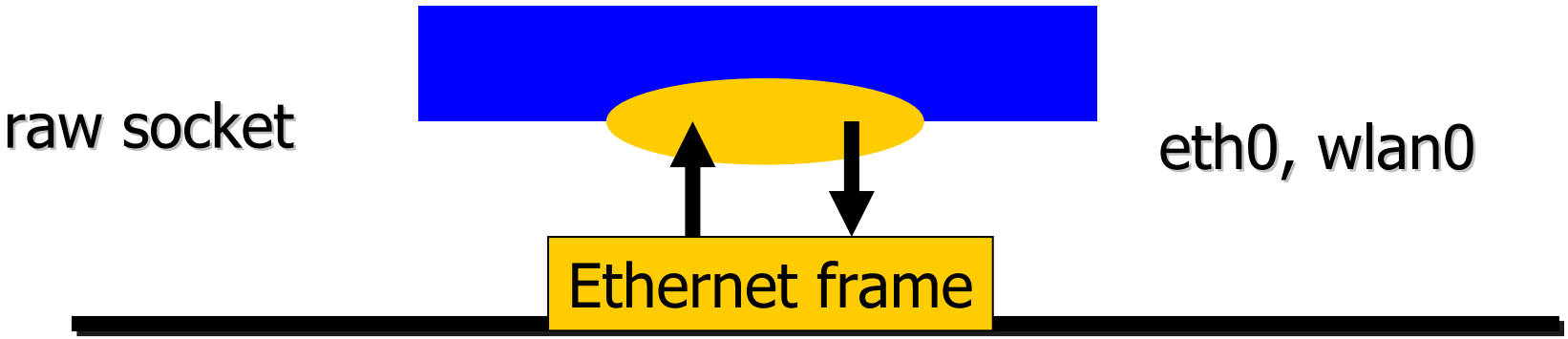
bako@ieee.org

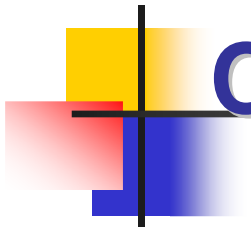
What is Raw Socket ?

Most socket application programming interfaces (APIs), especially those based on Berkeley sockets, support raw sockets.
Usually raw sockets receive packets inclusive of the header, as opposed to standard sockets which receive just the packet payload without headers.
When transmitting packets, the automatic addition of a header may be a configurable option of the socket.

What is Raw Socket ?





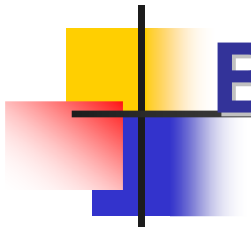


Creating raw socket

```
int create_rawsocket(int protocol_to_sniff)
{
int rawsock;
if((rawsock = socket(PF_PACKET, SOCK_RAW,
htons(protocol_to_sniff)))== -1)
        {
                perror("Error creating raw socket: ");
                exit(-1);
        }
        return rawsock;
}
```

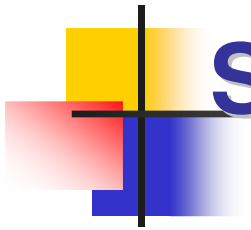
Binding raw socket

```
int bind_rawsocket(char *device, int rawsock, int protocol)
{
        struct sockaddr_ll sll;
        struct ifreq ifr;
        bzero(&sll, sizeof(struct sockaddr_ll));
        bzero(&ifr, sizeof(struct ifreq));
        /* First Get the Interface Index  */
        strncpy((char *)ifr.ifr_name, device, IFNAMSIZ);
        if((ioctl(rawsock, SIOCGIFINDEX, &ifr)) == -1)
        {
        printf("Error getting Interface index !\n"); exit(-1);
        }
```



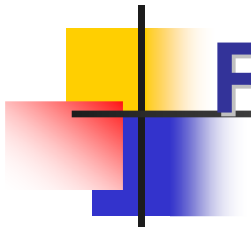
Binding raw socket

```
...
/* Bind our raw socket to this interface */
sll.sll_family = AF_PACKET;
sll.sll_ifindex = ifr.ifr_ifindex;
sll.sll_protocol = htons(protocol);
if((bind(rawsock, (struct sockaddr *)&sll, sizeof(sll)))== -1)
        {
                perror("Error binding raw socket to interface\n");
                exit(-1);
        }
        return 1;
}
```



Send raw packet on line

```
int send_rawpacket(int rawsock, unsigned char *pkt,
int pkt_len)
{
int sent= 0;
if((sent=write(rawsock, pkt, pkt_len)) != pkt_len)
{
printf("Could only send %d bytes of packet of length
%d\n", sent, pkt_len);
return 0;
}
return 1;
}
```

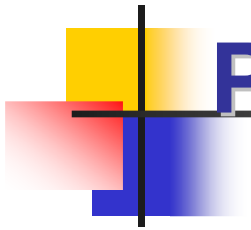


Receive raw packet from line

```
int recv_rawpacket(int rawsock, unsigned char *pkt,
int pkt_len)
{
int recv= 0;
recv = read(rawsock, pkt, pkt_len);
return recv;
}
```

Create Ethernet packet header

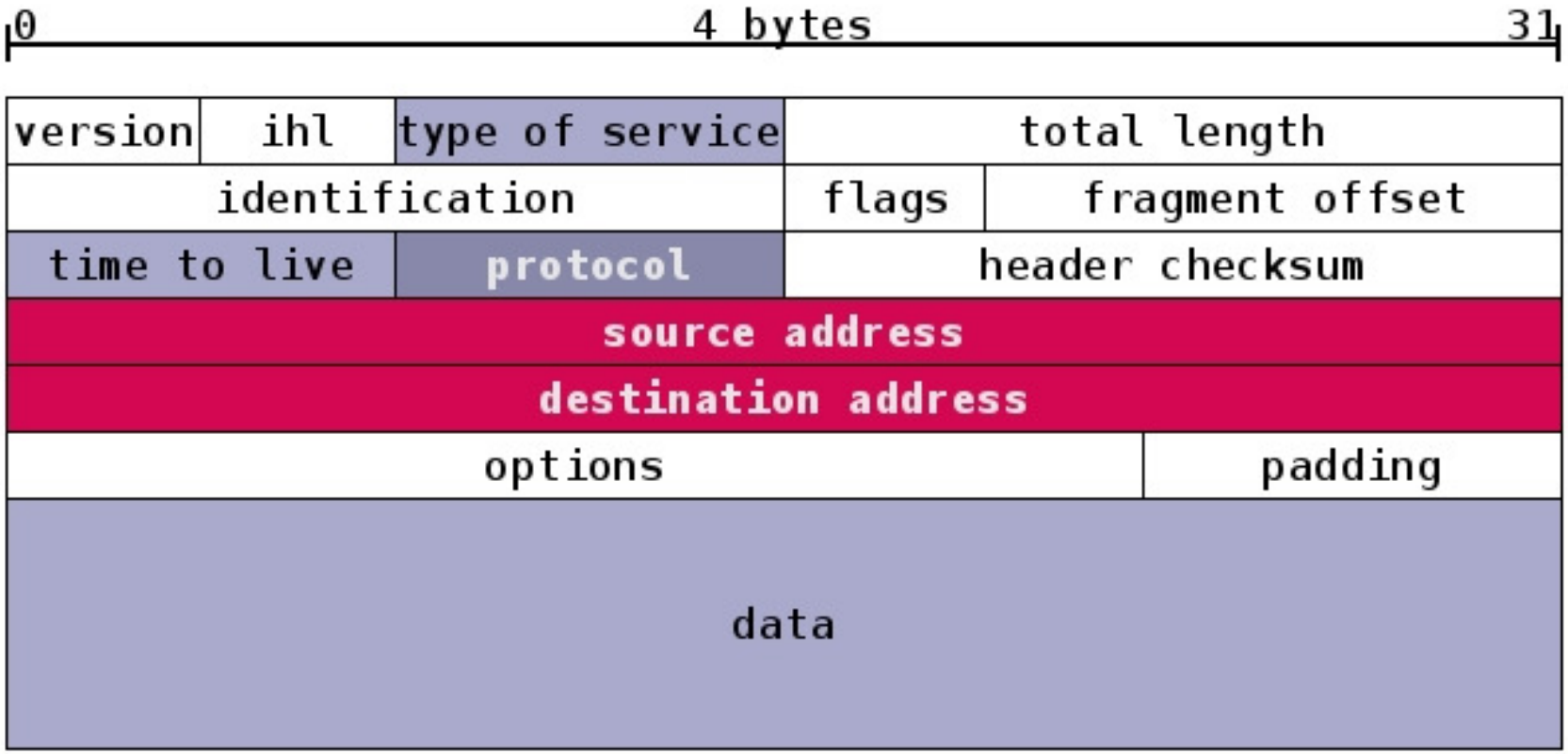
```
unsigned char *create_eth(char *src_mac, char *dst_mac,
unsigned short protocol)
{
unsigned char *ethbuf;  unsigned char abuf[6];
unsigned padding;  unsigned short type=htons(protocol);
ethbuf = (unsigned char *) malloc(14);
parse_mac(abuf,dst_mac);    // from ff:ff:ff:ff:ff:ff form
memcpy(ethbuf,abuf,6);
parse_mac(abuf,src_mac);
memcpy(ethbuf+6,abuf,6);
memcpy(ethbuf+12,(unsigned char *)&type,2);
return ethbuf;
}
```

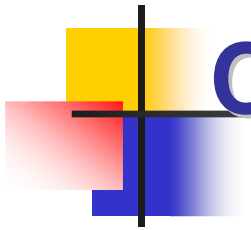


Print Ethernet packet header

```
int  print_ethhdr(unsigned char *eth_headstr)
{ unsigned char *ethhead; int j;
ethhead=eth_headstr;
printf("Ethernet header\ndestination address: ");
for(j=0;j<6;j++) printf("%02x:",*(ethhead+j));
printf("  source address: ");
for(j=6;j<12;j++) printf("%02x:",*(ethhead+j));
printf("   Ether protocol number: ");
for(j=12;j<14;j++) printf("%02x",*(ethhead+j));
printf("\nend of Ethernet header\n");
if (*(ethhead+12)==8 && *(ethhead+13)==0) return 1; // IP
if (*(ethhead+12)==8 && *(ethhead+13)==6) return 2; // ARP
return 0;
}
```

Create IP packet header



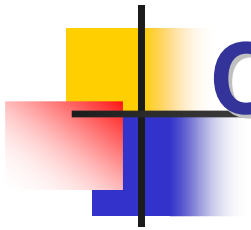


Create IP packet header

```
unsigned char *create_iphdr(
                    unsigned char verlen,
                    unsigned char tos,
                    unsigned short totlen,
                    unsigned short id,
                    unsigned short foffset,
                    unsigned char ttl,
                    unsigned char proto,
                    unsigned short checksum,
                    unsigned int sa,
                    unsigned int da)
```

Create IP packet header

```
struct ip_hdr *ip_header;
/*
unsigned char ip_version_and_header_length;
unsigned char ip_tos;      // type of service
unsigned short ip_len;     // total lengthether_dest_addr
unsigned short ip_id;      // identification number
unsigned short ip_frag_offset; // fragment offset and flags
unsigned char ip_ttl;                 // time to live
unsigned char ip_type;                // protocol type
unsigned short ip_checksum;           // checksum
unsigned int ip_src_addr;             // source IP address
unsigned int ip_dest_addr;            // destination IP address
*/
```



Create IP packet header

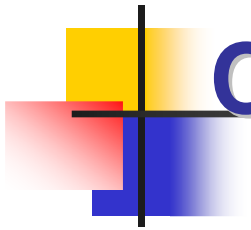
```
...
ip_header = (struct ip_hdr *)malloc(sizeof(struct ip_hdr));
ip_header->ip_version_and_header_length = verlen;
ip_header->ip_tos = tos; ip_header->ip_len = totlen;
ip_header->ip_id = id;ip_header->ip_frag_offset=foffset;
ip_header->ip_ttl = ttl; ip_header->ip_type = proto;
ip_header->ip_checksum = checksum;
ip_header->ip_src_addr=sa;ip_header->ip_dest_addr=da;
return ((unsigned char *)ip_header);
}
```

Print IP packet header

```
void  print_iphdr(unsigned char *ip_headerstr)
{
struct ip_hdr *ip_header;
unsigned char sa[4]; unsigned char da[4];
ip_header = (struct ip_hdr *)malloc(sizeof(struct ip_hdr));
memcpy(ip_header,ip_headerstr,sizeof(struct ip_hdr));
memcpy(sa,(unsigned char *)&ip_header->ip_src_addr,4);
memcpy(da,(unsigned char *)&ip_header->ip_dest_addr,4);
printf("Length of IP packet: %d\n",ntohs(ip_header->ip_len));
printf("Identifier of IP packet: %d\n",ntohs(ip_header->ip_id));
printf("Time To Live: %d\n",ip_header->ip_ttl);
printf("Protocol type: %d\n",ip_header->ip_type);
printf("IP header checksum: %d\n",ip_header->ip_checksum);
printf("Sender IP address: %u.%u.%u.%u\n", sa[0],sa[1],sa[2],sa[3]);
printf("Dest. IP address:%u.%u.%u.%u\n", da[0],da[1],da[2],da[3]);
}
```

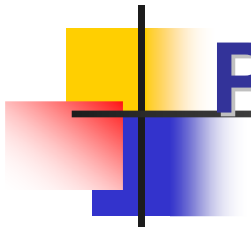
Create UDP packet header


```
unsigned short uh_sport;    // source port
unsigned short uh_dport;    // destination port
unsigned short uh_ulen;     // udp length
unsigned short uh_sum;      // udp header checksum
```



Create UDP packet header

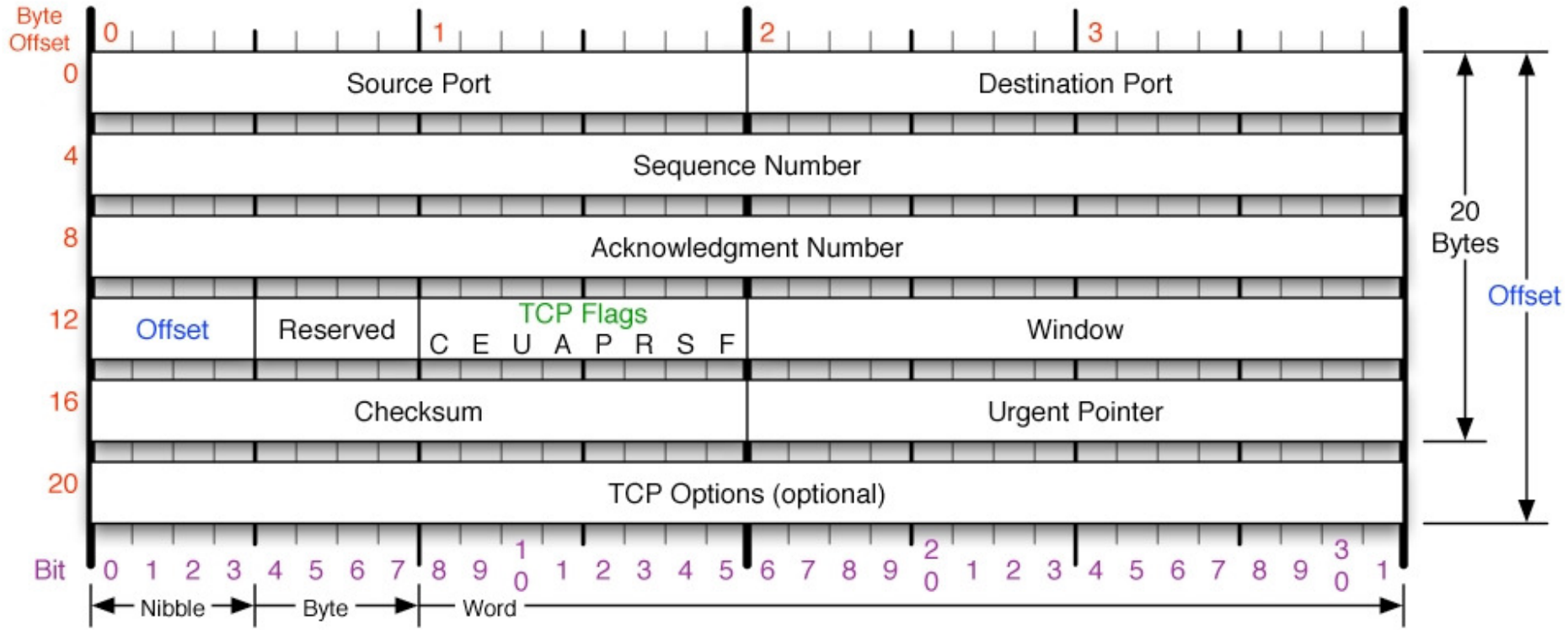
```
unsigned char *create_udphdr(
                    unsigned short sp,
                    unsigned short dp,
                    unsigned short len,
                    unsigned short checksum)
{
struct udp_hdr *udp_header;
udp_header = (struct udp_hdr *)malloc(sizeof(struct udp_hdr));
udp_header->uh_sport = sp; udp_header->uh_dport = dp;
udp_header->uh_ulen = len; udp_header->uh_sum = checksum;
return ((unsigned char *)udp_header);
}
```



Print UDP packet header

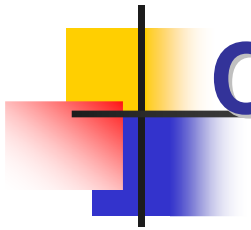
```
void  print_udphdr(unsigned char *udp_headerstr)
{
struct udp_hdr *udp_header;
udp_header = (struct udp_hdr *)malloc(sizeof(struct udp_hdr));
memcpy(udp_header,udp_headerstr,sizeof(struct udp_hdr));
printf("Source UDP port: %d\n",ntohs(udp_header->uh_sport));
printf("Destination UDP port: %d\n",ntohs(udp_header->uh_dport));
printf("UDP packet length: %d\n",ntohs(udp_header->uh_ulen));
printf("UDP header checksum: %d\n",ntohs(udp_header->uh_sum));
}
```

Create TCP packet header



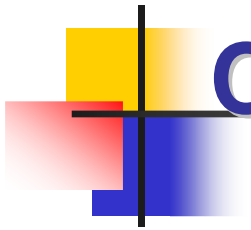
Create TCP packet header

```
unsigned short tcp_src_port;    // source TCP port
unsigned short tcp_dest_port;   // destination TCP port
unsigned int tcp_seq;           // TCP sequence number
unsigned int tcp_ack;           // TCP acknowledgement number
unsigned char resoff;           // 4-bits + TCP offset
unsigned char tcp_flags;        // TCP flags
#define TCP_FIN   0x01
#define TCP_SYN   0x02
#define TCP_RST   0x04
#define TCP_PUSH  0x08
#define TCP_ACK   0x10
#define TCP_URG   0x20
unsigned short tcp_window;      // TCP window size
unsigned short tcp_checksum;    // TCP checksum
unsigned short tcp_urgent;      // TCP urgent pointer
```



Create TCP packet header

```
unsigned char *create_tcphdr(
                    unsigned short sp,
                    unsigned short dp,
                    unsigned int seq,
                    unsigned int ack,
                    unsigned char resoff,
                    unsigned char flags,
                    unsigned short window,
                    unsigned short checksum,
                    unsigned short urgp)
```



Create TCP packet header

```
...
tcp_header = (struct tcp_hdr *)malloc(sizeof(struct tcp_hdr));
tcp_header->tcp_src_port = sp;tcp_header->tcp_dest_port = dp;
tcp_header->tcp_seq = seq;tcp_header->tcp_ack = ack;
tcp_header->resoff = resoff; tcp_header->tcp_flags = flags;
tcp_header->tcp_window = window;
tcp_header->tcp_checksum = checksum;
tcp_header->tcp_urgent = urgp;
return ((unsigned char *)tcp_header);
}
```

Print TCP packet header

```
void  print_tcphdr(unsigned char *tcp_headerstr)
{ struct tcp_hdr *tcp_header;
tcp_header = (struct tcp_hdr *)malloc(sizeof(struct tcp_hdr));
memcpy(tcp_header,tcp_headerstr,sizeof(struct tcp_hdr));
printf("Source TCP port: %u\n",ntohs(tcp_header->tcp_src_port));
printf("Destination TCP port: %u\n",ntohs(tcp_header->tcp_dest_port));
printf("TCP sequence number: %u\n",ntohl(tcp_header->tcp_seq));
printf("TCP acknowledge number: %u\n",ntohl(tcp_header->tcp_ack));
printf("TCP flags: %2.2x\n",tcp_header->tcp_flags);
if(tcp_header->tcp_flags & TCP_FIN) printf("FIN ");
if(tcp_header->tcp_flags & TCP_SYN) printf("SYN ");
if(tcp_header->tcp_flags & TCP_RST) printf("RST ");
if(tcp_header->tcp_flags & TCP_PUSH) printf("PUSH ");
if(tcp_header->tcp_flags & 0x10) printf("ACK ");
if(tcp_header->tcp_flags & TCP_URG) printf("URG "); printf("\n");
printf("TCP window: %u\n",ntohs(tcp_header->tcp_window));
printf("TCP checksum: %u\n",ntohs(tcp_header->tcp_checksum)); }
```

My Ethernet packet send

```
#include "mypackage.h"
#define SRC_ETHER_ADDR          "aa:aa:aa:aa:aa:aa"
#define DST_ETHER_ADDR          "ff:ff:ff:ff:ff:ff"
#define ETHER_TYPE  0x8000
main(int c, char **a)
{
unsigned char message[128];
int sd, ret,fn,ncar,i=0;
unsigned char *eth;
unsigned char abuf[6];
unsigned char packet[1000];
unsigned short type=htons(ETHER_TYPE);
if(c==1) { printf("Usage: mypacethersend [eth0, wlan0] frame_number\n"); exit(1);}
fn=atoi(a[2]);
printf("Give the data message to send in the frame:\n");
ncar = read(0,message,128);
sd = create_rawsocket(ETH_P_ALL);
ret= bind_rawsocket(a[1],sd,ETH_P_ALL);
eth = (unsigned char *) create_eth(SRC_ETHER_ADDR,DST_ETHER_ADDR,ETHER_TYPE);
memcpy(packet,eth,ETHER_HDR_LEN);
memcpy(packet+ETHER_HDR_LEN,message,ncar);
while(i<fn)
                {
                send_rawpacket(sd,packet,ETHER_HDR_LEN+ncar);
                sleep(1); i++;
                }
}
```

My Ethernet packet receive

```
#include "mypackage.h"
main(int c, char **a)
{
unsigned char message[128];
int sd, ret,ncar=128;
unsigned char *eth; unsigned char abuf[6];
unsigned char packet[1000]; unsigned int i=0;
int type=0,fn=0; char v=0;
if(c==1) { printf("Usage: mypacetherrecv [eth0, wlan0] frame_number [d,p]\n"); exit(1);}
sd = create_rawsocket(ETH_P_ALL);
ret= bind_rawsocket(a[1],sd,ETH_P_ALL);
if(c>2) fn=atoi(a[2]);  if(c>3) v=*a[3];
while(i<fn)
                {
                recv_rawpacket(sd,packet,ETHER_HDR_LEN+ncar);
                type= type_ethhdr(packet);
                if(v=='d') hex_dump(packet,64);
                switch (type) {
                  case 1 : printf("IP protocol\n");break;
                  case 2 : printf("ARP protocol\n");break;
                  case 3 : printf("RARP protocol\n");break;
                  default:  printf("other protocol\n");break;
                  }
                i++;
                if(v=='p') print_ethhdr(packet);
                }
}
```

My IP packet send

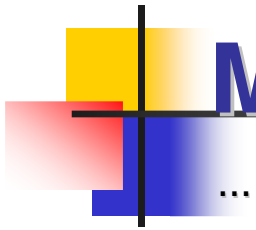
```
#include "mypackage.h"
#define SA "172.19.64.142"
#define DA "172.19.71.255"
#define ETHER_TYPE 0x0800
#define SRC_ETHER_ADDR          "aa:aa:aa:aa:aa:aa"
#define DST_ETHER_ADDR          "ff:ff:ff:ff:ff:ff"
main(int c, char **a)
{
unsigned char message[128];
int sd, ret, ncar=128;    // 20 bytes - IP packet header
unsigned char *eth; unsigned char *ip;
unsigned char packet[1000]; unsigned int i=0;
int pn=0; char v=0;
if(c==1) { printf("Usage: mypacetheripsend [eth0, wlan0] packet_number \n"); exit(1);}
pn= atoi(a[2]);
sd = create_rawsocket(ETH_P_ALL);
ret= bind_rawsocket(a[1],sd,ETH_P_ALL);
if(ret<0) { printf("Can't bind to %s\n",a[1]); exit(2);}
eth = (unsigned char *) create_eth(SRC_ETHER_ADDR,DST_ETHER_ADDR,ETHER_TYPE);
memcpy(packet,eth,ETHER_HDR_LEN);
ip = (unsigned char *)create_iphdr(0x45,0,htons(128),htons(2010),0,128,6,0,inet_addr(SA),inet_addr(DA));
memcpy(packet+ETHER_HDR_LEN,ip, IP_HDR_LEN);
while(i<pn)
                { send_rawpacket(sd,packet,ETHER_HDR_LEN+IP_HDR_LEN+ncar);
                i++; sleep(1);
                }
}
```

My IP packet receive

```
#include "mypackage.h"
main(int c, char **a)
{
unsigned char message[128];
int sd, ret,ncar=128;    // 28 bytes - ARP packet, 20 bytes - IP packet
unsigned char *eth;  unsigned char abuf[6];
unsigned char packet[1000];  unsigned int i=0;
int ethtype=0;  int iptype=0,pn=0;  char v=0;
if(c==1) { printf("Usage: mypacetheriprecv [eth0, wlan0] packet_number [d,p]\n"); exit(1);}
sd = create_rawsocket(ETH_P_ALL);
ret= bind_rawsocket(a[1],sd,ETH_P_ALL);
pn= atoi(a[2]);  if(c>3) v=*a[3];
while(i<pn)
                {
                recv_rawpacket(sd,packet,ETHER_HDR_LEN+ncar);
                ethtype= type_ethhdr(packet);
                if(ethtype==1)
                {
                iptype=type_iphdr(packet+14);
                if(iptype==1) printf("\n -------- IMCP packet --------\n");
                                if(iptype==2) printf("\n -------- TCP packet --------\n");
                                if(iptype==3) printf("\n -------- UDP packet --------\n");
                if(v=='p') print_iphdr(packet+14);
                if(v=='d') hex_dump(packet+14,64);
                                }
                i++;
                }
}
```

My UDP packet send

```
#include "mypackage.h"
#define SA "172.19.64.142"
#define DA "172.19.71.255"
#define ETHER_TYPE 0x0800
#define SRC_ETHER_ADDR    "aa:aa:aa:aa:aa:aa"
#define DST_ETHER_ADDR    "ff:ff:ff:ff:ff:ff"
main(int c, char **a)
{
unsigned char message[128];
int sd, ret, ncar=128;    // 20 bytes - IP packet header
unsigned char *eth; unsigned char *ip;
unsigned char *udp; unsigned char packet[1000];
unsigned int i=0; int pn=0; char v=0;
if(c==1) { printf("Usage: mypacetheripsend [eth0, wlan0] packet_number \n");
exit(1);}
pn= atoi(a[2]);  sd = create_rawsocket(ETH_P_ALL);
ret= bind_rawsocket(a[1],sd,ETH_P_ALL);
…
```



My UDP packet send

...

```
if(ret<0) { printf("Can't bind to %s\n",a[1]); exit(2);}
printf("Give the packet message (data) to send :\n");
ncar= read(0,message,128);
eth = (unsigned char *)
create_eth(SRC_ETHER_ADDR,DST_ETHER_ADDR,ETHER_TYPE);
memcpy(packet,eth,ETHER_HDR_LEN);
ip = (unsigned char *)create_iphdr(0x45,0,htons(128),htons(2010),0,128,UDP,0,
inet_addr(SA), inet_addr(DA));
memcpy(packet+ETHER_HDR_LEN,ip, IP_HDR_LEN);
udp = (unsigned char *)create_udphdr(htons(80),htons(80),htons(32),0);
memcpy(packet+ETHER_HDR_LEN+IP_HDR_LEN,udp,UDP_HDR_LEN);
memcpy(packet+ETHER_HDR_LEN+IP_HDR_LEN+UDP_HDR_LEN,message, ncar);
while(i<pn)
{
send_rawpacket(sd,packet,ETHER_HDR_LEN+IP_HDR_LEN+UDP_HDR_LEN+ncar);
i++; sleep(1);
}
}
```

My UDP packet receive

```
#include "mypackage.h"
main(int c, char **a)
{
unsigned char message[128];
int sd, ret,ncar=128;    // 28 bytes - ARP packet, 20 bytes - IP packet
unsigned char *eth;  unsigned char abuf[6];
unsigned char packet[1000];  unsigned int i=0;
int ethtype=0;  int iptype=0,pn=0;  char v=0;
if(c==1)
  { printf("Usage: mypacetheriprecv [eth0, wlan0] packet_number [d,p]\n"); exit(1);}
sd = create_rawsocket(ETH_P_ALL);
ret= bind_rawsocket(a[1],sd,ETH_P_ALL);
pn= atoi(a[2]);
if(c>3) v=*a[3];
...
```

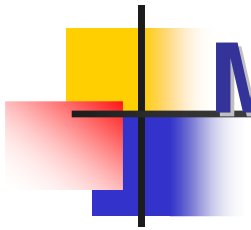
My UDP packet receive

...

```
while(i<pn)
                {
                recv_rawpacket(sd,packet,ETHER_HDR_LEN+IP_HDR_LEN+UDP_HDR_LEN+ncar);
                ethtype= type_ethhdr(packet);
                  if(ethtype==1)
                  {
                  iptype=type_iphdr(packet+ETHER_HDR_LEN);
                  if(iptype==17)
                                {
                                printf("\n---- UDP packet ----\n");
                                if(v=='p')
                                                {
                                                print_iphdr(packet+ETHER_HDR_LEN);
                                                print_udphdr(packet+ETHER_HDR_LEN+IP_HDR_LEN);
                                                }
                                if(v=='d') hex_dump(packet,64);
                                }
                  }
                i++;
                }
}
```

My TCP packet send

```
#include "mypackage.h"
#define SA "172.19.64.142"
#define DA "172.19.71.255"
#define ETHER_TYPE 0x0800
#define SRC_ETHER_ADDR          "aa:aa:aa:aa:aa:aa"
#define DST_ETHER_ADDR          "ff:ff:ff:ff:ff:ff"
main(int c, char **a)
{
unsigned char message[128];
int sd, ret, ncar=128;    // 20 bytes - IP packet header
unsigned char *eth;
unsigned char *ip;
unsigned char *tcp;
unsigned char packet[1000];
unsigned int i=0;
int pn=0;
char v=0;
if(c==1) { printf("Usage: mypacetheriptcpsend [eth0, wlan0] packet_number \n"); exit(1);}
pn= atoi(a[2]);
sd = create_rawsocket(ETH_P_ALL);
ret= bind_rawsocket(a[1],sd,ETH_P_ALL);
if(ret<0) { printf("Can't bind to %s\n",a[1]); exit(2);}
printf("Give the packet message (data) to send :\n");
ncar= read(0,message,128);
eth = (unsigned char *) create_eth(SRC_ETHER_ADDR,DST_ETHER_ADDR,ETHER_TYPE);
memcpy(packet,eth,ETHER_HDR_LEN);
…
```



My TCP packet send

```
...
ip = (unsigned char *)  create_iphdr(
0x45,0,                              // verlen,tos
htons(IP_HDR_LEN+TCP_HDR_LEN+128),htons(2010),    // totlen,id
0,128,6,cksum((unsigned short *)(packet+ETHER_HDR_LEN),IP_HDR_LEN),   // cksum(),
TTL,proto, inet_addr(SA),inet_addr(DA));
memcpy(packet+ETHER_HDR_LEN,ip, IP_HDR_LEN);
tcp =(unsigned char *)
create_tcphdr(htons(80),htons(80),htonl(1111),htonl(2222),0,0x02,htons(1000),0,0);
memcpy(packet+ETHER_HDR_LEN+IP_HDR_LEN,tcp,TCP_HDR_LEN);
memcpy(packet+ETHER_HDR_LEN+IP_HDR_LEN+TCP_HDR_LEN,message, ncar);
while(i<pn)
                {
                send_rawpacket(sd,packet,ETHER_HDR_LEN+IP_HDR_LEN+TCP_HDR_LEN+ncar);
                i++; sleep(1);
                }

}
```

My TCP packet receive

```
 include "mypackage.h"
main(int c, char **a)
{
unsigned char message[128];
int sd, ret,ncar=128;    // 28 bytes - ARP packet, 20 bytes - IP packet
unsigned char *eth;
unsigned char abuf[6];
unsigned char packet[1000];  unsigned int i=0;
int ethtype=0;
int iptype=0,pn=0;
char v=0;
if(c==1)
  { printf("Usage: mypacetheriptcprecv [eth0, wlan0] packet_number [d,p]\n");
exit(1);}
sd = create_rawsocket(ETH_P_ALL);
ret= bind_rawsocket(a[1],sd,ETH_P_ALL);
pn= atoi(a[2]);
if(c>3) v=*a[3];

...
```

My TCP packet receive

```
...
while(i<pn)
                {
                recv_rawpacket(sd,packet,ETHER_HDR_LEN+IP_HDR_LEN+TCP_HDR_LEN+ncar);
                ethtype= type_ethhdr(packet);
                  if(ethtype==1)
                  {
                  iptype=type_iphdr(packet+ETHER_HDR_LEN);
                  if(iptype==6)
                                {
                                printf("\n---- TCP packet ----\n");
                                if(v=='p')
                                                {
                                                print_iphdr(packet+ETHER_HDR_LEN);
                                                print_tcphdr(packet+ETHER_HDR_LEN+IP_HDR_LEN);
                                                }
                                if(v=='d') hex_dump(packet,64);
                                }
                  } i++;
                }

}
```

Summary

- Creating and Binding raw sockets

- Creating and Displaying Ethernet frames
- Creating and Displaying IP packets
- Creating and Displaying UDP packets
- Creating and Displaying TCP packets

- Sending and Receiving Ethernet frames
- Sending and Receiving IP packets
- Sending and Receiving UDP packets
- Sending and Receiving TCP packets