* In the previous code "exp.c" we made a simple sniffer which created a raw socket and started receiving on it. But it had few drawbacks :

1. Could sniff only incoming data.
2. Could sniff only TCP or UDP or ICMP or any one protocol packets at a time.
3. Provided IP frames , so ethernet headers were not available.

* In this Program "sniffall.c" we are going to modify the same code to fix the above 3 drawbacks. However we shall not be using libpcap. This will be done using pure linux sockets.
* The difference is very small and is 2 lines :

Instead of :

sock\_raw = socket(AF\_INET , SOCK\_RAW , IPPROTO\_TCP);

We do :

sock\_raw = socket( AF\_PACKET , SOCK\_RAW , htons(ETH\_P\_ALL)) ;

//Optional

//setsockopt(sock\_raw , SOL\_SOCKET , SO\_BINDTODEVICE , "eth0" , strlen("eth0")+ 1 );

and we are done.

* Now it will :
  1. Sniff both incoming and outgoing traffic.
  2. Sniff ALL ETHERNET FRAMES , which includes all kinds of IP packets and even more if there are any.
  3. Provides the Ethernet headers too , which contain the mac addresses.
* The setsockopt line is optional.
* It’s important to provide the correct interface name to setsockopt , eth0 in this case and in most cases. So maybe you would like to present the user with a list of interfaces available and allow him to choose the one to be sniffed.
* In the above log we can see the ethernet headers being printed. They show the source and destination mac address along with the packet protocol. 8 means IP protocol
* As

**Note :**

1. If you want to sniff only IP and ARP packets for example then you can try this :

sock\_raw = socket( AF\_PACKET , SOCK\_RAW , htons(ETH\_P\_IP|ETH\_P\_ARP)) ;

**The complete list of protocols is found in /usr/include/linux/if\_ether.h**

/\*

 \*  These are the defined Ethernet Protocol ID's.

 \*/

#define ETH\_P\_LOOP  0x0060      /\* Ethernet Loopback packet \*/

#define ETH\_P\_PUP   0x0200      /\* Xerox PUP packet     \*/

#define ETH\_P\_PUPAT 0x0201      /\* Xerox PUP Addr Trans packet  \*/

#define ETH\_P\_IP    0x0800      /\* Internet Protocol packet \*/

#define ETH\_P\_X25   0x0805      /\* CCITT X.25           \*/

#define ETH\_P\_ARP   0x0806      /\* Address Resolution packet    \*/

#define ETH\_P\_BPQ   0x08FF      /\* G8BPQ AX.25 Ethernet Packet  [ NOT AN OFFICIALLY REGISTERED ID ] \*/

#define ETH\_P\_IEEEPUP   0x0a00      /\* Xerox IEEE802.3 PUP packet \*/

#define ETH\_P\_IEEEPUPAT 0x0a01      /\* Xerox IEEE802.3 PUP Addr Trans packet \*/

#define ETH\_P\_DEC       0x6000          /\* DEC Assigned proto           \*/

#define ETH\_P\_DNA\_DL    0x6001          /\* DEC DNA Dump/Load            \*/

#define ETH\_P\_DNA\_RC    0x6002          /\* DEC DNA Remote Console       \*/

#define ETH\_P\_DNA\_RT    0x6003          /\* DEC DNA Routing              \*/

#define ETH\_P\_LAT       0x6004          /\* DEC LAT                      \*/

#define ETH\_P\_DIAG      0x6005          /\* DEC Diagnostics              \*/

#define ETH\_P\_CUST      0x6006          /\* DEC Customer use             \*/

#define ETH\_P\_SCA       0x6007          /\* DEC Systems Comms Arch       \*/

#define ETH\_P\_TEB   0x6558      /\* Trans Ether Bridging     \*/

#define ETH\_P\_RARP      0x8035      /\* Reverse Addr Res packet  \*/

#define ETH\_P\_ATALK 0x809B      /\* Appletalk DDP        \*/

#define ETH\_P\_AARP  0x80F3      /\* Appletalk AARP       \*/

#define ETH\_P\_8021Q 0x8100          /\* 802.1Q VLAN Extended Header  \*/

#define ETH\_P\_IPX   0x8137      /\* IPX over DIX         \*/

#define ETH\_P\_IPV6  0x86DD      /\* IPv6 over bluebook       \*/

#define ETH\_P\_PAUSE 0x8808      /\* IEEE Pause frames. See 802.3 31B \*/

#define ETH\_P\_SLOW  0x8809      /\* Slow Protocol. See 802.3ad 43B \*/

#define ETH\_P\_WCCP  0x883E      /\* Web-cache coordination protocol

                     \* defined in draft-wilson-wrec-wccp-v2-00.txt \*/

#define ETH\_P\_PPP\_DISC  0x8863      /\* PPPoE discovery messages     \*/

#define ETH\_P\_PPP\_SES   0x8864      /\* PPPoE session messages   \*/

#define ETH\_P\_MPLS\_UC   0x8847      /\* MPLS Unicast traffic     \*/

#define ETH\_P\_MPLS\_MC   0x8848      /\* MPLS Multicast traffic   \*/

#define ETH\_P\_ATMMPOA   0x884c      /\* MultiProtocol Over ATM   \*/

#define ETH\_P\_LINK\_CTL  0x886c      /\* HPNA, wlan link local tunnel \*/

#define ETH\_P\_ATMFATE   0x8884      /\* Frame-based ATM Transport

                     \* over Ethernet

                     \*/

#define ETH\_P\_PAE   0x888E      /\* Port Access Entity (IEEE 802.1X) \*/

#define ETH\_P\_AOE   0x88A2      /\* ATA over Ethernet        \*/

#define ETH\_P\_TIPC  0x88CA      /\* TIPC             \*/

#define ETH\_P\_1588  0x88F7      /\* IEEE 1588 Timesync \*/

#define ETH\_P\_FCOE  0x8906      /\* Fibre Channel over Ethernet  \*/

#define ETH\_P\_FIP   0x8914      /\* FCoE Initialization Protocol \*/

#define ETH\_P\_EDSA  0xDADA      /\* Ethertype DSA [ NOT AN OFFICIALLY REGISTERED ID ] \*/

/\*

 \*  Non DIX types. Won't clash for 1500 types.

 \*/

#define ETH\_P\_802\_3 0x0001      /\* Dummy type for 802.3 frames  \*/

#define ETH\_P\_AX25  0x0002      /\* Dummy protocol id for AX.25  \*/

#define ETH\_P\_ALL   0x0003      /\* Every packet (be careful!!!) \*/

#define ETH\_P\_802\_2 0x0004      /\* 802.2 frames         \*/

#define ETH\_P\_SNAP  0x0005      /\* Internal only        \*/

#define ETH\_P\_DDCMP     0x0006          /\* DEC DDCMP: Internal only     \*/

#define ETH\_P\_WAN\_PPP   0x0007          /\* Dummy type for WAN PPP frames\*/

#define ETH\_P\_PPP\_MP    0x0008          /\* Dummy type for PPP MP frames \*/

#define ETH\_P\_LOCALTALK 0x0009      /\* Localtalk pseudo type    \*/

#define ETH\_P\_CAN   0x000C      /\* Controller Area Network      \*/

#define ETH\_P\_PPPTALK   0x0010      /\* Dummy type for Atalk over PPP\*/

#define ETH\_P\_TR\_802\_2  0x0011      /\* 802.2 frames         \*/

#define ETH\_P\_MOBITEX   0x0015      /\* Mobitex (kaz@cafe.net)   \*/

#define ETH\_P\_CONTROL   0x0016      /\* Card specific control frames \*/

#define ETH\_P\_IRDA  0x0017      /\* Linux-IrDA           \*/

#define ETH\_P\_ECONET    0x0018      /\* Acorn Econet         \*/

#define ETH\_P\_HDLC  0x0019      /\* HDLC frames          \*/

#define ETH\_P\_ARCNET    0x001A      /\* 1A for ArcNet :-)            \*/

#define ETH\_P\_DSA   0x001B      /\* Distributed Switch Arch. \*/

#define ETH\_P\_TRAILER   0x001C      /\* Trailer switch tagging   \*/

#define ETH\_P\_PHONET    0x00F5      /\* Nokia Phonet frames          \*/

#define ETH\_P\_IEEE802154 0x00F6     /\* IEEE802.15.4 frame       \*/

#define ETH\_P\_CAIF  0x00F7      /\* ST-Ericsson CAIF protocol    \*/

1. The code actually captures all ethernet packets including arp packets. In case of ARP packets the protocol field of the ethernet header would contain the value 0x0806

So you can check like this

if( ntohs(eth->h\_proto) == 0x0806 ) { // arp packet, process }

that’s the basic idea, you would need to parse the arp packet according to its packet structure, details of which can found on google.

1. how to extract payload data

in the functions print\_icmp\_packet, print\_tcp\_packet and print\_udp\_packet the payload can be extracted as follows

char \*payload = Buffer + header\_size;

int payload\_size = Size – header\_size;

buffer is the full packet content. removing the headers from the buffer will give the payload.

1. What following program lines do
   1. struct iphdr \*iph = (struct iphdr \*)( Buffer + sizeof(struct ethhdr) );

the above line gets the pointer to the ip header in the structure ‘iphdr’ which can be used to access individual fields of the header

* 1. iphdrlen = iph->ihl\*4

the above computes the length of the ip header using the ‘ihl’ field of the ip header. the ihl field contains the ip header length in total bytes/4.

* 1. struct tcphdr \*tcph=(struct tcphdr\*)(Buffer + iphdrlen + sizeof(struct ethhdr));

the above line gets the pointer to the tcp header in the structure ‘tcphdr’ which can be used to access the individual fields of the tcp header.

1. same code should work when sniffing packets over wi-fi.