Network Programming Assignment (Raw Socket)

Name – Rahul Kumar

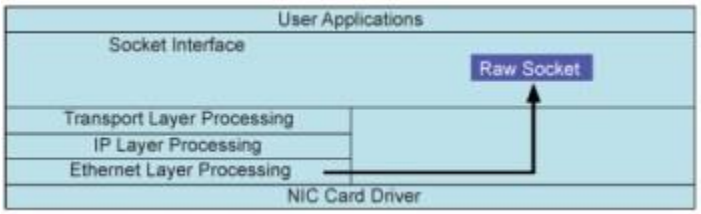
Branch – computer Science (3rd year)

Roll No : 171210046

Faculty : Dr. Ravi Kumar Arya

Write a report of 500 words on Raw Sockets.

A raw socket is used to receive raw packets. This means packets received at the Ethernet layer will directly pass to the raw socket. Stating it precisely, a raw socket bypasses the normal TCP/IP processing and sends the packets to the specific user application (see Figure 1).



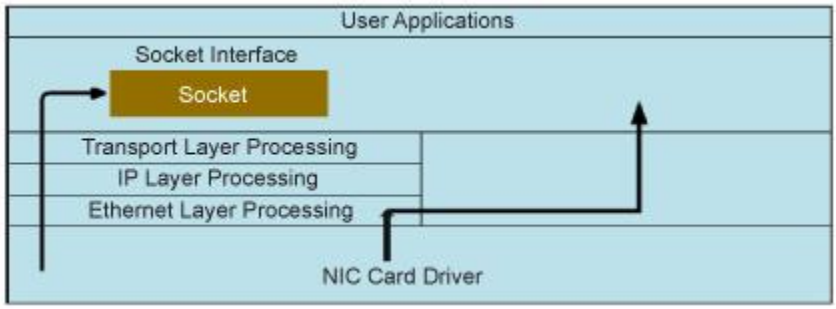
(Figure 1)

Raw socket

Other sockets like stream sockets and data gram sockets receive data from the transport layer that contains no headers but only the payload. This means that there is no information about the source IP address and MAC address. If applications running on the same machine or on different machines are communicating, then they are only exchanging data.

The purpose of a raw socket is absolutely different. A raw socket allows an application to directly access lower level protocols, which means a raw socket receives un-extracted packets (see

Figure 2). There is no need to provide the port and IP address to a raw socket, unlike in the case of stream and datagram sockets.



(Figure 2)

The biggest problem with RAW-sockets (also the PACKET-sockets we discuss later) is that there is no uniform API for using RAW-sockets under different operating systems:

• The provided APIs differ in regard to the used byte order. Depending on the operating system the fields of the packets have to be filled with data in network- or host-byte-order.

• Differences also exists in the types of packets and protocols that can be created using the RAW-socket API.

• The usage and functionality of the APIs is also different for each operating system. • Some operating systems do not allow certain packet types to be received using the RAW-socket API.

• There are also differences in the definitions and paths of the necessary header files provided by each operating system.

• The required access levels for using the RAW-socket API can also differ. However most operating systems require root or admin access permissions to use them. The user has to keep these things in mind if he wants to use the RAW-socket API, especially if he plans to use them across different operating systems.

Raw Sockets and Security

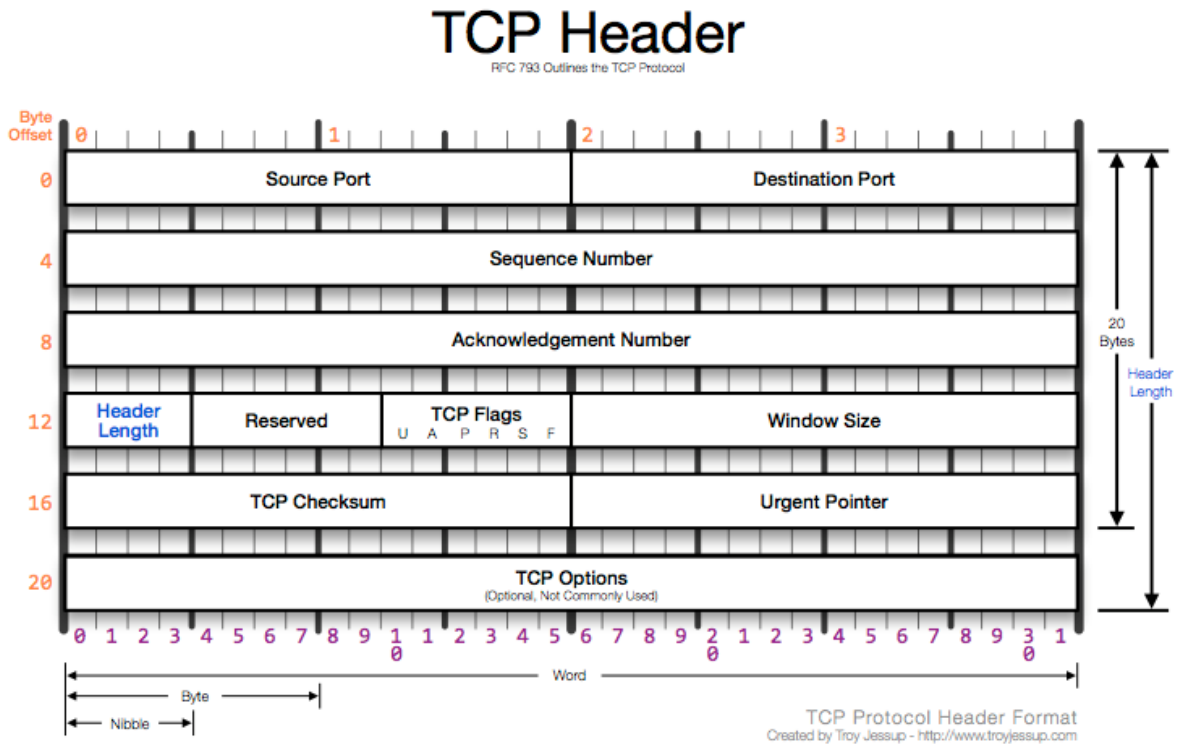
An important thing worth noting about this is that there is important security-related information stored in TCP and UDP headers:

For instance, the destination port is stored in the TCP headers. This means that packets read from raw sockets don't have any notion of "port".

To step back a little bit, an application using TCP or UDP must, when opening up a STREAM or DGRAM socket, declare a port to receive data on. When the application reads data from that socket, they will only see data that was sent to that particular port.

Now, with raw sockets, because network-level IP packets do not have a notion of "port", all packets coming in over the server's network device can be read. Applications that open raw sockets have to do the work of filtering out packets that are not relevant themselves, by parsing the TCP headers. The security implications of this are pretty serious–it means that applications with a raw socket open can read any inbound network packets, including those headed to other applications running on the system, which may or may not be run as the same unix user as the application with the raw socket open.

To prevent this from happening, Linux requires that any program that accesses raw sockets be run as root. Actually running network programs as root is dangerous, especially for a sufficiently complicated program, like a TCP implementation.



Implementations for different Operating Systems

First we want to give a short overview over some of the APIs and libraries that are available for RAW-socket and Data Link Layer network programming. In general we want to split the available libraries into two parts, the APIs provided by the operating systems and independent libraries that are available for multiple operating systems.

1 Windows

Windows as a operating system is pretty restricted when it comes to RAW-socket, PACKET-socket and Data Link Layer programming. In general it is difficult to get the programs running under Windows and the available options are pretty limited. For that reason in general 3rd party libraries are recommended if the user wants to do network programming on Windows systems and still write portable code.

2 Linux

Linux is one of the operating systems for which it is easy to do RAW-socket and Data Link Layer Programming. It provides the APIs to do both, but the kernel has to be compiled with the option to support the options.

Two ways to define a raw socket

1. Let kernel fill in the IP header and we just create transport protocol header or

2. We are responsible for the whole packet and create both the IP header and underlying protocol headers

Creating Raw Sockets

int sockfd; sockfd = socket(AF_INET, SOCK_RAW, protocol);

1. This creates a raw socket and lets the kernel fill in the ip header, we will fill in the protocol header for ICMP or IGMP or TCP or UDP - const int on = 1; setsockopt (sockfd, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on);

2. Setting option, IP_HDRINCL allows us to create our own IP header and not let the kernel do it for us

Raw Sockets Operation (ICMP)

User Space				Kernel Space
Socket App		Linux		Protocol
socket ( )	→	Create Socket	→	ICMP
	← Ok			
Sendto	→	Pass data thru local	→	IP, Internet
	← ok	stack to remote host		

Raw Socket Input

• Most ICMP packets are passed to raw socket – Some exceptions for Berkeley-derived implementations

• All IGMP packets are passed to a raw socket

• All IP datagrams with protocol field that kernel does not understand (process) are passed to a raw socket – If packet has been fragmented, packet is reassembled before being passed to raw socket

Raw Socket Input

• Most ICMP packets are passed to raw socket – Some exceptions for Berkeley-derived implementations • All IGMP packets are passed to a raw socket

• All IP datagrams with protocol field that kernel does not understand (process) are passed to a raw socket – If packet has been fragmented, packet is reassembled before being passed to raw socket

Raw Socket Output

• Normal output performed using sendto or sendmsg – Write or send can be used if the socket has been connected

• If IP_HDRINCL not set, starting addr of data (buf) specifies first byte following IP header that kernel builds

• If IP_HDRINCL is set, starting addr of data identifies first byte of IP header, that we build

Attempting to read from a raw socket

To see what happens when we read data from sockets, let's use this C program that simply sniffs packets entering the system and prints out some information about the packet as an example:

```
// raw_sock.c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<netinet/ip.h>
#include<sys/socket.h>
#include<arpa/inet.h>
```

```
int main()
{
// Structs that contain source IP addresses
struct sockaddr_in source_socket_address, dest_socket_address;

int packet_size;
// Allocate string buffer to hold incoming packet data
    unsigned char *buffer = (unsigned char *)malloc(65536);
    // Open the raw socket
    int sock = socket (PF_INET, SOCK_RAW, IPPROTO_TCP);
    if(sock == -1)

    {
 //socket creation failed, may be because of non-root privileges
      perror("Failed to create socket");
      exit(1);
    }
    while(1)
   {
      // recvfrom is used to read data from a socket
      packet_size = recvfrom(sock , buffer , 65536,0,NULL,NULL);
      if (packet_size == -1) {
        printf("Failed to get packets\n");
        return 1;
   }
```

```
struct iphdr *ip_packet = (struct iphdr *)buffer;
                          memset(&source_socket_address, 0,
sizeof(source_socket_address));

source_socket_address.sin_addr.s_addr = ip_packet->saddr;

memset(&dest_socket_address,0,sizeof(dest_socket_address));

dest_socket_address.sin_addr.s_addr = ip_packet->daddr;

printf("Incoming Packet: \n");

printf("Packet Size (bytes): %d\n",ntohs(ip_packet->tot_len));

printf("SourceAddress:%s\n",(char*)inet_ntoa(source_socket_addre
ss.sin_addr));


printf("DestinationAddress:%s\n",(char*)inet_ntoa(dest_socket_ad
dress.sin_addr));

printf("Identification: %d\n\n", ntohs(ip_packet->id));

}

    return 0;
}
```