CENG393 Computer Networks
Labwork 4

Socket Programming: Raw Sockets

1 Raw Sockets

Previously, we have studied TCP and UDP sockets. Remember that a TCP socket is defined by requesting a connection-oriented socket from the operating system via:

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
// alternatively;
// sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

And a UDP socket is defined by requesting a connectionless datagram socket:

```
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
// alternatively;
// sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

When the third parameter of socket system call is left 0 (desired transport layer protocol is not specified by name), the operating system associates the default protocol available to the chosen socket type. Remember that the default protocol associated with SOCK_STREAM is TCP and for SOCK_DGRAM it is UDP. When these types of sockets are requested, the operating system automatically handles some duties related to the chosen transport layer protocol itself, such as preparing the TCP / UDP headers.

In this laboratory, you will be introduced to raw sockets. Raw sockets are used for implementing custom protocols in user programs. Because the user has more access to bare network operations with raw sockets, the operating system will prevent unauthorized users to execute programs that utilize raw sockets. In order to run your own raw socket programs, you must temporarily elevate your privileges (by using the sudo command) or use a privileged account (root).

Instead of defining a novel protocol, the code rawclient.c in Appendix A uses a raw socket that uses the standard UDP protocol; therefore the code itself is responsible from generating a proper UDP header. But the code also enables `IP_HDRINCL` IPv4 protocol option, which allows defining custom network layer headers (IPv4 in this case) for a packet instead of using the system generated ones. It's definition from manpages (`man 7 ip`) is as follows:

```
If  enabled,  the user supplies an IP header in front of the user data. Valid only for
SOCK_RAW sockets; see raw(7) for more information.  When  this  flag  is  enabled, the
values set by IP_OPTIONS, IP_TTL, and IP_TOS are ignored.
```

In summary; in addition to defining how the socket operates and what data is transferred between the programs using this socket, the code is now also responsible from generating proper IPv4 and UDP headers and sending them before the data. The diagrams for IPv4 and UDP headers are given in Figure 1.

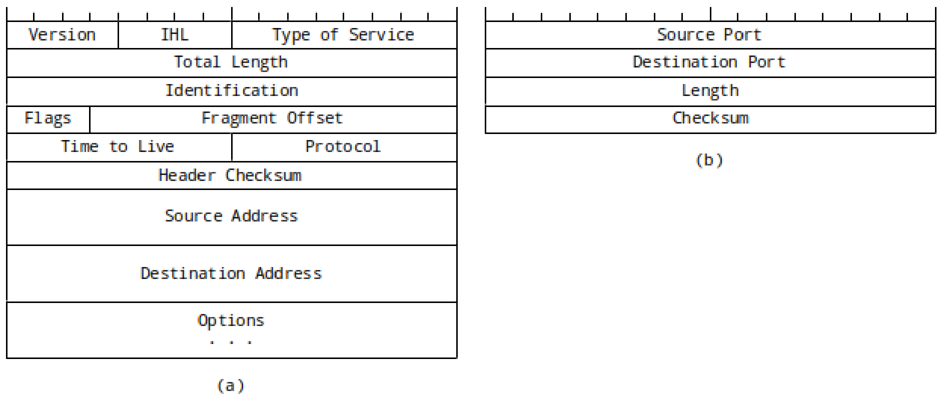

Figure 1: Packet Structures for **a)** IPv4 and **b)** UDP protocols

2 Exercises

Use the code in Appendix A together with the UDP server program we have used before.

1. Modify the UDP port number in udpserver.c so that both programs operate successfully.
2. Modify both programs so that when the given raw socket program sends a message to UDP server, the UDP server sends back a reply message which must be printed on screen by raw socket program.
3. What are the risks of using raw sockets in a program? Why does the operating system prohibit usage of raw sockets by unprivileged user accounts? Can you demonstrate an example?
4. Read “`man 7 raw`”.

Appendices

A rawclient.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <sys/socket.h>

unsigned short ip_cksum(unsigned short*, int);
unsigned short udp_check(struct udphdr*, unsigned short, unsigned long, unsigned long);

struct my_packet {
  struct iphdr ip;
  struct udphdr udp;
  char data[8];
};

int main() {
  int sockfd, one = 1;
  unsigned short local_port = htons(3000), syn_port = htons(2000);
  unsigned long syn_daddr = inet_addr("127.0.0.1"), syn_saddr = inet_addr("127.0.0.1");
  unsigned char in[65000];
  socklen_t size;
  struct my_packet out;
  struct sockaddr_in to = { AF_INET, syn_port }, from = { AF_INET, local_port };
  to.sin_addr.s_addr = syn_daddr;
  from.sin_addr.s_addr = syn_saddr;

  if ((sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_UDP)) < 0) {
    printf("socket creation error\n");
    exit(-1);
  }
  bind(sockfd, (struct sockaddr*) &from, sizeof(from));

  bzero(&out, sizeof(out));
  strcpy(out.data, "testing");
  out.ip.ihl = 5;
  out.ip.version = 4;
  out.ip.tos = 0;
  out.ip.tot_len = htons(sizeof(out));
  out.ip.id = getpid();
  out.ip.frag_off = 0;
  out.ip.ttl = 255;
  out.ip.protocol = IPPROTO_UDP;
  out.ip.saddr = syn_saddr;
  out.ip.daddr = syn_daddr;
  out.ip.check = 0;
  out.ip.check = ip_cksum((unsigned short*) &out.ip, 20);
  out.udp.source = local_port;
  out.udp.dest = syn_port;
  out.udp.len = htons(sizeof(struct udphdr) + 8);
  out.udp.check = 0;
  out.udp.check = udp_check(&out.udp, sizeof(struct udphdr) + 8, out.ip.saddr, out.ip.daddr);

  if (setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL, &one, sizeof(one)))
    printf("setsockopt error\n");
  /* rawsocket.c cont'd */

  if (sendto(sockfd, &out, sizeof(out), 0, (struct sockaddr*) &to, sizeof(to)) < 0)
    printf("sendto error\n");
  printf("%d bytes packet sent to ...\n", (int) sizeof(out));

  if (recvfrom(sockfd, &in, 4096, 0, (struct sockaddr*) &from, &size) < 0)
    printf("recvfrom error \n");

  return 0;
}

unsigned short ip_cksum(unsigned short *buff, int len) {
  unsigned long sum = 0;
  while (len > 1) {
    sum += *buff++;
    len -= 2;
  }
  if (len == 1)
    sum += (*buff & 0xff);
  sum = (sum >> 16) + (sum & 0xffff);
  sum += (sum >> 16);
  sum = ~sum;
  return (sum & 0xffff);
}

unsigned short udp_check(struct udphdr *th, unsigned short len,
    unsigned long saddr, unsigned long daddr) {
  unsigned long sum = 0;
  unsigned short *buff;
  buff = (unsigned short*) &saddr;
  sum += *buff++;
  sum += *buff;
  buff = (unsigned short*) &daddr;
  sum += *buff++;
  sum += *buff;
  sum += IPPROTO_UDP * 256;
  sum += htons(len) & 0xffff;
  buff = (unsigned short*) th;
  while (len > 1) {
    sum += *buff++;
    len -= 2;
  }
  if (len == 1)
    sum += (*buff & 0xff);
  sum = (sum >> 16) + (sum & 0xffff);
  sum += (sum >> 16);
  return ((~sum) & 0xffff);
}
```