Computer Networks Project 1

2015123080 Jaihong Park

I. Introduction/Reference

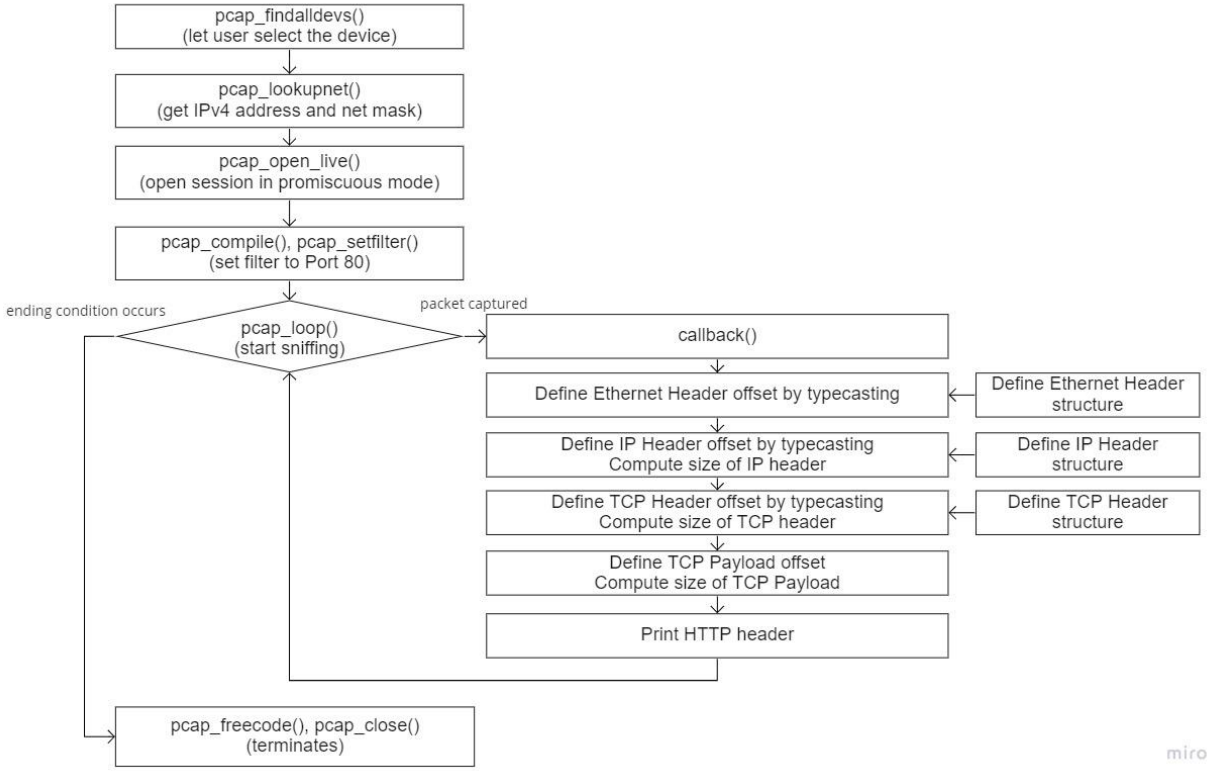
- software environment : ubuntu 20.04, Linux OS
- programming language : C (version 7.5.0)
- reference :

<Man Page of PCAP>
https://www.tcpdump.org/manpages/pcap.3pcap.html

<Programming with pcap>
https://www.tcpdump.org/pcap.html

<Packet Sniffer Develop>
https://g0pher.tistory.com/337?category=816387

II. Flow chart



III. Logical Explanations

1. Finding the device and let user select the device

```
/* find the devices */
printf("-----------list of network devices-----------\n");
tmp = pcap_findalldevs(&alldevsp, errbuf);
if(tmp==-1)
{
    printf("**pcap_findalldevs error**: %s\n", errbuf);
    return -1;
}
/* print the list of devices */
for(d=alldevsp; d; d=d->next){
    printf("%d. %s", i++, d->name);
     if(d->description){
        printf("(%s)\n", d->description);
    } else{
        printf("(None)\n");
    }
}
```

Used *pcap_findalldevs(pcap_if_t **alldevsp, char *errbuf)* function to get list of network devices.

It returns 0 for success, -1 for failure and populates errbuf string with error message.

Print the list of devices with descriptions.

```
pcap_if_t *alldevsp;            // device list pointer
pcap_if_t *d;                   //device pointer
pcap_t *handle;                 //session handler

int tmp =0, i=0;
char errbuf[PCAP_ERRBUF_SIZE];  // Error string
```

These are the pointers. PCAP_ERRBUF_SIZE is 256. Tmp is used for getting result of functions

```c
/* print the list of devices */
for(d=alldevsp; d; d=d->next){
    printf("%d. %s", i++, d->name);
    if(d->description){
        printf("(%s)\n", d->description);
    } else{
        printf("(None)\n");
    }
}
/* let user select the device */
printf("Select the number of device you want to sniff > ");
scanf("%d", &i);

for(d=alldevsp;i>0;d=d->next, i--);
printf("selected device : %s \n", d->name);
```

Print the list of devices with descriptions, get the number of device from the user.

Move the pointer of device list to the next n times, where n is the number that user input.

2. Find the IPv4 network number and net mask of the device

```c
/* Find the IPv4 network number and netmask for the device */
tmp = pcap_lookupnet(d->name, &netp, &maskp, errbuf);
if(tmp == -1){
    printf("**pcap_lookupnet error** : %s\n", errbuf);
    return -1;
}
```

Used *pcap_lookupnet(char *device, bpf_u_int32 *netp, bpf_u_int32 *maskp, char *errbuf)* to get IPv4 network number(IP address) and network mask for the device.

We need network number netp to apply filter.

IPv4 network number is assigned to netp, network mask is assigned to maskp.

```c
bpf_u_int32 maskp;              // The netmask of the sniffing device
bpf_u_int32 netp;               // The IP of the sniffing device
```

3. Open the device for sniffing

```
/* created the session in promiscuous mode, buffer size 8192, 1000ms time out */
handle = pcap_open_live(d->name, BUFSIZ, 1, 1000, errbuf);
if(handle == NULL) {
    fprintf(stderr, "**pcap_open_live error** %s : %s\n", d->name, errbuf);
    return -1;
}
```

Created the sniffing session with
*pcap_t *pcap_open_live(char *device, int snaplen, int promisc, int to_ms, char *ebuf)* function.
I chose promiscuous mode to sniff all traffic on the wire, 1 second time out to read.

4. Filter traffic by HTTP

```
/* Compile and apply the filter to HTTP(port 80) */
if(pcap_compile(handle, &fp, filter_exp, 0, netp) == -1) {
    fprintf(stderr, "**pcap_compile error** %s: %s\n", filter_exp, pcap_geterr(handle));
    return -1;
}
if(pcap_setfilter(handle, &fp) == -1){
    fprintf(stderr, "**pcap_setfilter error** %s: %s\n", filter_exp, pcap_geterr(handle));
    return -1;
}
```

In order to set filter, I used 2 functions.

*int pcap_compile(pcap_t *p, struct bpf_program *fp, char *str, int optimize, bpf_u_int32 netmask)*

*int pcap_setfilter(pcap_t *p, struct bpf_program *fp)*

First, compile the filter program. I used filter expression “port 80”, not optimized.

```
struct bpf_program fp;          // The compiled filter expression
char filter_exp[] = "port 80";  // The filter expression set to HTTP
```

5. Start sniffing

```
/* start sniffing, until ending condition occurs */
pcap_loop(handle, 0, callback, NULL);
```

Start sniffing using *int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user*) function.
I used 0 to loop until ending condition occurs. Defined callback function for each sniffing event.
u_char *user is mostly set as NULL. I do not need to send any other u_char * pointer to callback function.

Callback function is used to analyze the sniffed packet

```
//callback function for the sniffing event
void callback(u_char *args, const struct pcap_pkthdr *header, const u_char *packet){

    static int count = 1;                   // packet counter

    /* pointers to packet headers */
    const struct sniff_ethernet *ethernet;  // The ethernet header [1]
    const struct sniff_ip *ip;              // The IP header
    const struct sniff_tcp *tcp;            // The TCP header
    const char *payload;                    // Packet payload

    int size_ip;
    int size_tcp;
    int size_payload;

    int i;
    const u_char *ch;
```

The second argument *pcap_pkthder *header* contains information of the sniffed packet.

```
struct pcap_pkthdr {
                    struct timeval ts; /* time stamp */
                    bpf_u_int32 caplen; /* length of portion present */
                    bpf_u_int32 len; /* length this packet (off wire) */ };
```

6. Define Ethernet header

```
/* Ethernet addresses are 6 bytes */
#define ETHER_ADDR_LEN  6
#define SIZE_ETHERNET 14

    /* Ethernet header */
    struct sniff_ethernet {
        u_char ether_dhost[ETHER_ADDR_LEN]; /* Destination host address */
        u_char ether_shost[ETHER_ADDR_LEN]; /* Source host address */
        u_short ether_type; /* IP? ARP? RARP? etc */
    };
```

Define structure of ethernet header for typecating. It reads 6 bytes for destination host address, 6 bytes for source host address, 2 bytes for ethernet type.

By typecasting, I can get the ethernet header data from the sniffed packet.

```
 /* define Ethernet header */
 ethernet = (struct sniff_ethernet*)(packet);
```

7. Define IP Header offset and compute the size

```
/* IP header */
struct sniff_ip {
    u_char ip_vhl;      /* version << 4 | header length >> 2 */
    u_char ip_tos;      /* type of service */
    u_short ip_len;     /* total length */
    u_short ip_id;      /* identification */
    u_short ip_off;     /* fragment offset field */
#define IP_RF 0x8000        /* reserved fragment flag */
#define IP_DF 0x4000        /* don't fragment flag */
#define IP_MF 0x2000        /* more fragments flag */
#define IP_OFFMASK 0x1fff   /* mask for fragmenting bits */
    u_char ip_ttl;      /* time to live */
    u_char ip_p;        /* protocol */
    u_short ip_sum;     /* checksum */
    struct in_addr ip_src,ip_dst; /* source and dest address */
};
#define IP_HL(ip)       (((ip)->ip_vhl) & 0x0f)
#define IP_V(ip)        (((ip)->ip_vhl) >> 4)
```

Define structure of IP Header for typecasting. Each pointer matches the length of each component of the IP header structure.

```
/* define/compute IP header offset */
ip = (struct sniff_ip*)(packet + SIZE_ETHERNET);
size_ip = IP_HL(ip)*4;
if (size_ip < 20) {
    printf("   * Invalid IP header length: %u bytes\n", size_ip);
    return;
}
```

By typecasting, I can define IP header offset from the packet. Starting location of IP header in the packet data is calculated by address of packet pointer + size of ethernet header.

The size of IP header = value of IP header length filed(5 or 6) * 4 bytes. The value of IP header length field is stored in IP_HL(ip) (next 4 bit of version).

Check the validity of size of IP header. Minimum size of IP header is 5 rows * 4 byte = 20 bytes.

8. Define TCP header offset and compute the size

```
/* TCP header */
typedef u_int tcp_seq;

struct sniff_tcp {
    u_short th_sport;   /* source port */
    u_short th_dport;   /* destination port */
    tcp_seq th_seq;     /* sequence number */
    tcp_seq th_ack;     /* acknowledgement number */
    u_char th_offx2;    /* data offset, rsvd */
#define TH_OFF(th)  (((th)->th_offx2 & 0xf0) >> 4)
    u_char th_flags;
#define TH_FIN 0x01
#define TH_SYN 0x02
#define TH_RST 0x04
#define TH_PUSH 0x08
#define TH_ACK 0x10
#define TH_URG 0x20
#define TH_ECE 0x40
#define TH_CWR 0x80
#define TH_FLAGS (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
    u_short th_win;     /* window */
    u_short th_sum;     /* checksum */
    u_short th_urp;     /* urgent pointer */
```

Define TCP header structure. Each pointer is used for typecasting as well.

```
/* define/compute TCP header offset */
tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
size_tcp = TH_OFF(tcp)*4;
if (size_tcp < 20) {
    printf("   * Invalid TCP header length: %u bytes\n", size_tcp);
    return;
}
```

Starting location of TCP header is calculated by packet address + size of ethernet header + size of IP header. Size of TCP header = header length value * 4bytes. TH_OFF(tcp) has header length value(first 4 bits from data offset field)

Minimum size of TCP header is 5 rows * 4bytes = 20 bytes.

Check the validity of TCP header length.

9. Define TCP Payload offset and compute the size

```
/* define/compute TCP Payload (segment) offset */
payload = (u_char *)(packet + SIZE_ETHERNET + size_ip + size_tcp);

/* compute TCP Payload (segment) size */
size_payload = ntohs(ip->ip_len) - (size_ip + size_tcp);

/* analyze payload data */
ch = payload;
```

Starting point of payload is calculated by address of packet + size of ethernet header + size of IP header + size of TCP header.

Size of payload = total length of IP packet – size of IP header – size of TCP header

10. Print HTTP header

```
const u_char *ch;
```

```
  /* analyze payload data */
  ch = payload;

  /*check the HTTP Method */
  char *METHOD;
  if(*ch == 'H' && *++ch == 'T'){
      METHOD = "RESPONSE";
  }
  else{
      METHOD = "REQUEST";
  }

  ch = payload;
```

ch is a pointer for accessing Payload data.

First check whether the method is request or response. If first two character is “HT” assign RESPONSE, else assign REQUEST. After checking method

After it is finished, move the pointer to start of the Payload.

```
/* print the HTTP header*/
if (size_payload > 0) {
    /* print Packet Number, IP, PORT, Method  info*/
    printf("%d %s:%d %s:%d HTTP %s \n", count, inet_ntoa(ip->ip_src), ntohs(tcp->th_sport), inet_ntoa(ip->ip_dst), ntohs(tcp->th_dport), METHOD);

    count++;          // increase packet count
```

Print Packet Count, IP, TCP port, method.

I used inet_ntoa() function to convert IPv4 address into an ASCII string in Internet standard format, and used ntohs() to convert TCP port number from network byte order to host byte order.

```
/* print request line & header line*/
if(*ch=='}'){}     //'}' bug fix
else{

    for(i = 0; i < size_payload; i++) {
    if (isprint(*ch)){
        printf("%c", *ch);
    }
    else if(*ch=='\n'){
        printf("%c",*ch);

        if(*++ch=='\r'){  //check if it is the end of header, which is '/r/n/r/n'
            printf("\n");
            break;
        } else{
            ch--;
        }

    }
    ch++;
    }
```

There was some bug that ‘}’ appears.

I used isprint() function to ignore ‘\n’ or ‘\r’ or ‘/t’.

If it is end of the line print ‘\n’

If the next character is ‘\r’ then it is the end of the header which is ‘\r\n\r\n’, so callback function ends.

Callback function(from step 6 ~ 10) is called for every sniffing event.

11. Terminate when ending condition occurs

```
/* clean up */
pcap_freecode(&fp);
pcap_close(handle);

return 0;
```

```c
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>
#include <ctype.h>
```

These are header files I used.