Capturing Packets In Your C Program, With Libpcap

By **Pankaj Tanwar** - February 1, 2011 31

This article provides an introduction to libpcap, and shows, with examples of source code, how you can use it to create your own packet-sniffing programs.

All data on the network travels in the form of packets, which is the data unit for the network. To understand the data a packet contains, we need to understand the protocol hierarchy in the reference models. I recommend that if you don't know the ISO OSI (Open Systems Interconnection) Reference Model, you should read up on it. A good starting point is Wikipedia.

The network layer is where the term packet is used for the first time. Common protocols at this layer are IP (Internet Protocol), ICMP (Internet Control Message Protocol), IGMP (Internet Group Management Protocol) and IPsec (a protocol suite for securing IP). The transport layer's protocols include TCP

(Transmission Control Protocol), a connection-oriented protocol; UDP (User Datagram Protocol), a connection-less protocol; and SCTP (Stream Control Transmission Protocol), which has features of both TCP and UDP. The application layer has many protocols that are commonly used, like HTTP, FTP, IMAP, SMTP and more.

Capturing packets means collecting data being transmitted on the network. Every time a network card receives an Ethernet frame, it checks if its destination MAC address matches its own. If it does, it generates an interrupt request. The routine that handles this interrupt is the network card's driver; it copies the data from the card buffer to kernel space, then checks the ethertype field of the Ethernet header to determine the type of the packet, and passes it to the appropriate handler in the protocol stack. The data is passed up the layers until it reaches the user-space application, which consumes it.

When we are sniffing packets, the network driver also sends a copy of each received packet to the packet filter. To sniff packets, we will use libpcap, an open source library.

Understanding libpcap

libpcap is a platform-independent open source library to capture packets (the Windows version is winpcap). Famous sniffers like tcpdump and Wireshark make the use of this library.

To write our packet-capturing program, we need a network interface on which to listen. We can specify this device, or use a function which libpcap provides: `char *pcap_lookupdev(char *errbuf)` .

This returns a pointer to a string containing the name of the first network device suitable for packet capture; on error, it returns NULL (like other libpcap functions). The `errbuf` is a user-supplied buffer for storing an error message in case of an error — it is very useful for debugging your program. This buffer must be able to hold at least `PCAP_ERRBUF_SIZE` (currently 256) bytes.

Getting control of the Network Device

Next, we open the chosen network device using the function `pcap_t *pcap_open_live(const char *device, int snaplen, int promisc, int to_ms, char *errbuf)` . It returns an interface handler of type `pcap_t`, which other libpcap functions will use. The first argument is the network interface we want to open; the second is the maximum number of bytes to capture.

Setting it to a low value will be useful when we only want to grab packet headers. The Ethernet frame size is 1518 bytes. A value of 65535 will be enough to hold any packet from any network. The `promisc` flag indicates whether the network interface should be put into promiscuous mode or not. (In promiscuous mode, the NIC will pass all frames it receives to the CPU, instead of just those addressed to the NIC's MAC address. Read more on Wikipedia.)

The `to_ms` option tells the kernel to wait for a particular number of milliseconds before copying information from kernel space to user space. A value of zero will cause the read operation to wait until enough packets are collected. To save extra overhead in copying from kernel space to user space, we set this value according to the volume of network traffic.

Actual capture

Now, we need to start getting packets. Let's use `u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h)`. Here, `*p` is the pointer returned by `pcap_open_live();` the other argument is a pointer to a variable of type `struct pcap_pkthdr` in which the first packet that arrives is returned.

The function `int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user)` is used to collect the packets and process them. It will return when `cnt` number of packets have been captured. A callback function is used to handle captured packets (we need to define this callback function). To pass extra information to this function, we use the `*user` parameter, which is a pointer to a `u_char` variable (we will have to cast it ourselves, according to our needs in the callback function).

The callback function signature should be of the form: `void callback_function(u_char *arg, const struct pcap_pkthdr* pkthdr, const u_char* packet)`. The first argument is the `*user parameter` we passed to `pcap_loop()`; the next argument is a pointer to a structure that contains information about the captured packet. The structure of `struct pcap_pkthdr` is as follows (from `pcap.h`):

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

An alternative to `pcap_loop()` is `pcap_dispatch(pcap_t *p, int cnt, pcap_handler callback, u_char *user)`. The only difference is that it returns when the timeout specified in `pcap_open_live()` is exceeded.

Filtering traffic

Until now, we have been just getting all the packets coming to the interface. Now, we'll use a `pcap` function that allows us to filter the traffic coming to a specific port. We might use this to only process packets of a specific protocol, like ARP or FTP traffic, for example. First, we have to compile the filter using the following code:

```
int pcap_compile(pcap_t *p, struct bpf_program *fp, const char *str, int optimize, bpf_u_int
```

The first argument is the same as before; the second is a pointer that will store the compiled version of the filter. The next is the expression for the filter. This expression can be a protocol name like ARP, IP, TCP, UDP, etc. You can see a lot of sample expressions in the pcap-filter or tcpdump man pages, which should be installed on your system.

The next argument indicates whether to optimise or not (0 is false, 1 is true). Then comes the netmask of the network the filter applies to. The function returns -1 on error (if it detects an error in the expression).

After compiling, let's apply the filter using `int pcap_setfilter(pcap_t *p, struct bpf_program *fp)`. The second argument is the compiled version of the expression.

Finding IPv4 information

```
int pcap_lookupnet(const char *device, bpf_u_int32 *netp, bpf_u_int32 *maskp, char *errbuf)
```

We use this function to find the IPv4 network address and the netmask associated with the device. The address will be returned in `*netp` and the mask in `*mask`.

A small sniffer program

Now let's write a small sniffer program that will help us understand how pcap works. Let's name it `sniff.c`. It's a program from the pcap tutorials from tcpdump.org, by Martin Casado. First of all, let us make the necessary includes:

```
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h>
```

Next, let's write the callback function to process our captured packets. This function just prints out a running count of packets, as captured. Afterwards we'll write another callback function. The function is very clear, so there's no need to explain it.

```
void my_callback(u_char *args, const struct pcap_pkthdr* pkthdr, const u_char*
    packet)
{
    static int count = 1;
    fprintf(stdout, "%3d, ", count);
    fflush(stdout);
    count++;
}
```

Now comes the `main()` function. We can make use of the functions we learnt about earlier, in this function:

```
int main(int argc,char **argv)
{
    int i;
    char *dev;
    char errbuf[PCAP_ERRBUF_SIZE];
    pcap_t* descr;
    const u_char *packet;
    struct pcap_pkthdr hdr;
    struct ether_header *eptr;    /* net/ethernet.h */
    struct bpf_program fp;        /* hold compiled program */
    bpf_u_int32 maskp;            /* subnet mask */
    bpf_u_int32 netp;             /* ip */

    if(argc != 2){
        fprintf(stdout, "Usage: %s \"expression\"\n"
            ,argv[0]);
        return 0;
    }

    /* Now get a device */
    dev = pcap_lookupdev(errbuf);

    if(dev == NULL) {
        fprintf(stderr, "%s\n", errbuf);
        exit(1);
    }
        /* Get the network address and mask */
    pcap_lookupnet(dev, &netp, &maskp, errbuf);

    /* open device for reading in promiscuous mode */
    descr = pcap_open_live(dev, BUFSIZ, 1,-1, errbuf);
    if(descr == NULL) {
        printf("pcap_open_live(): %s\n", errbuf);
        exit(1);
    }

    /* Now we'll compile the filter expression*/
    if(pcap_compile(descr, &fp, argv[1], 0, netp) == -1) {
        fprintf(stderr, "Error calling pcap_compile\n");
        exit(1);
    }

    /* set the filter */
    if(pcap_setfilter(descr, &fp) == -1) {
        fprintf(stderr, "Error setting filter\n");
        exit(1);
    }

    /* loop for callback function */
    pcap_loop(descr, -1, my_callback, NULL);
    return 0;
}
```

Compile the program as follows, and run it as the root (necessary for permissions to execute in promiscuous mode):

```
$ gcc -lpcap sniff.c -o sniffer
# ./sniffer ip
```

Check the output in Figure 1.

Figure 1: Output of the program

As we have given `ip` as the expression, your screen will soon fill with the count of the number of IP packets. You can replace `ip` with any expression of your choice, like tcp, arp, etc — take a look at the tcpdump man pages.

Here's another callback function, which will display the contents of the packets accepted by your filter expression (it's already in `sniff.c`):

```
void another_callback(u_char *arg, const struct pcap_pkthdr* pkthdr,
        const u_char* packet)
{
    int i=0;
    static int count=0;

    printf("Packet Count: %d\n", ++count);    /* Number of Packets */
    printf("Recieved Packet Size: %d\n", pkthdr->len);    /* Length of header */
    printf("Payload:\n");                     /* And now the data */
    for(i=0;i<pkthdr->len;i++) {
        if(isprint(packet[i]))                /* Check if the packet data is printable *
```

```
            printf("%c ",packet[i]);          /* Print it */
        else
            printf(" . ",packet[i]);          /* If not print a . */
        if((i%16==0 && i!=0) || i==pkthdr->len-1)
            printf("\n");
    }
}
```

You can modify the `pcap_loop()` line in `main()`, which calls `my_callback()`, to call this callback function instead. Compile the changed program, and run it with the same expression as its argument. The output, as you can see in Figure 2, is the payload of IP packets.

Figure 2: Output displaying payload of packets

I think we should wrap up this introduction here. Do test and experiment with pcap, and see the power behind the best (and our favourite) sniffers out there: tcpdump and Wireshark.

TAGS C connection oriented protocol Ethernet ethernet frames Ethernet header Internet protocols IPv4 kernel space LFY February 2011 libpcap MAC addresses Martin Casado network packets packet sniffing SCTP stream control subnet mask TCP TCP/IP UDP

Suka 4

Previous article

Next

Advanced Nmap: Scanning Firewalls

Puppet Data Centre Automation Solution, Part 4: Classes & Modules

Pankaj Tanwar

The author is a FOSS enthusiast who loves rock music and C.

RELATED ARTICLES

Stay away from SQL Injections

August 13, 2021

Allstar Aims To Fix Vulnerabilities in Open Source Projects

August 13, 2021

Tech Giants Team Up To Drive Open Source eBPF Projects

August 12, 2021

31 COMMENTS

Farah Chaudhary March 8, 2012 At 11:35 PM

useful link! thanks! :)

Reply

Farah Chaudhary March 8, 2012 At 11:35 PM

useful link! thanks! :)

Reply

Farah Chaudhary March 8, 2012 At 11:35 PM

useful link! thanks! :)

Reply

RAJA March 29, 2012 At 2:43 PM

Superb explanatin....

Reply

Chandramohan July 10, 2012 At 4:40 PM

Hi All,

I am trying to sniff the packets between machine A with IP 10.0.0.1 and Machine B with IP 10.0.0.2 in Machine C with IP 10.0.0.3
the machine C is port mirrored with Machine A
So using the above code I am able to sniff the Packet going from A to B

but NOT ABLE TO SNIFF from Machine B to A
I ran tcpdump in Machine C at same time and it is able to capture all
the packets from Machine B to A but why above sniffer code is able to
catch Packets from Machine B to A……???
Please help………..
note : I am running above code in Machine C and interface eth1 is port mirrored with Machine A and in the code also I have modified to sniff on eth1

Reply

tux July 10, 2012 At 5:09 PM

printf the info returned by pcap_open_live and also post it here

Reply

Chandramohan July 10, 2012 At 7:32 PM

```
descr->fd = 3
descr->selectable_fd = 3
descr->snapshot = 8192
descr->linktype = 1
descr->tzoff = 0
descr->activated = 1
descr->oldstyle = 1
descr->break_loop = 0
descr->bufsize = 8256
descr->buffer =
descr->bp = (null)
descr->cc = 254
```

Reply

Chandramohan July 10, 2012 At 7:37 PM

descr->direction = 0

Reply

Chandramohan July 10, 2012 At 8:02 PM

descr->send_fd = -1

tux July 11, 2012 At 1:53 PM

you forgot to print the *dev

kodem naresh August 7, 2012 At 1:09 PM

hai iam trying to interface ethernet with lpc2378 microcontroller ,so give a suggestions for that send the links what you know?

Reply

vinoth September 3, 2012 At 12:16 PM

Help me on libpcap packet capturing….

Reply

singularity September 4, 2012 At 11:59 AM

for particular type of packet we have set the filter in pcap_compile look into it

Reply

born2learn December 13, 2012 At 11:14 AM

i hav compiled above program in terminal..it gives error "sniff.c:1:18: error: pcap.h: No such file or directory" What should I do?Is there any packages need to install? I am using RHEL 5.0

Reply

tux December 13, 2012 At 11:38 AM

check that you have installed libpcap and use $ gcc -lpcap program.c while compiling

Reply

jc October 31, 2013 At 1:18 PM

Hi,

I compiled the program and when I run it, it says "no suitable device found".

May I know how can I solve this problem?

Thanks for the informative writeup.

Reply

Pankaj Tanwar December 22, 2013 At 2:13 PM

Please check if you are root user or not. To set the device into promiscuous mode you need to be root.

Reply

Jayanth December 22, 2013 At 1:36 PM

Hi,
I compiled and ran the program I too get the error "No suitable device found"….May i know how to overcome this

Reply

Pankaj Tanwar December 22, 2013 At 2:13 PM

you need to be root to set your device into promiscuous mode.

Reply

Jayanth December 22, 2013 At 4:48 PM

thanks for ur quick reply man…but now am not able to see the output u r getting..i mean the callback function is not getting called at all…what should i do now…??…

Reply

Pankaj Tanwar December 22, 2013 At 9:07 PM

you need to be more specific on the problem you are having. have you tried printing some text.. is there any activity on the network. I mean are you getting any packet.

Reply

Jayanth December 22, 2013 At 10:08 PM

Yes i included printf statements after each function execution….nd everything was being displayed until setfilter function…after executing setfilter function i had a printf which was "callback func to be called" which too was printed….inside the callback func i had a printf initially which never got printed….that is y i told the callback func was never called…nd btw will the callback be executed only if ter is packet in flow…???…nd also how to check whether i am receiving the packets or not…???…am using ubuntu…nd thanks again….

Pankaj Tanwar December 23, 2013 At 11:25 PM well while you are on network you are continuously receiving packets.. also try another program tcpdump to check you are receiving packets anyway.. it should be working..

Jayanth December 24, 2013 At 11:00 PM

Jus found out that the eth0 network interface is not receiving or sending any packets….so removed pacp_lookupdev and hard coded dev to point to "wlan0" interface…now am able to receive packets…nd get the output…nd once again thanks for ur help….

ash_2012 March 4, 2015 At 12:00 PM

how could we direct the program to sniff packets for a particular interface.I tried to capture the ip packets but program is sniffing on interface on which there is no traffic flow.could you please help me

Reply

Pankaj Tanwar March 4, 2015 At 10:57 PM

/* Now get a device */
dev = pcap_lookupdev(errbuf);

you can set dev yourself to according to your requirements like dev="wlan0" just for example. you should know the interfaces or get them in program

Reply

nikhitha March 11, 2015 At 12:36 PM

how to implement the raw sockets like packet capturing and filtering using c or c++

Reply

Pankaj Tanwar March 11, 2015 At 9:10 PM

This is quite raw we are taking ethernet packets if you want rawer check out libpcap source. Socket API is on transport layer so we already are rawer here.

Reply

Vivalldi April 29, 2015 At 7:05 PM

How do I filter packets coming from a single IP?

Reply

Abhishek Singh February 20, 2016 At 3:20 PM

How can I just capture all wifi beacon frames sent and extract out their ssid?

Reply

Djitan July 17, 2017 At 6:32 PM

Hey, I got the following error with the second callback function :

In function 'void another_callback(u_char*, const pcap_pkthdr*, const u_char*)':

warning: too many arguments for format [-Wformat-extra-args]
printf(" . ",packet[i]); /* If not print a . */

Someone got an answer to this error ?

Reply

LEAVE A REPLY

Comment:

Name:*

Email:*

Website:

☐ Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

ABOUT US