PACKET SNIFFER IMPLEMENTATION

Using PCAP library functions

Project Report

Computer Networks

CS425: (2014-2015 1st Semester)

Submitted by Group 16

Members:

Aayush Mudgal (12008)

Chetan Dalal (11218)

Dheeraj Aggarwal (10327239)

Sidharth Gupta (11714)

Installation Instruction:

- Install using MAKEFILE
- Or through command line instructions : $gcc packet_sniffer.c –o packet_sniffer –lpcap
- Command for running: sudo ./packet_sniffer

Packet Sniffer

Packet sniffers work by intercepting and logging network traffic that they 'see' via the wired or wireless network interface that it has access to on the host computer. On a wired network, what can be captured depends on the structure of the network. It might be able to see traffic on an entire network or only a certain segment of it, depending on the configuration and placement of network switches. On wireless networks, packet sniffers can usually only capture one channel at a time unless the host computer has multiple wireless interfaces that allow for multichannel capture. Our implementation of Packet Sniffer is in C based on the PCAP library.

Overview

Our implementation of the Packet Sniffer is coded in C and uses the PCAP library functions for interception of network traffic. PCAP (Packet CAPture) consists of an application programming interface (API) for capturing network traffic. UNIX-like systems implement pcap in the libpcap library, whereas Windows uses a port of libpcap known as WinPcap. All packets, even those destined for other, are accessible through this mechanism.

The pcap API is written in C, other languages such as JAVA, .NET languages, and scripting languages generally use a wrapper; no such wrappers are provided by libpcap or WinPcap itself. C++ programs can link directly to the C API or use an object-oriented wrapper. Libpcap was originally developed by the tcpdump developers in the Network Research Group at Lawrence Berkeley Laboratory. The low-level packet capture, capture file reading, and capture file writing code of tcpdump was extracted and made into a library, with which tcpdump was linked.

Features of the implemented Packet Sniffer

- Pcap library functions are used to capture packets from a device. The user can chose the desired device from a list of displayed devices.
- The user can also prefer sniffing of packets over a particular IP or for all IP.
- He also is provided with the choice of choosing a particular type of header structure if he desires. Available header structures by the tool are : Ethernet, IP, TCP, UDP, ICMP
- The user can store the log in different files through multiple instantiations of the tool
- On exit, the tool provides with a detailed statistics, this can enable the user to study the packet flow across the network.

API Functions Used

Some important API features used:

- **pcap_findalldevs:**

Synopsis:

char errbuf[PCAP_ERRBUF_SIZE];

int pcap_findalldevs(pcap_if_t **alldevsp, char *errbuf);
void pcap_freealldevs(pcap_if_t *alldevs);

It constructs a list of network devices that can be opened with pcapwith pcap_create() and pcap_activate() or with pcap_open_live(). (Those process that do not have sufficient privileges to open them for capturing will not appear on the list.)
If pcap_findalldevs() succeeds, the pointer pointed to by *alldevsp* is set to point to the first element of the list, or to NULL if no devices were found (this is considered success).

Each element of the list is of type pcap_if_t, and has the following members:

next
if not NULL, a pointer to the next element in the list; NULL for the last element of the list
name
a pointer to a string giving a name for the device to pass to pcap_open_live()
description
if not NULL, a pointer to a string giving a human-readable description of the device
addresses
a pointer to the first element of a list of network addresses for the device, or NULL if the device has no addresses
flags
device flags:
PCAP_IF_LOOPBACK
set if the device is a loopback interface
PCAP_IF_UP
set if the device is up
PCAP_IF_RUNNING
set if the device is running

Usage in the Packet Sniffer Implementation:

```
printf("Welcome to GROUP-16 PACKET SNIFFER\n");
if( pcap_findalldevs( &all_devices , error_buffer) )
{
    printf("ERROR: Devices not found : %s" , error_buffer);
    exit(1);
}
```

- **pcap_lookupnet()**

  char errbuf[PCAP_ERRBUF_SIZE];

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

pcap_lookupnet() is used to determine the IPv4 network number and mask associated with the network device device. Both netp and maskp are bpf_u_int32 pointers. It returns 0 on success and -1 on failure. If -1 is returned, errbuf is filled in with an appropriate error message. errbuf is assumed to be able to hold at least **PCAP_ERRBUF_SIZE** chars.

Usage in the Packet Sniffer Implementation:

```
//looks up the device for network number and subnet mask
if (pcap_lookupnet(device, &net_num, &subnet_mask, error_buffer))
{
    printf("ERROR: Could not determine IP address and subnet mask of selected device");
    return 0;
}
```

- **pcap_open_live()**

  char errbuf[PCAP_ERRBUF_SIZE];

  pcap_open_live(3) **char *device, int snaplen,**
  int promisc, int to_ms, char *errbuf);

It is used to obtain a packet capture handle to look at the packets on the network. Device is a string that specifies the network device to open. On linux systems with 2.2 or later kernels, a device argument of 'any' or NULL can be used to capture packets from all devices. Other arguments are:

- snaplen: specifies the snapshot length to be set on the handle.
- Promisc: specifies if the interface is to be put into promiscuous mode
- To_ms specifieds the read timeout in milliseconds

pcap_open_live returns a pacp_t * on success and NULL on failure. If NULL is returned, error_buffer is filled with an appropriate error message. Errbuf may also be set to warning text when it succeeds.

Usage in the Packet Sniffer Implementation:

```
//open a live device and binds it to the handle descriptor
descriptor = pcap_open_live(device,BUFSIZ, 1, 0, error_buffer);
if(descriptor==NULL)
{
    printf("ERROR: Could not open device %s\n", device);
    return 0;
}
```

- **Pcap_loop()**

typedef void(*pcap_handler)(u_char *user, const struct pcap_pkthdr *h, const u_char *bytes);

pcap_loop(3) ***p, int cnt, pcap_handler callback, u_char *user);**
int pcap_dispatch(pcap_t *p, int cnt, pcap_handler callback, u_char *user);

pcap_loop() processes packets from a live capture or '' savefile'' until cnt packets are processed, the end of the ''savefile'' is reached when reading from one, pcap_breakloop() is called or an error occurs. It does not return when live read timeouts occurs. A value of -1 or for cnt is equivalent to infinity so that packets are processed until another ending condition occurs.

- P pints to a packet capture descriptor returned from the pcap_open_live subroutine.
- Cnt is the number of packets to be processed
- Callback points to a user-subroutine that is called for each packet received.
- User specifies the first of the three arguments to be passed into the callback routine
- Callback is of type typedef void(*pcap_handler)(u_char *arg, const struct pcap_pkthdr *, const u_char *);

Return Value: The function returns 0 when cnt is exhausted. Or -1 if an error occurs. It doesn't return when live read timeouts occur; instead it attempts to read more packets.

Usage in the code:

```
//go in an infinte loop and execute packet_receive function for sniffing
pcap_loop(descriptor, no, packet_receive, NULL);
```

PACKETS

1. ETHERNET

A data packet on an Ethernet link is called an *Ethernet packet*, which transports an **Ethernet frame** as payload.

An Ethernet frame is preceded by a preamble and start frame delimiter (SFD), which are both part of the layer 1 (physical layer) Ethernet packet. Each Ethernet frame starts with an Ethernet header, which contains destination and source MAC addresses as its first two fields. The middle section of the frame is payload data including any headers for other protocols (for example Internet Protocol) carried in the frame. The frame ends with a frame check sequence (FCS), which is a 32-bit cyclic redundancy check used to detect any in-transit corruption of data.

802.3 Ethernet packet and frame structure

Layer	Preamble	Start of frame delimiter	MAC destination	MAC source	802.1Q tag (optional)	Ethertype (Ethernet II) or length (IEEE 802.3)	Payload	Frame check sequence (32-bit CRC)	Interpacket gap
	7 octets	1 octet	6 octets	6 octets	(4 octets)	2 octets	46(42)[b]–1500 octets	4 octets	12 octets
Layer 2 Ethernet frame			← 64–1518(1522) octets →						
Layer 1 Ethernet packet	← 72–1526(1530) octets →								

Source: Wikipedia

Ethernet Structure as defined:

```
//structure of ethernet header
struct  ethernet_header {
    u_char  ethernet_dhost[ETHER_ADDR_LEN];     //destination host address
    u_char  ethernet_shost[ETHER_ADDR_LEN];     //source host address
    u_short ethernet_type;                      //type of packet
};
```

2. Internet Protocol (IP)

The **Internet Protocol** (**IP**) is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. Its routing function enables internetworking, and essentially establishes the Internet.

IP, as the primary protocol in the Internet layer of the Internet protocol suite, has the task of delivering packets from the source host to the destination host solely based on the IP addresses in the packet headers. For this purpose, IP defines packet structures that encapsulate the data to be delivered. It also defines addressing methods that are used to label the datagram with source and destination information.

The first major version of IP, Internet Protocol Version 4 (IPv4), is the dominant protocol of the Internet. Its successor is Internet Protocol Version 6 (IPv6).

IPv4 header structure:

IPv4 Header Format

<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td>0</td><td colspan="4">Version</td><td colspan="4">IHL</td><td colspan="6">DSCP</td><td colspan="2">ECN</td><td colspan="16">Total Length</td></tr>
<tr><td>4</td><td>32</td><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment Offset</td></tr>
<tr><td>8</td><td>64</td><td colspan="8">Time To Live</td><td colspan="8">Protocol</td><td colspan="16">Header Checksum</td></tr>
<tr><td>12</td><td>96</td><td colspan="32">Source IP Address</td></tr>
<tr><td>16</td><td>128</td><td colspan="32">Destination IP Address</td></tr>
<tr><td>20</td><td>160</td><td colspan="32">Options (if IHL > 5)</td></tr>
</table>

Source: Wikipedia

IP header structure in the code:

```
//structure of ip header
struct ip_header {
    u_char version;                 //version<<4 | header length >> 2
    u_char TOS;                     //type of service
    u_char headerLen;                   //header length
    u_char ID;                      //identification
    u_char fragOffset;                      //fragment offset
    #define IP_RF 0x8000                //reserved fraagment
    #define IP_DF 0x4000                //don't fragment
    #define IP_MF 0x2000                //more fragments
    #define IP_OFFMASK 0x1fff           //mask for fragmenting bits
    u_char TTL;                     //time to live
    u_char protocol;                        //protocol
    u_char checksum;                    //checksum
    struct in_addr source, destination; //source and destination addresses
};
```

3. Transmission Control Protocol (TCP)

TCP provides reliable, ordered and error-checked delivery of a stream of octets between programs running on computers connected to a local area network, intranet or the public Internet. It resides at the transport layer.

Web browsers use TCP when they connect to servers on the World Wide Web, and it is used to deliver email and transfer files from one location to another. HTTP, HTTPS, SMTP, POP3, IMAP, SSH, FTP, Telnet and a variety of other protocols are typically encapsulated in TCP.

Applications that do not require the reliability of a TCP connection may instead use the connectionless User Datagram Protocol (UDP), which emphasizes low-overhead operation and reduced latency rather than error checking and delivery validation.

TCP Header Structure:

TCP Header

<table>
<tr><th colspan="2">Offsets Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td>0</td><td colspan="16">Source port</td><td colspan="16">Destination port</td></tr>
<tr><td>4</td><td>32</td><td colspan="32">Sequence number</td></tr>
<tr><td>8</td><td>64</td><td colspan="32">Acknowledgment number (if ACK set)</td></tr>
<tr><td>12</td><td>96</td><td colspan="4">Data offset</td><td colspan="3">Reserved
0 0 0</td><td>N
S</td><td>C
W
R</td><td>E
C
E</td><td>U
R
G</td><td>A
C
K</td><td>P
S
H</td><td>R
S
T</td><td>S
Y
N</td><td>F
I
N</td><td colspan="16">Window Size</td></tr>
<tr><td>16</td><td>128</td><td colspan="16">Checksum</td><td colspan="16">Urgent pointer (if URG set)</td></tr>
<tr><td>20
...</td><td>160
...</td><td colspan="32">Options (if data offset > 5. Padded at the end with "0" bytes if necessary.)
...</td></tr>
</table>

Source: Wikipedia

TCP header structure in the code:

```
//tcp-header structure
struct tcp_header {
    u_short sourcePort;         //Source Port
    u_short dstPort;            //Destination Port
    u_int seqNo;                //Sequence Number
    u_int ackNo;                //Acknowledgement number
    u_char offset;                  //data offset, reserved bits
    u_char tcpFlags;                //tcp control bits and ecn options
    #define TCP_FIN  0x01
    #define TCP_SYN  0x02
    #define TCP_RST  0x04
    #define TCP_PUSH 0x08
    #define TCP_ACK  0x10
    #define TCP_URG  0x20
    #define TCP_ECE  0x40
    #define TCP_CWR  0x80
    #define TCP_FLAGS        (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
    u_short recvWin;                //window size
    u_short checksum;               //checksum
    u_short urgent;             //urgent pointer
};
```

4. **User Datagram Protocol (UDP)**

UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.

With UDP, computer applications can send messages, in this case referred to as *datagrams*, to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths.

UDP Header Structure:

UDP Header

Offsets Octet		0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

Source: Wikipedia

UDP header structure in the code:

```
//udp-header structure
struct udp_header {
    u_short sourcePort;         //source port
    u_short dstPort;            //destination port
    u_short headerLen;              //header length
    u_short checksum;               //check sum
};
```

5. ICMP (Internet Control Messaging Protocol)

It is used by network devices, like routers, to send error messages indicating, for example, that a requested service is not available or that a host or router could not be reached. ICMP can also be used to relay query messages. It is assigned protocol number 1. ICMP differs from transport protocols such as TCP and UDP in that it is not typically used to exchange data between systems, nor is it regularly employed by end-user network applications (with the exception of some diagnostic tools like ping and traceroute).

ICMP Header Structure:

ICMP Header Format

<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><th>0</th><th>0</th><td colspan="8">Type</td><td colspan="8">Code</td><td colspan="16">Checksum</td></tr>
<tr><th>4</th><th>32</th><td colspan="32">Rest of Header</td></tr>
</table>

Source: Wikipedia

ICMP header structure in the code:

```
//icmp-header structure
struct icmp_header {
    u_char type;                //icmp type
    u_char code;                //icmp code
    u_short checksum;               //icmp checksum
    u_int data;             //other data
};
```

References

- Wikipedia : www.wikipedia.org
- Man-Pages : http://www.manpagez.com/
- TCP and libpcap public repository: http://www.tcpdump.org/