Send Orders for Reprints to reprints@benthamscience.ae

The Open Cybernetics & Systemics Journal, 2014, *8*, 779-783

Open Access

Design of Network Protocol Analyzers Using WinPcap

Wang Huiran[1,*] and Ma Ruifang[2]

[1]*College of Computer Science, Xi'an Polytechnic University, Xi'an, Shaanxi 710048, China;* [2]*Software Engineering School, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China*

Abstract: This paper introduces two approaches to develop the network protocol analyzers, one of which is based on NDIS (Network Driver Interface Specification), the other is based on WinPcap. The second approach is discussed in details. This paper outlines the WinPcap architecture. Functions exported by WinPcap are classified in three types. On this classification, we can accurately configure the developing environment, *e.g.* define the preprocessors, set working directories. Three basic functions, *i.e.* pcap_ findalldevs_ex(), pcap_open(), and pcap_next_ex(), are interpreted thoroughly. In the end, a step-by-step example is given with its outcomes.

Keywords: Network protocol analyzers, network protocol, network traffic, winpcap.

1. INTRODUCTION

Network traffic is the stream of information flowing in networks. Its characteristics play an important role in networks. Understanding of network traffic is useful in network design and management [1]. The first step in understanding of traffic is capturing packets from the network [2]. There are several tools already in existence, *e.g.* TcpDump, WinDump, and Ethereal. However, we require a traffic capturer with special features to meet our practical needs sometimes. In this case, we have to develop a traffic capturer by ourselves.

This paper intends to introduce how to develop a traffic capturer/ monitor in details.

2. TWO BASIC METHODS FOR TRAFFIC CAPTURERS PROGRAMMING

Netware traffic capturers can be developed based on two architectures [3, 4]. One architecture is NDIS (Network Driver Interface Specification), and the other is WinPcap or Libpcap. NDIS is a product of Microsoft, whereas WinPcap was devised by Torino University of Italy.

2.1. Capturers based on NDIS

NDIS describes an interface by which one or more NIC (Network Interface Card) drivers communicate with one or more overlying protocol drivers and the operating system. NDIS is implemented in a library, NDIS.SYS. The library exports all NT Kernel-mode functions that can be used for NIC driver development. It also takes care of all the Operating system specific tasks and also maintains binding and state information about all underlying NIC drivers [3]. With NDIS, we can develop three types of drivers, *i.e.* network interface card (NIC) drivers, intermediate protocol drivers, and upper-level protocol drivers.

To aid development of the traffic capturer/monitor applications, Microsoft came up with the sample driver PACKET.SYS and PACKET32.DLL along with the Microsoft Windows NT Device Development Kit (DDK). Through PACKET32.DLL, a capture application communicates with NIC drivers [3] (as shown in Fig. **1**). This figure clearly depicts the way our application interacts with the PACKET.SYS driver. The capture application calls the functions in packet32. DLL, which in turn calls the entry points in the driver PACKET.SYS. The driver uses the NDIS.SYS exported functions to communicate with the Network Interface Card.

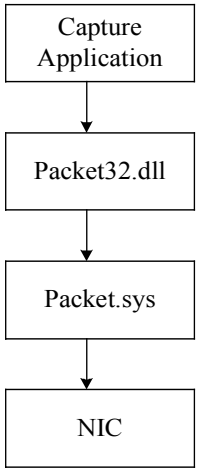

Fig. (1). NDIS architecture.

Packet32.dll exports a set of basic functions, such as PacketGetAdapterNames(), PacketOpen-Adapter(), PacketSetFilter(), and PacketReceivePacket(), with which we develop traffic capturers.

Based on NDIS, we can use its primitive functions and have a complete control over the adapters. Unfortunately, we lack of advanced functions to analyze and process the packets in this case.

1874-110X/14

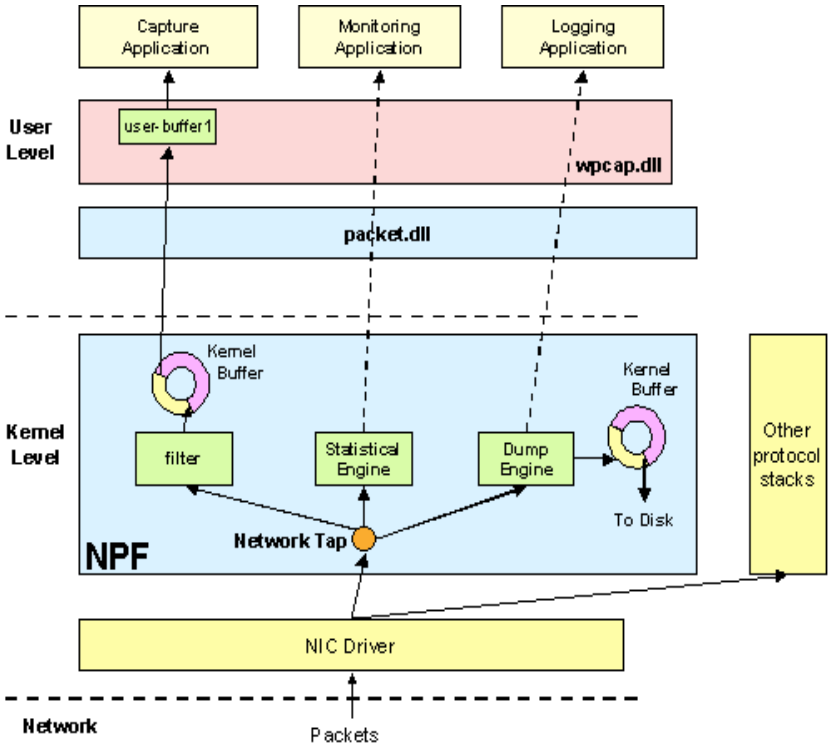

Fig. (2). NPF archtecture.

2.2. Capturers Based on WinPcap

WinPcap is architecture for packet capture or traffic monitor. It consists of three parts, Netgroup Packet Filter (NPF), packet.dll and wpcap.dll [4] (as shown in Fig. **2**).

The NPF runs inside the kernel of OS, interacts directly with the network interface drivers. It is system dependent. NPF bypasses the TCP/IP protocol stack in order to access the raw data transiting on the network. NPF is composed mainly of three components, *i.e.* filter, statistical engine and dump engine. They perform packet capture, traffic monitor and data dump operations, respectively.

The packet.dll offers a low-level API in order for user applications to directly access the functions of the NPF and to obtain a programming interface independent from the Microsoft OS.

The Wpcap.dll (lipcap.dll on Unix system) exports a more powerful set of high level capture functions, allowing to capture packets in a way independent from the underlying network hardware and operating system.

3. CAPTURE PROGRAMMING BASED ON WINPCAP

This section, we will discuss capture programming based on WinPcap in details.

3.1. Download and install WinPcap

Wpcap is free software. You can download it from its homepage http:// winpcap.polito.it. The latest version is WinPcap 3.1beta3. Following its guidance, you will install it successfully. Packet.dll and wpcap.dll will be installed under directory winnt\system32, and npf.sys under winnt\ system32\drivers, respectively.

Next, you should download the developing environment wpdpack from the same location as above. Wpdpack has the format of zip file. After unzipped, there are 4 subdirectories, *e.g.* lib, include, docs and examples under wpcaproot directory. Under subdirectory lib there are 2 libraries, packet.lib and wpcap.lib. Under subdirectory include are some header files. To aid programming, Wpdpack provides us with user manuals under subdirectory docs, and with examples under subdirectory examples.

3.2. Preparation for Programming

We divide functions exported by wpcap into three types, *i.e.* libpcap compatible, wpcap-specific and remote capture (as shown in Table **1**). You can know into which type a function of interest falls by reading the wpcap manual or by looking it up in these header files. Each type has header file and preprocessor of its own. In programming, you have to include pcap.h in the beginning of your source code, and set the corresponding preprocessor(s). Depending on how the preprocessors are set, the other two header files will be included by the system automatically.

Furthermore, the linker has to be set to include the wpcap.lib library file, as the wpcap is implemented in DLL. Also, the linker should be set to contain the wsock32.lib, which is located in directory Microsoft Visual Studio\VC98\Lib. Some functions in wsock32.lib may be invoked by wpcap.

In order for Microsoft Visual C++ to look for files successfully, we have to configure its directory option. The include file directory has to point to the directory under which the wpcap header files are located. The library file directory has to point to the directory which include wpcap.lib.

Table 1. Types of wpcap exported functions.

Function Type	Header File	Preprocessor	Example Functions
Libcap compatible	Pcap.h	WIN32	pcap_open_live pcap_setfilter
Wpcap specific	Win32-Extensions.h	WPCAP	pcap_live_dump pcap_offline_filter
Remote capture	remote-ext.h	HAVE_REMOTE	pcap_findalldevs_ex pcap_remoteact_list

Table **2** summarizes preparations for capturing programming.

Table 2. Preparations for capturing programming.

No.	Content
1	Include wpcap.h files in your source code
2	Define preprocessors WIN32, WPCAP, and optionally HAVE_REMOTE in the project settings dialog
3	Add wpcap.lib and wsock32.lib to the linker in the project settings dialog
4	Add *wpcaproot\include* under include file directory, and *wpcaproot\lib* under library file directory in the options dialog

3.3. Three basic Functions

To develop a traffic capturer successfully, we have to understand three basic functions, *i.e.* pcap_ findalldevs_ex(), pcap_open() and pcap_ next_ex().

Before capturing packets, we need to known how many adapters are installed in our machine, and what they are. To do this, we use function pcap_findalldevs_ex(), which creates a list of network interface adapters. As shown in Table **1**, this function belongs to the type of remote capture, and has the following format (in Wpcap 3.1beta3),

int pcap_findalldevs_ex(char *source, struct pcap_rmtauth *auth, pcap_if_t **alldevs, char *errbuf)

It takes four parameters, *i.e.* source, auth, alldevs, and errbuf. Parameter source is of character pointer type, and specifies the adapters location. It follows Source Specification Syntax, and may take one of the three formats as listed in Table **3**.

From the above table, we know that function pcap_findalldevs_ex() can find adapters in local machine, adapters in remote machine, and dumpfile in a given path.

The second parameter auth holds the information required to authenticate the connection to the remote host. It is not meaningful in case of a query to the local machine, and can be NULL.

The fourth parameter errbuf keeps error information of string type, which allow us to know what has happened if there is something wrong in the operation.

The third parameter alldevs is a pointer of structure pcap_if, and points to the first adapter of the adapter list.

Function pcap_findalldevs_ex() return '0', if the operation is successfully. Otherwise, it return '-1'.

After getting the list of network adapters, we can open one of them using function pcap_open(). Its prototype is：

pcap_t *pcap_open(const char *source, int snaplen, int flags, int read_timeout, struct pcap_rmtauth *auth, char *errbuf)

This function takes six parameters. Parameter source is a string containing name of the adapter to be opened.

Parameter snaplen defines the length of the portion of a packet, which will be sent to the traffic capturer or monitor. If snaplen equals , for example, to 150, only the first 150 bytes of the packet will be sent to the user in spite of this packet with a great length of more than 1500bytes, perhaps.

Parameter flags specifies the operation mode of this adapter. Flags with a value of PCAP_ OPENFLAG _PROMISCUOUS means that the adapter will be in promiscuous mode, which is the usual mode in packet capturing.

Parameter read_timeout specifies the time for reading packets. Even though a packet has been seen, the read opera-

Table 3. Parameter source of function pcap_findalldevs_ex.

Format	Example	Action by pcap_findalldevs_ex
rpcap://	rpcap://	to look for all adapters in the local machine
rpcap://host	rpcap://10.121.10.09	to look for all adapters in host 10.121.10.09
file://fold	file://d:\wcpt\dumpfile	to look for all of the pcap data files in the directory d:\ wcpt\ dumpfile

```
#include <pcap.h>

main(int argc, char **argv)
{
        // Define variables
        pcap_if_t *adapterlist, *d;
        pcap_t *adapterpointer;
        u_int adapterNo, i=0;
        char errbuf[PCAP_ERRBUF_SIZE];
        int scs;
        struct pcap_pkthdr *header;
        const u_char *pkt_data;
        struct tm *ltime;
        char timestr[16];

        // A. Retrieve the local device list
        if (pcap_findalldevs_ex("rcap://", NULL, &adapterlist, errbuf) !=0)
        {
                printf("Error in looking for adapters: %s\n", errbuf);
                return -1;
        }

        for(d=adapterlist; d; d=d->next)   /* Print the list */
                printf("\n %d. %s   ", ++i, d->name);

        // Select one adapter from the adapter list
        printf("\n\nSelect adapter by number (1-%d):",i);
        scanf("%d", &adapterNo);

        // Jump to the selected adapter
        for (d=adapterlist, i=0; i<adapterNo-1 ;d=d->next, i++);

        // B. Open the adapter
        adapterpointer=pcap_open(d->name,
                                        150       ,

                                        PCAP_OPENFLAG_PROMISCUOUS,
                                        50,

                                        NULL,
                                        errbuf);

        if (adapterpointer==NULL) return -1;

        // C. Read, convert and print packets,
        while((scs = pcap_next_ex(adapterpointer, &header, &pkt_data)) >= 0)
        {
                if(scs == 0) continue;  //Timeout elapsed

                // convert the timestamp to readable format
                ltime=localtime(&header->ts.tv_sec);
                strftime( timestr, sizeof timestr, "%H:%M:%S", ltime);

                // print timestamp and packet length
                printf("\n%s:%ld (%ld)\n", timestr, header->ts.tv_usec, header->len);

                for (i=1; (i <33) ; i++)
                {
                        printf("%.2x ", pkt_data[i-1]);
                        if ( (i % 16) == 0) printf("\n");
                };
        }
        return 0;
}
```

Fig. (3). Example capture.

tion does not return until the read_timeout elapses. In this manner, multiple packets may be read in a single operation.

Now that we have found and opened a adapter, we can read packets from it by function pcap_ next_ex(). This function has the following format,

int pcap_next_ex(pcap * p, struct pcap_pkthdr ** pkt_header, u_char ** pkt_data)

Parameter p is a pointer returned by function pcap_open(). Parameter pkt_data is a string holding the packet data received.

Parameter pcap_header is a structure pcap_pktdr pointer. Structure pcap_pktdr contains three members, *i.e.* ts, caplen and len. caplen is a integer denoting length of the packet data received. len is also an integer denoting length of the whole packet. ts is the timestamp of the packet and has two pointer members, tv_sec and tv_usec. tv_sec is the integer portion of the timestamp in seconds, whereas tv_usec is the fractural portion in microsecond.

3.4. An Example Capturer

To show how to develop a packet capturer clearly, we provide a step-by-step example (as shown in Fig. **3**).

Four steps are involved in development of packet capturers.

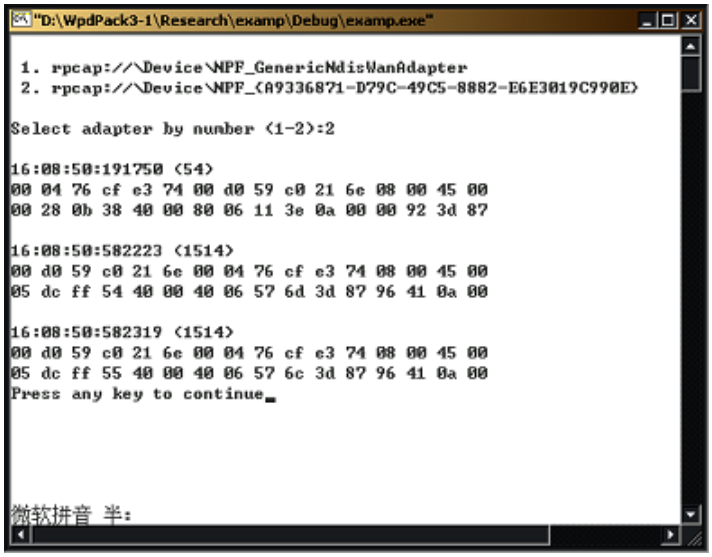

Fig. (4). Outputs of the example capturer.

Step 1. Install WinPcap (See paragraph 3.1).

Step 2. Configure Microsoft Visual C++ (See Table **2**)

Define preprocessors WIN32, WPCAP, HAVE_REMOTE.

Add wpcap.lib and wsock32.lib to the linker in the project settings dialog.

Add wpcaproot\include under include file directory, and wpcaproot\lib under library file directory in the options dialog

Step 3. Write the C source codes, and keep in mind to include pcap.h in the top of the codes.

Step 4. Build and run the capturer

When built and run, the example capturer prints results as shown in Fig. (**4**). As we see, it found 2 adapters. After selecting adapter No.2, it started capturing packets from this adapter and printing them out. First came the timestamp, following by packet length in parentheses, then the packet data (here only printed two lines to save space).

The result showed the example capturer operated well.

CONCLUSION

(1) Packet capturer can be developed based on NDIS, or on WinPcap.

(2) We divide functions exported by wpcap into three types, *i.e.* lipcap compatible, wpcap specific, and of remote capture.

(3) To develop captures successfully, one has to configure the developing environment correctly, *e.g.* to define the preprocessors, to set the include file directory and library file directory.

(4) Follow the example capturer given in this paper, it is easy for you to develop a capturer by yourself.

(5) Advanced functionalities, for example, performance analysis, traffic statistics and fault diagnosis, can be added conveniently to capturers.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

This paper was completed when I was visiting Kyoto University of Japan. I would like to thank Professor T. Takahashi and Doctor T. Asaka for their help and support.

REFERENCES

[1] De Souza, Erico; Matwin, Stan ; Fernandes, Stenio. Network traffic classification using AdaBoost Dynamic. 2013 IEEE International Conference on Communications Workshops, ICC 2013, pp. 1319-1324.

[2] Anderson, Keith F. A survey of techniques for improving the calibration of high-power network analyzers. 78th ARFTG Microwave Measurement Conference: High Power RF Measurement Techniques, ARFTG 2011.

[3] Hofstede, Rick; Drago, Idilio; Sperotto, Anna; Pras, Aiko. Flow monitoring experiences at the ethernet-layer. Lecture Notes in Computer Science , v6955, p 134-145, 2011.

[4] Loris Degioanni. WinPcap documentation 3.0. http://winpcap.polito.it. January 8, 2005.

Received: September 16, 2014 Revised: December 23, 2014 Accepted: December 31, 2014