WinPcap

Prof. Lin Weiguo

Sept 2013

Linux World

- libpcap :
  - http://www.tcpdump.org/
  - 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. It is now developed by the same tcpdump.org group that develops tcpdump.
- libnet :
  - http://sourceforge.net/projects/libnet-dev/
  - libnet provides a portable framework for low-level network packet construction.
- Ettercap:
  - http://ettercap.sourceforge.net/
  - Ettercap is a suite for man in the middle attacks on LAN. It features sniffing of live connections, content filtering on the fly and many other interesting tricks. It supports active and passive dissection of many protocols (even ciphered ones) and includes many feature for network and host analysis.

Windows World

- WinPcap
  - http://www.winpcap.org/
  - Windows Packet Capture Library; compatible with libpcap.
  - WinPcap started as a University project in 1998 within the Computer Networks Group (NetGroup) at the Politecnico di Torino(Polytechnic University of Turin, Italy).
  - 2005, CACE Technologies (Davis, CA), a company set up by some of the WinPcap developers, develops and maintains the product.
- Ethereal/WireShark
  - http://www.wireshark.org
  - Wireshark is a free packet sniffer computer application. It is used for network troubleshooting, analysis, software and communications protocol development, and education.
  - In June 2006 the project was renamed from Ethereal due to trademark issues. It was because Gerald Combs, creator of Ethereal, joined CACE Technologies

WinPcap

- WinPcap is the industry-standard tool for link-layer network access in Windows environments:
  - it allows applications to capture and transmit network packets bypassing the protocol stack,
  - and has additional useful features, including kernel-level packet filtering, a network statistics engine and support for remote packet capture.
- WinPcap consists of
  - a driver, that extends the operating system to provide low-level network access, and
  - a library that is used to easily access the low-level network layers. This library also contains the Windows version of the well known *libpcap* Unix API.

Get WinPcap Stable Version 4.1.2

- WinPcap auto-installer (driver +DLLs)
  - WinPcap_4_1_2.exe
  - Needed for WinPcap-based applications ready to work
- WinPcap Developer's Packs (.h + .lib +mannual & tutorial)
  - WpdPack_4_1_2.zip
  - Contains all the files needed to create WinPcap-based applications: libraries, include files, documentation and a complete set of example programs.
- Source Code Distributions
  - WpcapSrc_4_1_2.zip
  - the full WinPcap source code distribution. It includes the sources of *Wpcap.dll*, *packet.dll* and the drivers for the different Operating Systems.

WinPcap Developer Pack

- Unpack WpdPack_4_1_2.zip under your solution directory
- \mySolution\WpdPack

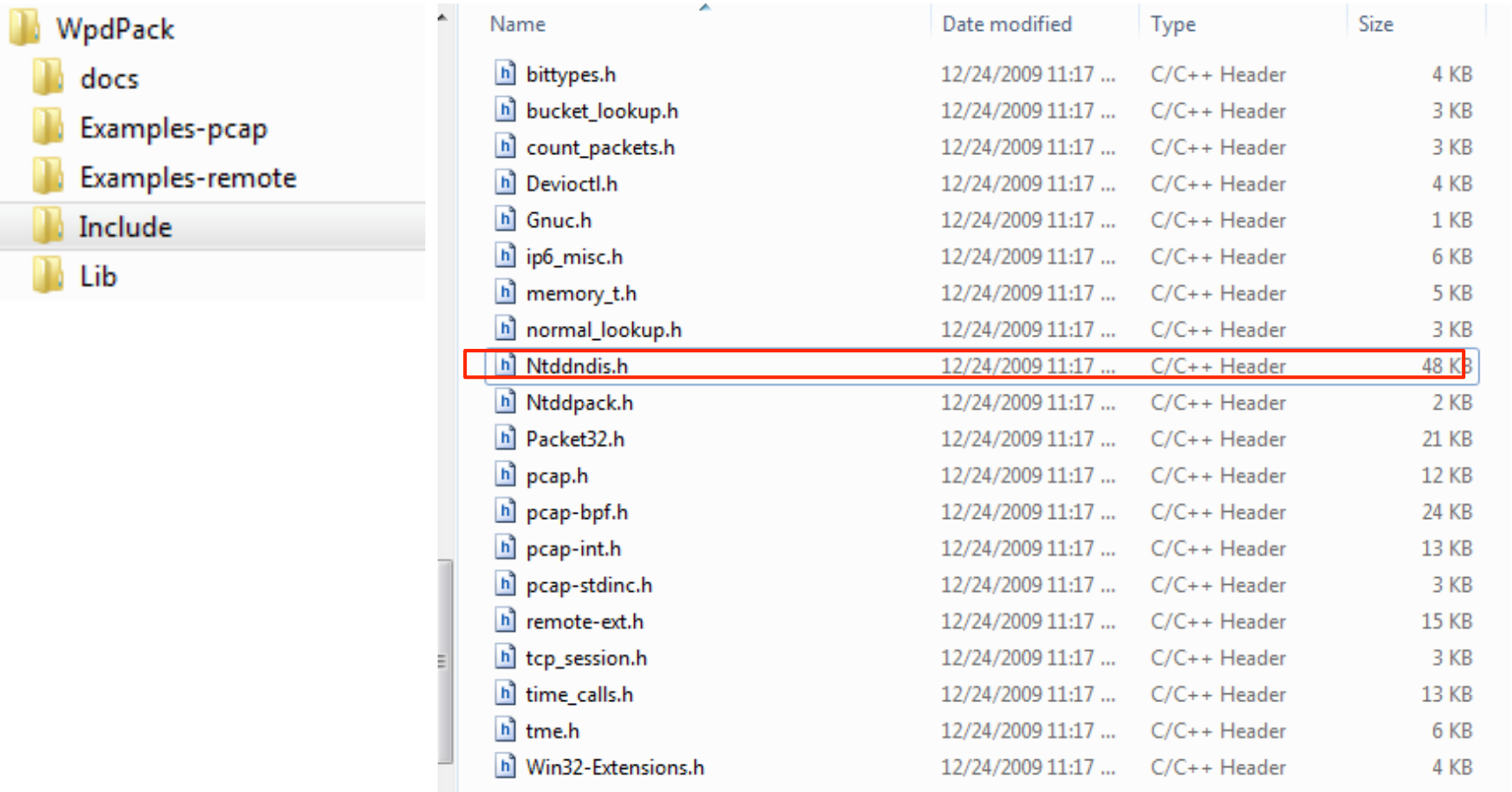
  .\docs

  .\Examples

  .\Include

  .\Lib

Include


WpdPack
docs
Examples-pcap
Examples-remote
Include
Lib
Name
Date modified
Type
Size
bittypes.h 12/24/2009 11:17 ... C/C++ Header 4 KB
bucket_lookup.h 12/24/2009 11:17 ... C/C++ Header 3 KB
count_packets.h 12/24/2009 11:17 ... C/C++ Header 3 KB
Devioctl.h 12/24/2009 11:17 ... C/C++ Header 4 KB
Gnuc.h 12/24/2009 11:17 ... C/C++ Header 1 KB
ip6_misc.h 12/24/2009 11:17 ... C/C++ Header 6 KB
memory_t.h 12/24/2009 11:17 ... C/C++ Header 5 KB
normal_lookup.h 12/24/2009 11:17 ... C/C++ Header 3 KB
Ntddndis.h 12/24/2009 11:17 ... C/C++ Header 48 KB
Ntddpack.h 12/24/2009 11:17 ... C/C++ Header 2 KB
Packet32.h 12/24/2009 11:17 ... C/C++ Header 21 KB
pcap.h 12/24/2009 11:17 ... C/C++ Header 12 KB
pcap-bpf.h 12/24/2009 11:17 ... C/C++ Header 24 KB
pcap-int.h 12/24/2009 11:17 ... C/C++ Header 13 KB
pcap-stdinc.h 12/24/2009 11:17 ... C/C++ Header 3 KB
remote-ext.h 12/24/2009 11:17 ... C/C++ Header 15 KB
tcp_session.h 12/24/2009 11:17 ... C/C++ Header 3 KB
time_calls.h 12/24/2009 11:17 ... C/C++ Header 13 KB
tme.h 12/24/2009 11:17 ... C/C++ Header 6 KB
Win32-Extensions.h 12/24/2009 11:17 ... C/C++ Header 4 KB

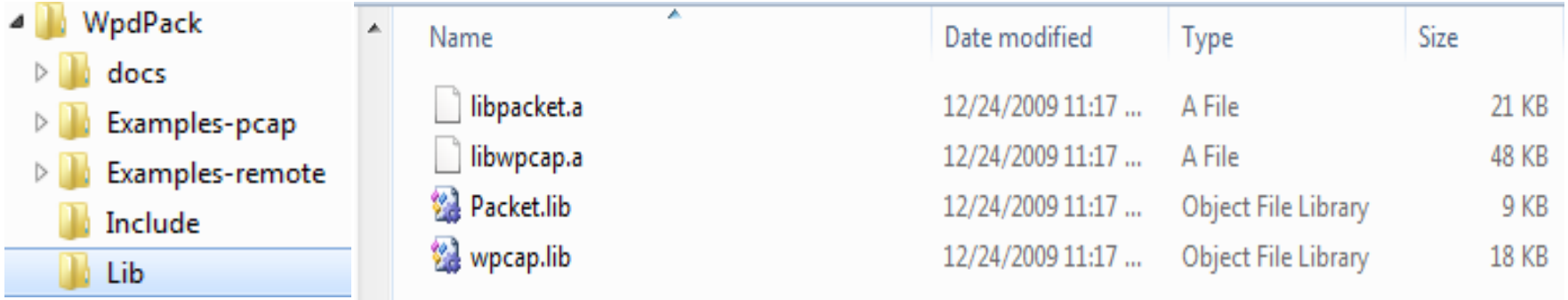
Build error with netioapi.h

- Problem:

C:\Program Files\Microsoft SDKs\Windows\v7.1\include\netioapi.h(155): error C2146: syntax error : missing ';' before identifier 'PhysicalMediumType'

- Reason:
  - "ntddndis.h" with Windows SDK v7.x is newer than the "ntddndis.h" file under "WpdPack\Include\"
- Solution
  - Modify "netioapi.h" to include local "ntddndis.h" file
    - #include <ntddndis.h> => #include "ntddndis.h"

lib

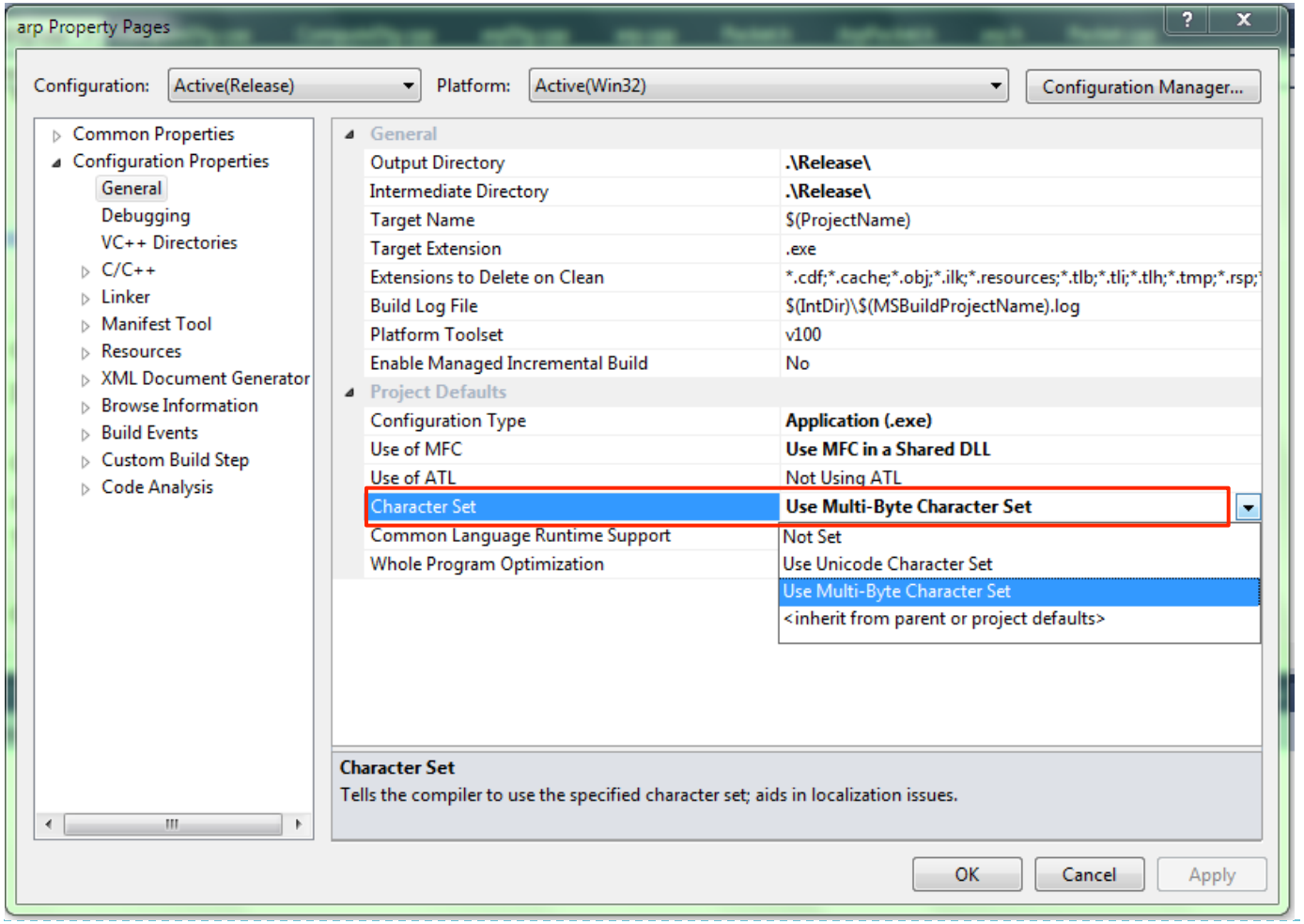

WpdPack
docs
Examples-pcap
Examples-remote
Include
Lib
Name
Date modified
Type
Size
libpacket.a
12/24/2009 11:17 ...
A File
21 KB
libwpcap.a
12/24/2009 11:17 ...
A File
48 KB
Packet.lib
12/24/2009 11:17 ...
Object File Library
9 KB
wpcap.lib
12/24/2009 11:17 ...
Object File Library
18 KB

Working with MS VS 2010

- Add *HAVE_REMOTE* among the preprocessor definitions.
- Add *WPCAP* among the preprocessor definitions (if needed).
- Include the file *pcap.h* at the beginning of every source file that uses the functions exported by library.
  - *pcap.h* can be found in the WinPcap developer's pack
- Set the options of the linker to include the *wpcap.lib* library file.
  - *wpcap.lib* can be found in the WinPcap developer's pack.
- Set the options of the linker to include the winsock library file *ws2_32.lib*.
  - This file is distributed with the VS 2010 and contains the socket functions for Windows.

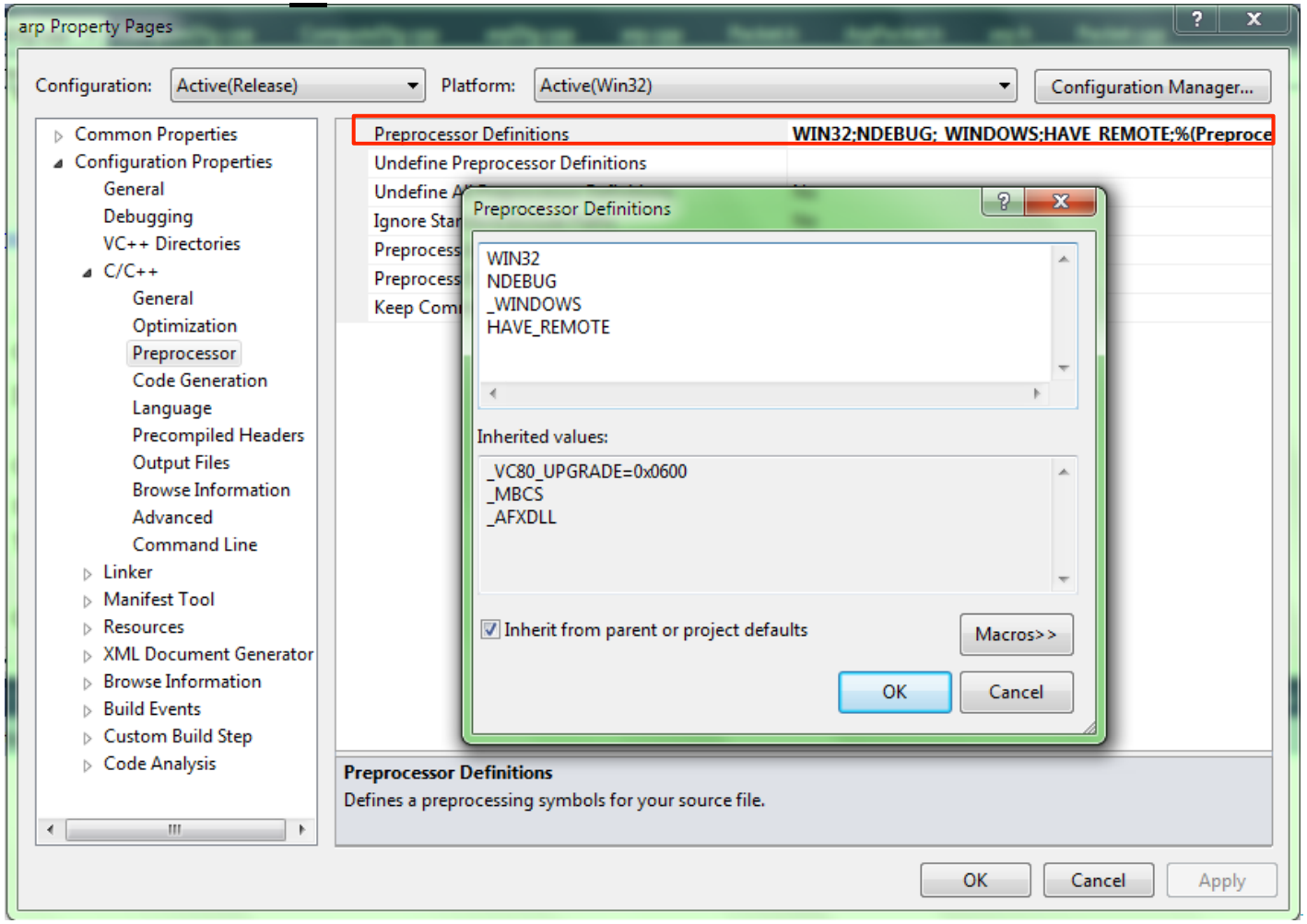
Project Configuration: General

- Choose Character Set



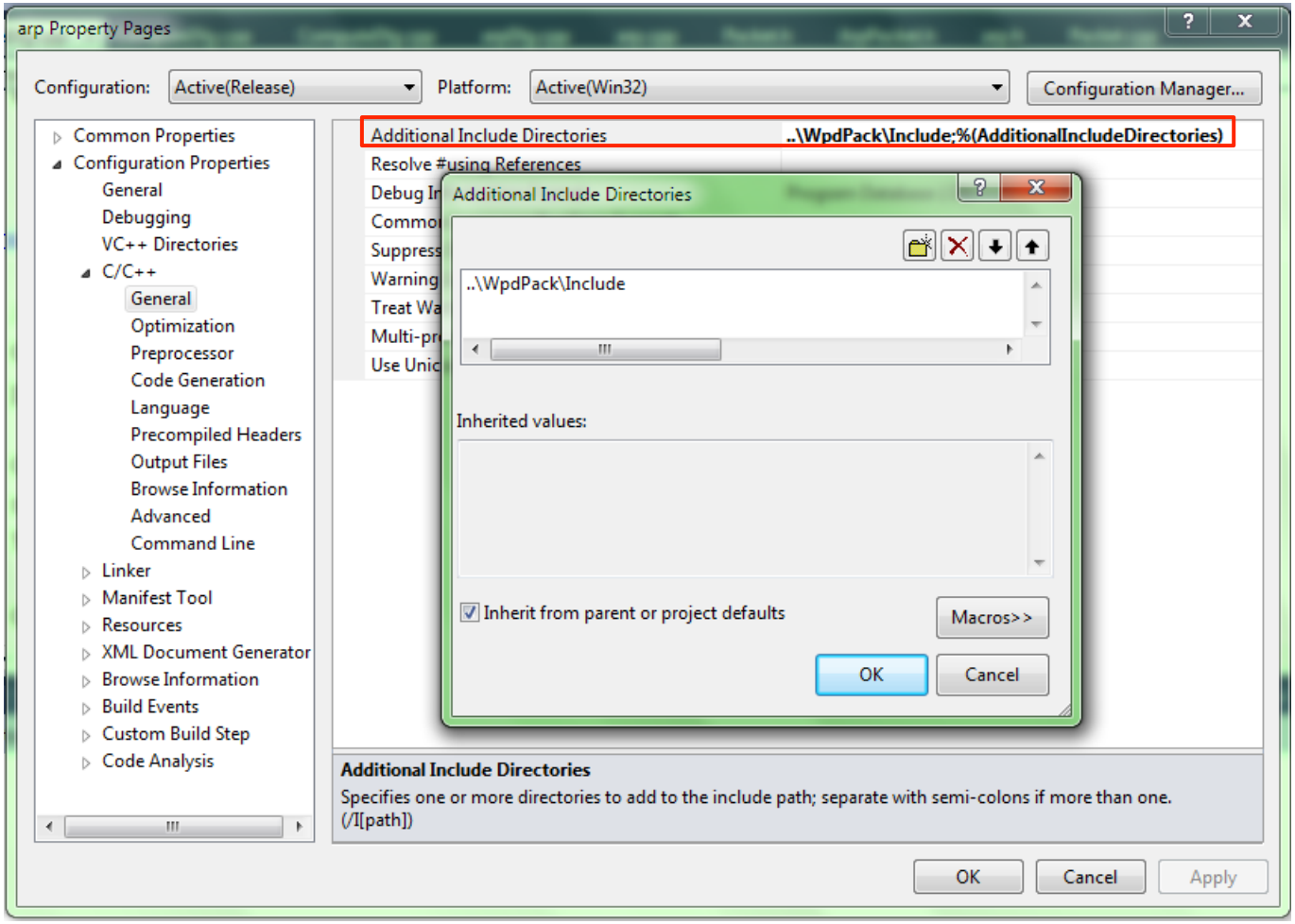
Project Configuration: C/C++ - Preprocessor

- Define HAVE_REMOTE



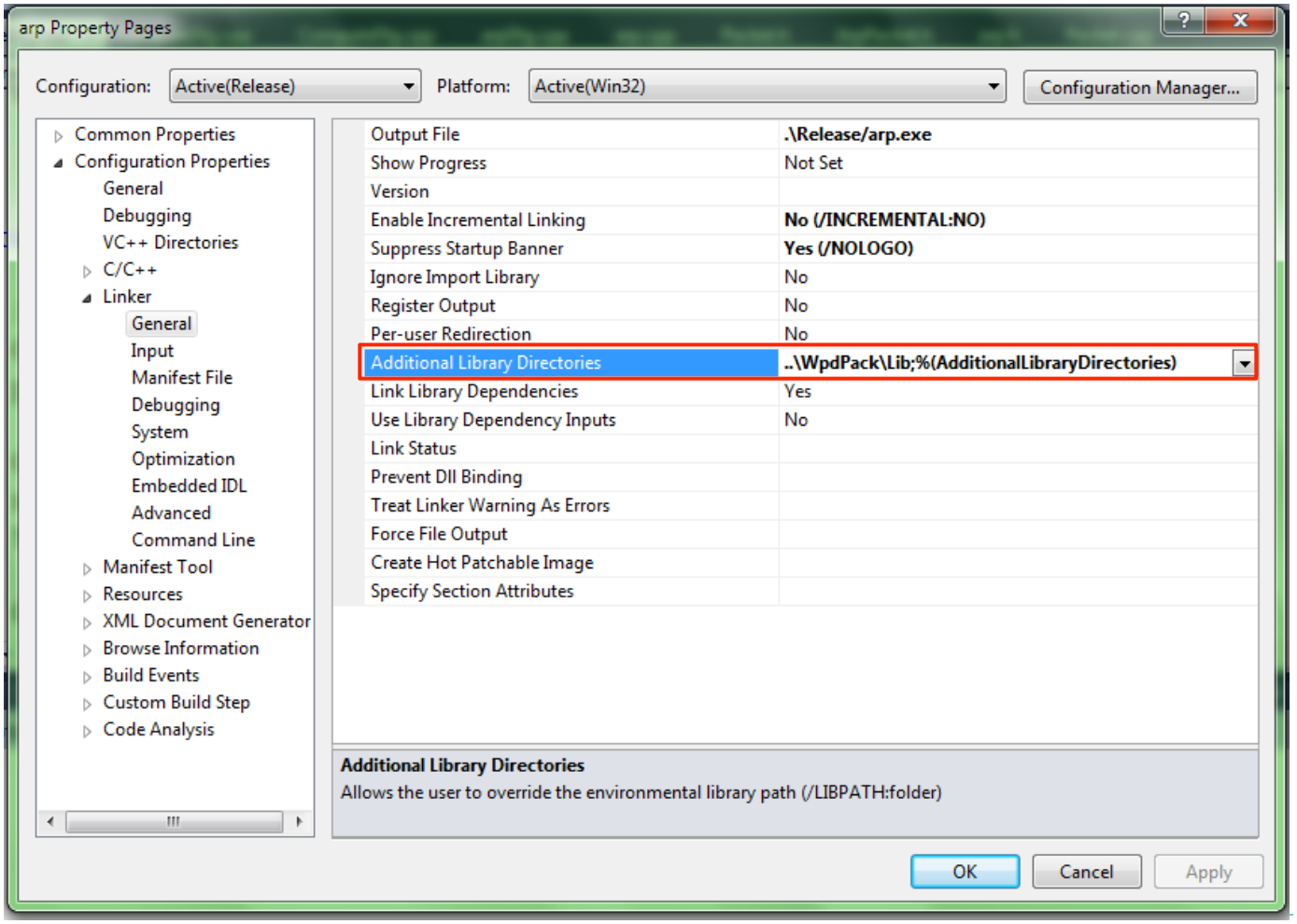
Project Configuration: C/C++ -General

- Add a new path where VS 2010 will look for include files



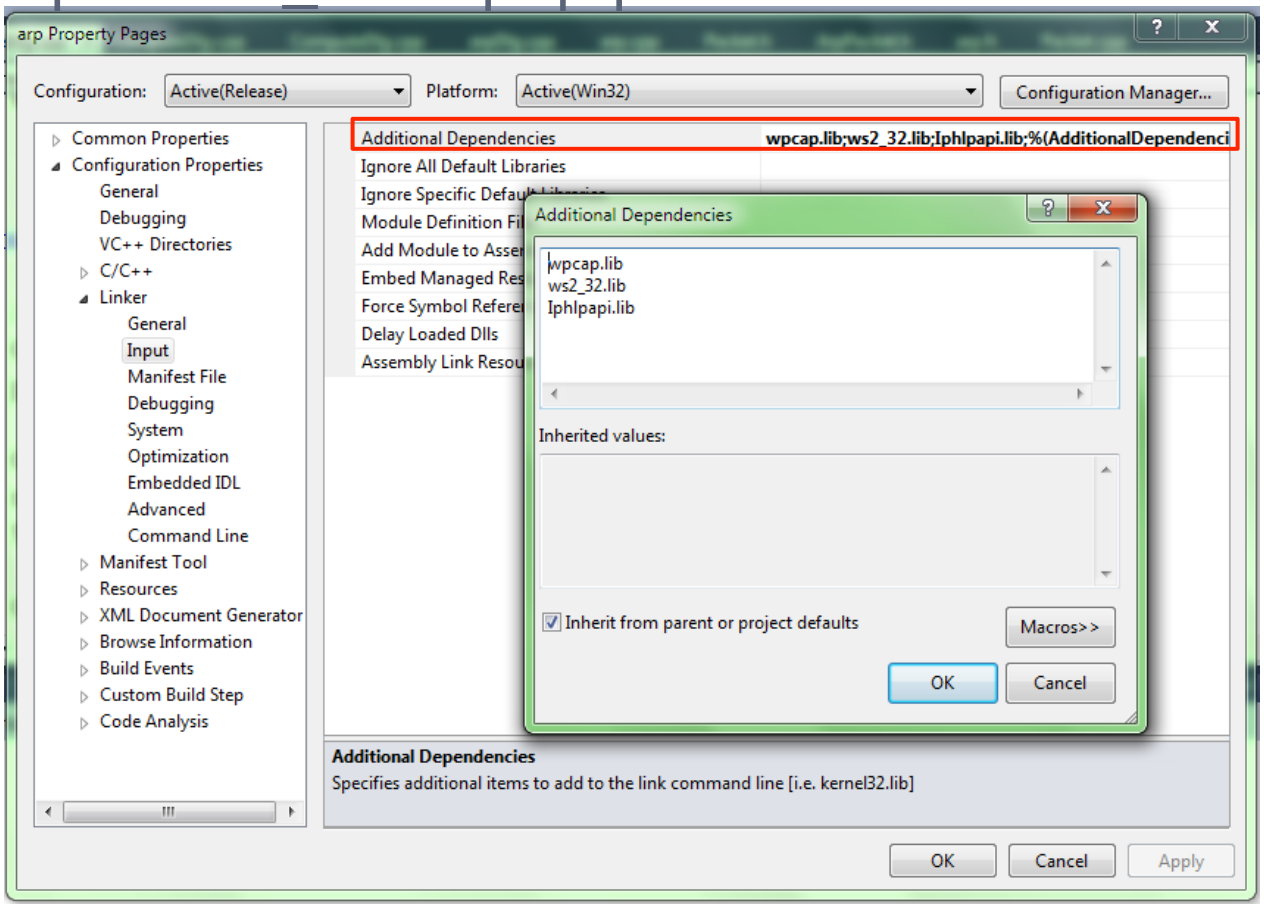
Project Configuration: General

- add a new path where VS 2010 will look for the libraries



Project Configuration: Input

- Add new libraries to the project
  - wpcap.lib ws2_32.lib Iphlpapi.lib



Alternative approach for .h and .lib

- #include <winsock.h> or //winsock 1.1 header
- #include <winsock2.h> //WinSock2 header

- #pragma comment(lib, "wsock32.lib") or //winsock 1.1 lib
- #pragma comment(lib, "ws2_32.lib") //WinSock2 library

- #define HAVE_REMOTE
- #include " pcap.h“
- #pragma comment(lib, " wpcap.lib")

#pragma pack directives

- #pragma pack([show] | [push | pop] [, identifier] , n)
  - *n* (optional)Specifies the value, in bytes, to be used for packing. The default value for *n* is 8. Valid values are 1, 2, 4, 8, and 16. The alignment of a member will be on a boundary that is either **a multiple of *n*** or **a multiple of the size of the member**, whichever is smaller.
  - Consider the following example,

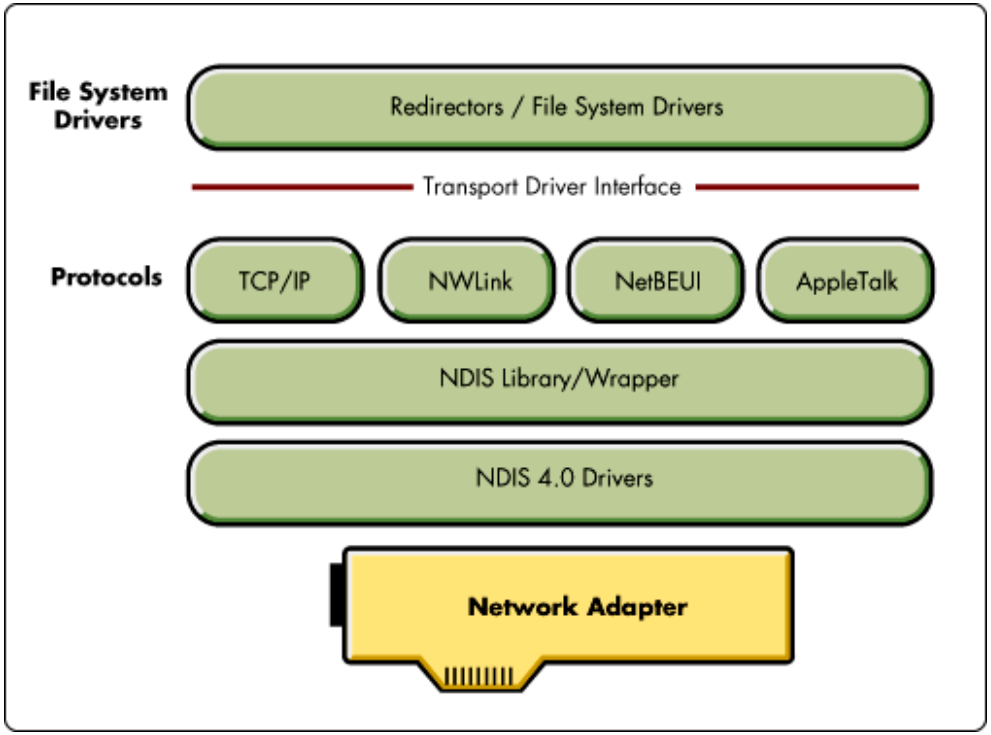
```
//default pragma pack (8)
struct s
{
    int i; // aligned on byte boundary 0, size is 4
    short j; // aligned on byte boundary 4, size is 2
    double k; // aligned on byte boundary 8, size is 8
}   //sizeof(s) is 16
```

```
#pragma pack (2)
struct s
{
    int i; // aligned on  0
    short j; // aligned on  4
    double k; // aligned on  6
}   //sizeof(s) is 14
```

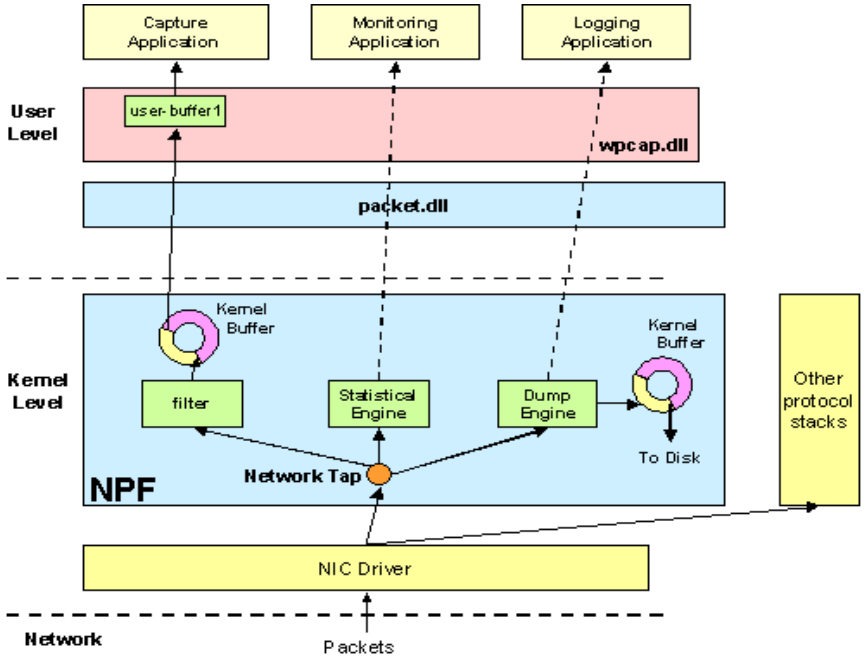
Documentation and Manuals

- The WinPcap manual and tutorial:
  - *http://www.winpcap.org/docs/docs_412/html/main.html*
  - an offline version can be found in the developer's pack
  - inside this manual you will find
    - the documentation of the WinPcap API,
    - a tutorial that will explain how to use the WinPcap functions with several samples,
    - the instructions to compile WinPcap and the applications that use it,
    - a complete description of the internals of WinPcap with links to the source code.

NDIS Drivers



Netgroup Packet Filter Structure



NPF(Netgroup Packet Filter) device driver

Loris Degioanni, Mario Baldi, Fulvio Risso and Gianluca Varenni, Profiling and Optimization of Software-Based Network-Analysis Applications, *Proceedings of the 15th IEEE SBAC-PAD 2003*, Sao Paulo, Brazil, Nov. 2003

Steps for a packet capturing

1. Retrieve device list from the local machine
2. Select the adapter and open it
3. Start the capture
4. May filter the traffic

1. Retrieve device list

- pcap_findalldevs_ex( )
  - Create a list of network devices that can be opened with pcap_open().

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

  - Para
    - *source,:* PCAP_SRC_FILE_STRING = “file://” for local host,
    - auth,: it can be NULL in case of a query to the local host.
    - *alldevs,:* a 'struct pcap_if_t' pointer, which will be properly allocated inside this function.
    - errbuf,: a pointer to a user-allocated buffer (of size PCAP_ERRBUF_SIZE) that will contain the error message (in case there is one).
  - Returns:
    - '0' if everything is fine, '-1' if some errors occurred. The list of the devices is returned in the 'alldevs' variable.

struct pcap_if & pcap_addr

```
struct pcap_if {
struct pcap_if * next;  //a pointer to the next element in the list; NULL for the last element of the list
char *      name; //a pointer to a string giving a name for the device to pass to pcap_open_live()
char *      description; //a pointer to a string giving a human-readable description of the
struct pcap_addr * addresses; //a pointer to the first element of a list of addresses for the interface
u_int       flags;  //PCAP_IF_ interface flags. Currently the only possible flag is PCAP_IF_LOOPBACK,
                    //that is set if the interface is a loopback interface.
};
```

```
struct pcap_addr {
    struct pcap_addr *next;         //a pointer to the next element in the list;
    struct sockaddr *addr;          //a pointer to a struct sockaddr containing an address
    struct sockaddr *netmask;       //a pointer to a struct sockaddr that contains the netmask corresponding
    to the address pointed to by addr.
    struct sockaddr *broadaddr; //a pointer to a struct sockaddr that contains the broadcast address
    corresponding to the address pointed to by addr; may be null if the interface doesn't support broadcasts
    struct sockaddr *dstaddr;       //a pointer to a struct sockaddr that contains the destination address corresponding
    to the address pointed to by addr; may be null if the interface isn't a point- to-point interface
};
```

Example

```
pcap_if_t *alldevs;
pcap_if_t *d;
int i=0;
char errbuf[PCAP_ERRBUF_SIZE];
if (pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL , &alldevs, errbuf) == -1)
{
    fprintf(stderr,"Error in pcap_findalldevs_ex: %s\n", errbuf);
    exit(1);
}
for(d= alldevs; d != NULL; d= d->next) {  /* Print the list */
    printf("%d. %s", ++i, d->name);
    if (d->description)
        printf(" (%s)\n", d->description);
    else
        printf(" (No description available)\n");
}
if (i == 0){
    printf("\nNo interfaces found! Make sure WinPcap is installed.\n");
    return;
}
pcap_freealldevs(alldevs);   /* free the device list*/
```

2. Select an adapter and open it

- pcap_open( ): Open a generic source in order to capture / send (WinPcap only) traffic.

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

- Parameters:
  - *source,:* string containing the source name to open.
  - *snaplen,:* length of the packet that has to be retained.
  - *flags,:* keeps several flags that can be needed for capturing packets.
  - *read_timeout,:* read timeout in milliseconds.
  - *auth,:* it can be NULL in case of a local host.
  - *errbuf,:* a pointer to a buffer which will contain the error in case this function fails.
- Returns:
  - A pointer to a '*pcap_t*' which can be used as a parameter to the following calls.(*This structure is **opaque** to the user*)
  - In case of problems, it returns NULL and the 'errbuf' variable keeps the error message.

Example

```
pcap_t *adhandle;
pcap_if_t *d;
for(d=alldevs, i=0; i< inum-1 ;d=d->next, i++); /* Jump to the selected adapter */
if ( (adhandle= pcap_open(d->name,    // name of the device
                        65536,        // portion of the packet to capture
                                      // 65536 guarantees that the whole packet
                        PCAP_OPENFLAG_PROMISCUOUS,    // promiscuous mode
                        1000,         // read timeout
                        NULL,         // authentication on the remote machine
                        errbuf        // error buffer
                        ) ) == NULL)
{
   fprintf(stderr,"\nUnable to open the adapter. %s is not supported by WinPcap\n", d->name);
    pcap_freealldevs(alldevs); /* Free the device list */
    return -1;
}
printf("\nlistening on %s...\n", d->description);
pcap_freealldevs(alldevs);          //free the device list
```

3.1 Start the capture with *callback* function

- pcap_loop( ): Collect a group of packets.

```
int pcap_loop  (  pcap_t *  p,
                  int cnt,
                  pcap_handler  callback,
                  u_char *     user
            )
```

- it keeps reading packets until cnt packets are processed or an error occurs.
  - A negative cnt causes pcap_loop() to loop forever (or at least until an error occurs).
  - -1 is returned on an error;
  - 0 is returned if cnt is exhausted;

pcap_handler *callback* function

- Prototype of the callback function that receives the packets.

```
//When pcap_dispatch() or pcap_loop() are called by the user,
//the packets are passed to the application by means of this callback.
typedef void(* pcap_handler)(
                    u_char *user,
                    const struct pcap_pkthdr *pkt_header,
                    const u_char *pkt_data)
```

- Parameters
  - **user** is a user-defined parameter that contains the state of the capture session, it corresponds to the user parameter of pcap_dispatch() and pcap_loop().
  - **pkt_header** is the header associated by the capture driver to the packet. It is NOT a protocol header.
  - **pkt_data** points to the data of the packet, including the protocol headers.

Example

```
pcap_loop(adhandle, 0, packet_handler, NULL);

/* Callback function invoked by libpcap for every incoming packet */
void packet_handler(u_char *user, const struct pcap_pkthdr *header, const u_char *pkt_data)
{
    struct tm *ltime;
    char timestr[16];
    time_t local_tv_sec;

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

    printf("%s,%.6d len:%d\n", timestr, header->ts.tv_usec, header->len);

}
```

```
struct pcap_pkthdr
{
    struct timeval ts;
    bpf_u_int32 caplen;
    bpf_u_int32 len;
};
```

3.2 Start capture without callback

```
//Read a packet from an interface or from an offline capture.
int pcap_next_ex ( pcap_t *  p,
                   struct pcap_pkthdr **  pkt_header,
                    const u_char **  pkt_data  )
```

This function is used to retrieve the next available packet, bypassing the callback method traditionally provided by libpcap.

pcap_next_ex fills the pkt_header and pkt_data parameters (see pcap_handler()) with the pointers to the header and to the data of the next captured packet.

The return value can be:

- **1** ***if the packet has been read without problems***
- **0** if the **timeout** set with **pcap_open**() has elapsed. In this case pkt_header and pkt_data don't point to a valid packet
- **-1** if an error occurred
- **-2** if EOF was reached reading from an offline capture

Example

```
struct pcap_pkthdr *header;
const u_char *pkt_data;
pcap_t *adhandle;
//…. adhandle= pcap_open( …)
/* Retrieve the packets */
while((res = pcap_next_ex( adhandle, &header, &pkt_data)) >= 0){

    if(res == 0)
        /* Timeout elapsed */
        continue;

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

    printf("%s,%.6d len:%d\n", timestr, header->ts.tv_usec, header->len);
}

if(res == -1){
    printf("Error reading the packets: %s\n", pcap_geterr(adhandle));
    return -1;
}
```

4. Filtering the traffic

- pcap_compile() takes a string containing a high-level Boolean (filter) expression and produces a low-level byte code that can be interpreted by the filter engine in the packet driver
- pcap_setfilter() associates a filter with a capture session in the kernel driver.
  - if (d->addresses != NULL
    - netmask=((struct sockaddr_in *)(d->addresses->netmask))->sin_addr.S_un.S_addr;
  - else
    - netmask=0xffffff;
- compile the filter
  - struct bpf_program fcode;
  - pcap_compile(adhandle, &fcode, "ip and tcp", 1, netmask)
- set the filter
  - pcap_setfilter(adhandle, &fcode)

pcap_compile( ) & pcap_setfilter( )

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

Compile a packet filter, a pointer to a bpf_program struct and is filled in by pcap_compile().

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

Associate a filter to a capture.
pcap_setfilter() is used to specify a filter program.
fp is a pointer to a bpf_program struct, usually the result of a call to pcap_compile().

Libpcap File Format

- The file has a global header containing some global information followed by zero or more records for each captured packet, looking like this:

Global Header	Packet Header	Packet Data	Packet Header	Packet Data	Packet Header	Packet Data	…

- This format is a standard used by many network tools including WinDump, Wireshark and Snort.
- PCAP Next Generation Dump File Format http://www.winpcap.org/ntar/draft/PCAP-DumpFileFormat.html

Global Header

- This header starts the libpcap file and will be followed by the first packet header:

```
typedef struct pcap_hdr_s {
        guint32 magic_number;   /* magic number */
        guint16 version_major;  /* major version number */
        guint16 version_minor;  /* minor version number */
        gint32  thiszone;       /* GMT to local correction */
        guint32 sigfigs;        /* accuracy of timestamps */
        guint32 snaplen;        /* max length of captured packets, in octets */
        guint32 network;        /* data link type */
} pcap_hdr_t;
```

magic_number: used to detect the file format itself and the byte ordering. The writing application writes 0xa1b2c3d4 with it's native byte ordering format into this field. The reading application will read either 0xa1b2c3d4 (identical) or 0xd4c3b2a1 (swapped). If the reading application reads the swapped 0xd4c3b2a1 value, it knows that all the following fields will have to be swapped too.

Record (Packet) Header & Data

- Each captured packet starts with (any byte alignment possible):

```
typedef struct pcaprec_hdr_s {
        guint32 ts_sec;         /* timestamp seconds */
        guint32 ts_usec;         /* timestamp microseconds */
        guint32 incl_len;       /* number of octets of packet saved in file */
        guint32 orig_len;        /* actual length of packet */
} pcaprec_hdr_t;
```

- **Packet Data**
  - The actual packet data will immediately follow the packet header as a data blob of *incl_len* bytes without a specific byte alignment.

Saving packets to a dump file

- *pcap_dumper_t* *dumpfile; //libpcap savefile descriptor.

```
pcap_t *adhandle;
adhandle= pcap_open(…);
dumpfile= pcap_dump_open(adhandle, filename);
 // start the capture
 pcap_loop(adhandle, 0, packet_handler, (unsigned char *)dumpfile);

/* Callback function invoked by libpcap for every incoming packet */
void packet_handler(u_char *dumpfile, const struct pcap_pkthdr *header, const u_char
    *pkt_data)
{
     pcap_dump(dumpfile, header, pkt_data); /* save the packet on the dump file */
}
```

Reading packets from a dump file

```
pcap_t *fp;
char source[PCAP_BUF_SIZE];
/* Create the source string according to the new WinPcap syntax */
pcap_createsrcstr( source,        // variable that will keep the source string
                   PCAP_SRC_FILE,  // we want to open a file
                   NULL,           // remote host
                   NULL,           // port on the remote host
                   filename,        // name of the file we want to open
                   errbuf          // error buffer
                   )
fp= pcap_open(source,        // name of the device
              65536,         // portion of the packet to capture
                             // 65536 guarantees that the whole packet
              PCAP_OPENFLAG_PROMISCUOUS,     // promiscuous mode
              1000,          // read timeout
              NULL,           // authentication on the remote machine
              errbuf         // error buffer
              )
pcap_loop(fp, 0, dispatcher_handler, NULL); // read and dispatch packets until EOF is
          reached or while((res = pcap_next_ex( fp, &header, &pkt_data)) >= 0) {…}
```

Sending a single packet

- Sending a single packet with pcap_sendpacket( )

```
pcap_t *fp;
u_char packet[100];  //packet to send
fp= pcap_open(…)
pcap_sendpacket(fp, packet, 100 /* size */)
```

```
//send a raw packet to the network.
int pcap_sendpacket (
        pcap_t *  p,        //the interface that will be used to send the packet
        u_char *  buf,      // contains the data of the packet to send
                            //(including the various protocol headers)
        int       size      //size is the dimension of the buffer pointed by buf
        )
```

Example

```
pcap_t *fp;
char errbuf[PCAP_ERRBUF_SIZE];
u_char packet[100];
int I;
/* Open the output device */
fp= pcap_open(argv[1],          // name of the device
              100,              // portion of the packet to capture (only the first 100 bytes)
              PCAP_OPENFLAG_PROMISCUOUS,  // promiscuous mode
              1000,             // read timeout
              NULL,             // authentication on the remote machine
              errbuf            // error buffer
              ) ;
/* Supposing to be on ethernet, set mac destination to 1:1:1:1:1:1 */
   packet[0]=1;  packet[1]=1; packet[2]=1;packet[3]=1; packet[4]=1; packet[5]=1;
/* set mac source to 2:2:2:2:2:2 */
   packet[6]=2; packet[7]=2; packet[8]=2; packet[9]=2; packet[10]=2; packet[11]=2;
/* Fill the rest of the packet */
   for(i=12;i<100;i++;  {  packet[i]=(u_char)I;  }
   /* Send down the packet */
   if (pcap_sendpacket(fp, packet, 100 /* size */) != 0) {
      fprintf(stderr,"\nError sending the packet: %s\n", pcap_geterr(fp));
      return;
   }
```

Send packets in queue

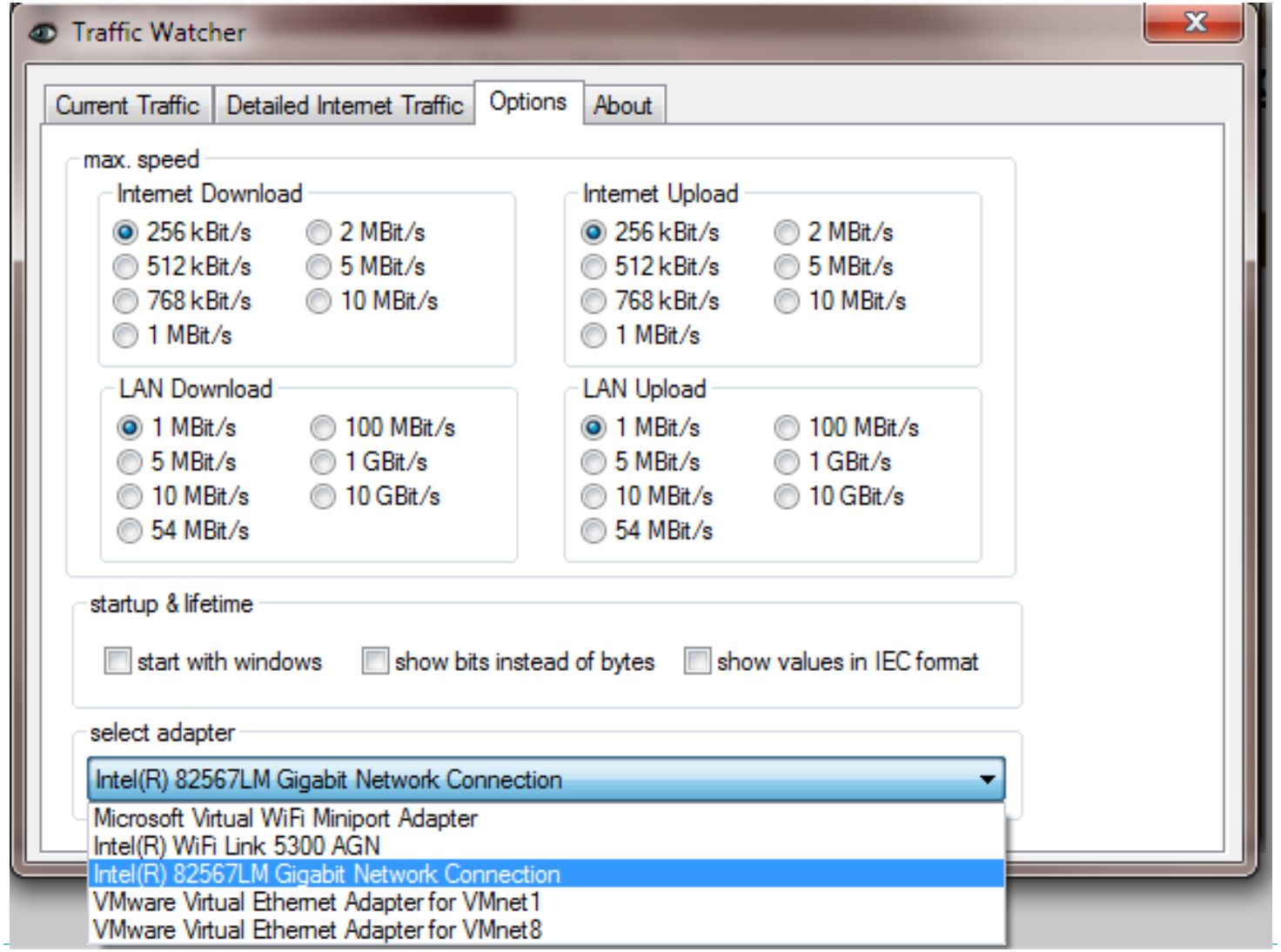
- A send queue is created calling the pcap_sendqueue_alloc()
  1. pcap_sendqueue_queue() to add a packet to the send queue.
  2. pcap_sendqueue_transmit() to transmit queue
  3. deleted queue with pcap_sendqueue_destroy()

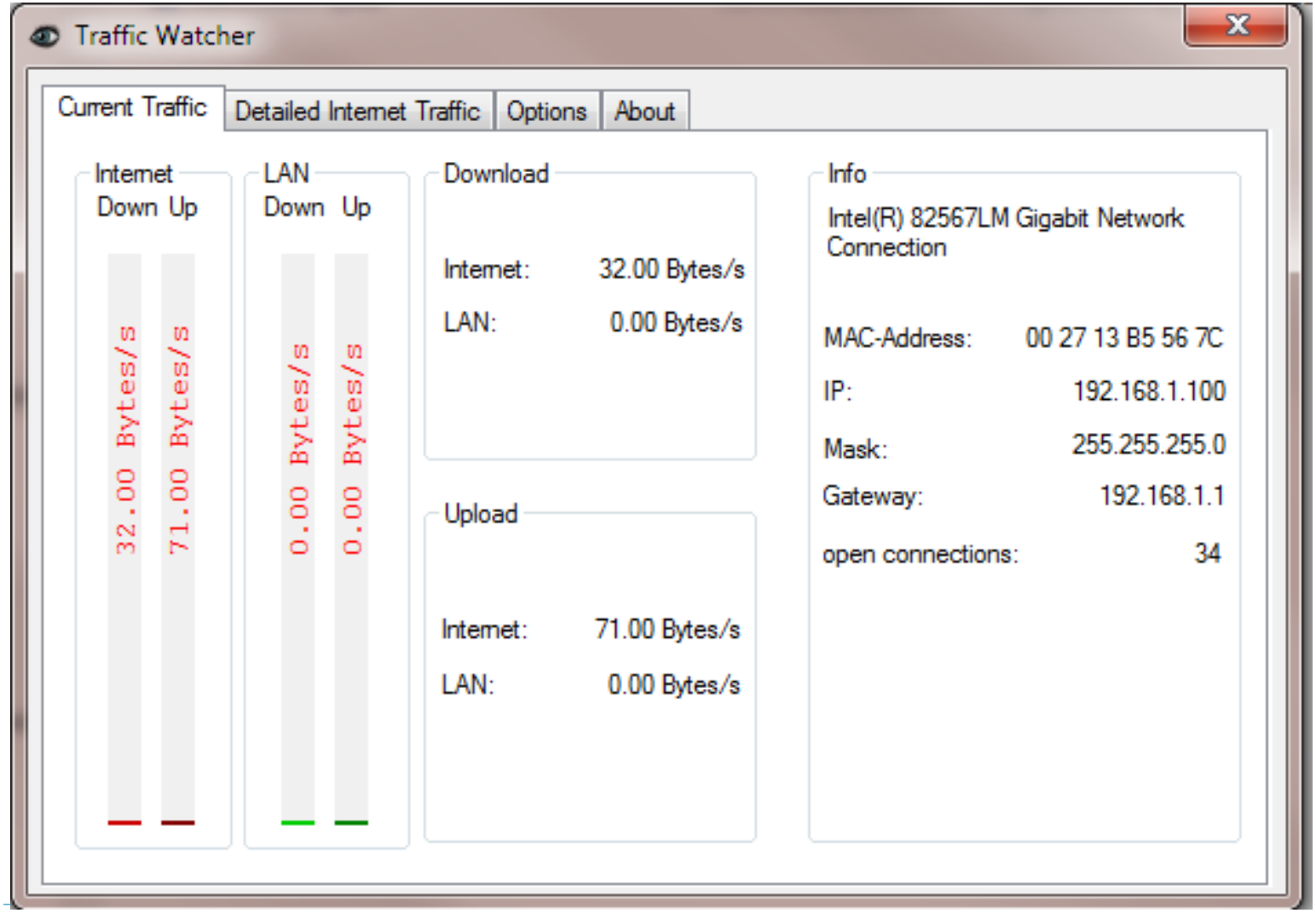
Note that transmitting a send queue with pcap_sendqueue_transmit() is much more efficient than performing a series of pcap_sendpacket(), because the send queue is buffered at kernel level drastically decreasing the number of context switches.

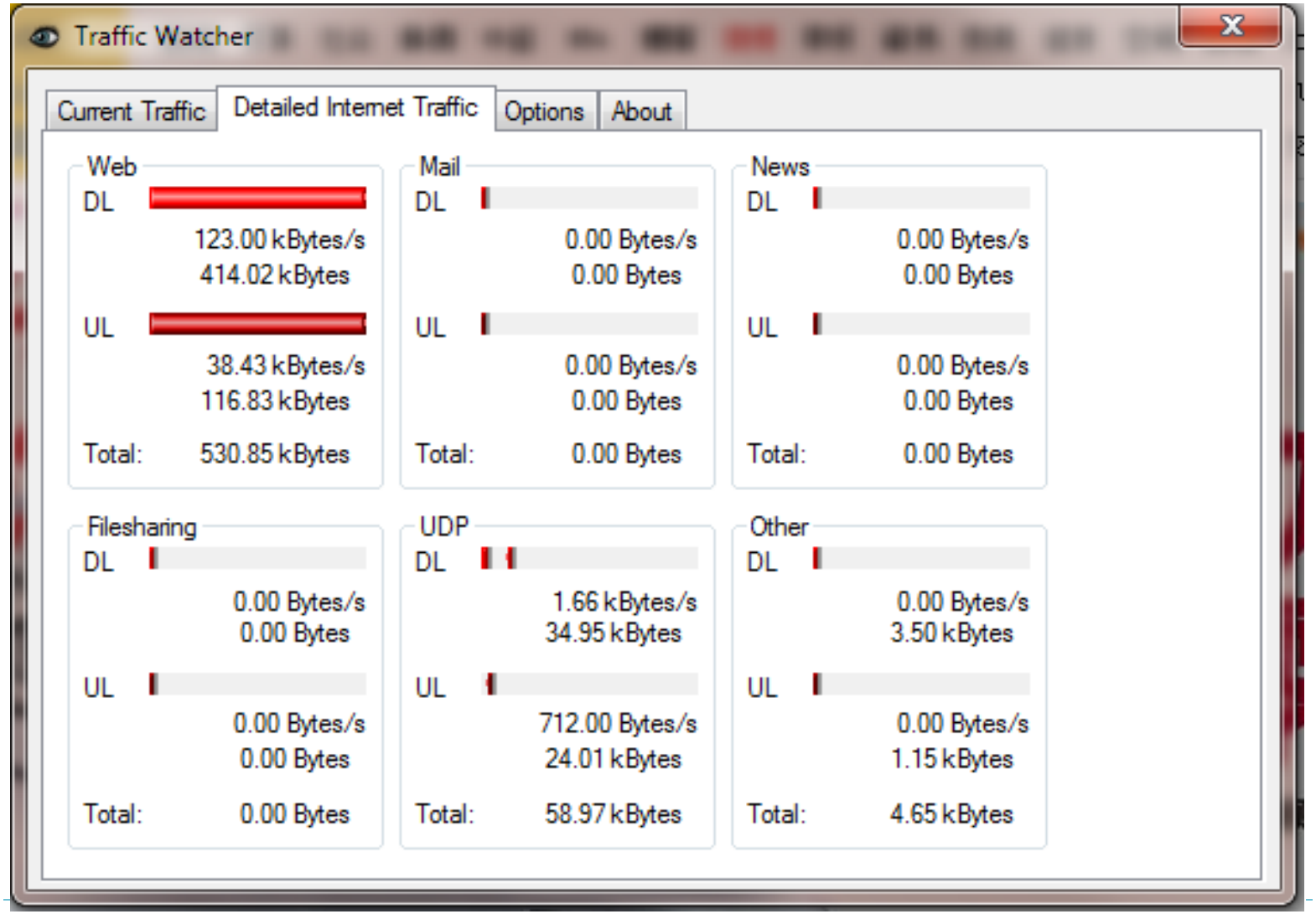
Example: Send a captured file

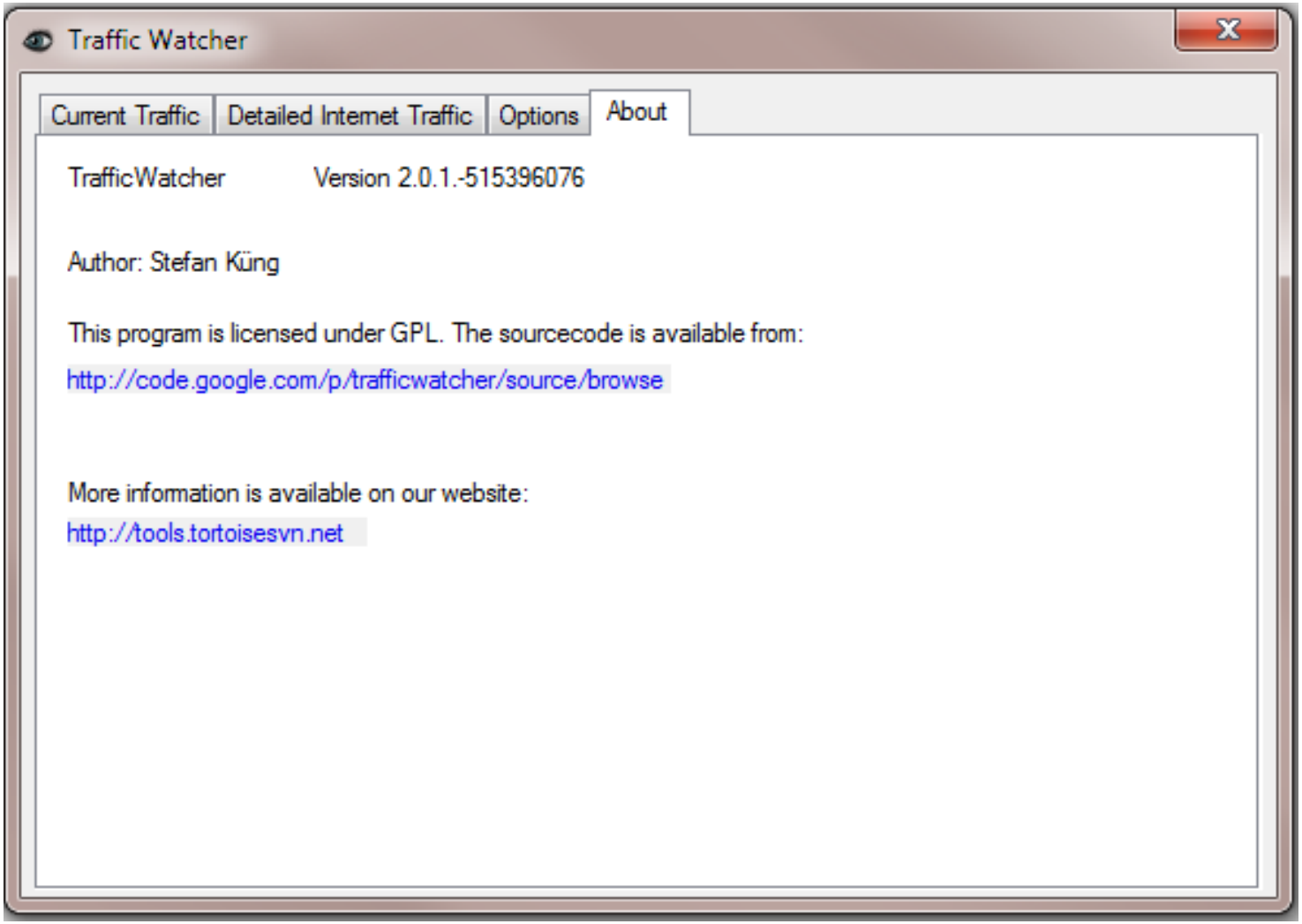
```
/* Open the capture */
/* Create the source string according to the new WinPcap syntax */
pcap_createsrcstr( source,        // variable that will keep the source string
                   PCAP_SRC_FILE,  // we want to open a file
                   NULL,           // remote host
                   NULL,           // port on the remote host
                   argv[1],        // name of the file we want to open
                   errbuf          // error buffer
                   ) ;
/* Open the capture file */
pcap_t * indesc= pcap_open(source, 65536, PCAP_OPENFLAG_PROMISCUOUS, 1000, NULL, errbuf);
/* Open the output adapter */
pcap_t * outdesc= pcap_open(argv[2], 100, PCAP_OPENFLAG_PROMISCUOUS, 1000, NULL, errbuf);
pcap_send_queue squeue = pcap_sendqueue_alloc(caplen);
/* Fill the queue with the packets from the file */
   while ((res = pcap_next_ex( indesc, &pktheader, &pktdata)) == 1)    {
      if (pcap_sendqueue_queue(squeue, pktheader, pktdata) == -1)       {
         printf("Warning: packet buffer too small, not all the packets will be sent.\n");
         break;
      }
      npacks++;
   }
 pcap_sendqueue_transmit(outdesc, squeue, sync)
 pcap_sendqueue_destroy(squeue);
```

Sample: Traffic Watcher




Traffic Watcher
Current Traffic
Detailed Internet Traffic
Options
About
Internet
Down Up
32.00 Bytes/s
71.00 Bytes/s
LAN
Down Up
0.00 Bytes/s
0.00 Bytes/s
Download
Internet: 32.00 Bytes/s
LAN: 0.00 Bytes/s
Upload
Internet: 71.00 Bytes/s
LAN: 0.00 Bytes/s
Info
Intel(R) 82567LM Gigabit Network Connection
MAC-Address: 00 27 13 B5 56 7C
IP: 192.168.1.100
Mask: 255.255.255.0
Gateway: 192.168.1.1
open connections: 34


Traffic Watcher
Current Traffic
Detailed Internet Traffic
Options
About
Web
DL
123.00 kBytes/s
414.02 kBytes
UL
38.43 kBytes/s
116.83 kBytes
Total: 530.85 kBytes
Mail
DL
0.00 Bytes/s
0.00 Bytes
UL
0.00 Bytes/s
0.00 Bytes
Total: 0.00 Bytes
News
DL
0.00 Bytes/s
0.00 Bytes
UL
0.00 Bytes/s
0.00 Bytes
Total: 0.00 Bytes
Filesharing
DL
0.00 Bytes/s
0.00 Bytes
UL
0.00 Bytes/s
0.00 Bytes
Total: 0.00 Bytes
UDP
DL
1.66 kBytes/s
34.95 kBytes
UL
712.00 Bytes/s
24.01 kBytes
Total: 58.97 kBytes
Other
DL
0.00 Bytes/s
3.50 kBytes
UL
0.00 Bytes/s
1.15 kBytes
Total: 4.65 kBytes


Traffic Watcher
Current Traffic
Detailed Internet Traffic
Options
About
TrafficWatcher
Version 2.0.1.-515396076
Author: Stefan Küng
This program is licensed under GPL. The sourcecode is available from:
http://code.google.com/p/trafficwatcher/source/browse
More information is available on our website:
http://tools.tortoisesvn.net

references

- http://www.winpcap.org/devel.htm
- http://www.winpcap.org/docs/docs_412/index.html
- http://www.ferrisxu.com/WinPcap/html/index.html
- http://wiki.wireshark.org/WinPcap
- http://en.wikipedia.org/wiki/Pcap
- http://tools.tortoisesvn.net/
- http://msdn.microsoft.com/en-us/library/2e70t5y1%28VS.80%29.aspx