CS3103 PROGRAMMING ASSIGNMENT 2:
NETWORK TRAFFIC AND PACKET FORMAT ANALYSIS
SEMESTER 1 / AY 2012-2013
SUBMISSION DATE: BEFORE 5 PM NOV 9TH 2012

1. **This assignment should be done individually. Exchange students are allowed to do the assignment either individually, or in groups of 2 (with other exchange students only).**
2. **Any programming language can be used - C/C++/C#/Java/Python/Perl etc.**
3. **Please upload a tarball/zip of your source code and Readme file with the name "CS3103_PA2_YourMatricNumber.zip" into the IVLE Workbin Student Submission folder before the submission date. Please specify your matric number(s) clearly in the Readme file.**
4. **There is a 10% penalty per day for late submission.**
5. **Demonstrations will be done on Nov 12th and 13th, using the code uploaded on IVLE**
6. **For any clarification on the assignment, please email Kartik Sankaran kartiks@comp.nus.edu.sg, or post your questions on the IVLE forum.**

OVERVIEW

One of the important tasks faced by network programmers is to analyze packet traces to debug and to study the behaviour of protocols. Often, the amount of traces collected is too voluminous to inspect one by one, and requires a more efficient and automated method. In this assignment, you are going to parse and interpret Ethernet frames. You will have the opportunity to understand how gigabytes of data for network research can be parsed efficiently.

DESCRIPTION

Implement a packet parser to process Ethernet frames captured by `wireshark` [1] in hexadecimal format. You may find it easier to use languages such as Python and Perl for this assignment, though Java/C++ will work fine too. The input text file contains the hex bytes of each packet. A snapshot of the file contents is given below –

```
...
0000  00 00 0c 07 ac 00 00 22 68 c3 b8 fd 08 06 00 01   ........"h.......
0010  08 00 06 04 00 01 00 22 68 c3 b8 fd ac 12 b6 9b   ........"h.......
0020  00 00 00 00 00 00 ac 12 b6 01                     ..........

0000  00 22 68 c3 b8 fd 00 00 0c 07 ac 00 08 06 00 01   ."h.............
0010  08 00 06 04 00 02 00 00 0c 07 ac 00 ac 12 b6 01   ................
0020  00 22 68 c3 b8 fd ac 12 b6 9b                     ."h.......

0000  ff ff ff ff ff ff 00 22 68 c3 b8 fd 08 06 00 01   ........"h.......
0010  08 00 06 04 00 01 00 22 68 c3 b8 fd ac 12 b6 9b   ........"h.......
0020  00 00 00 00 00 00 ac 12 b6 01                     ..........
...
```

The format is the similar to the output of the `od` (octal dump) command in UNIX. The first column is the byte offset in hex, the middle part is the actual bytes of the packet in hex, the last column displays the bytes in ASCII (a dot `.` is used if the byte is not an ASCII character). Packets are separated by blank lines.

Your program will take the input file and print a summary regarding the total number of Ethernet, ARP, IP, ICMP, TCP, UDP, Ping, DHCP and DNS packets respectively. For the DNS packets, the program will print out the details regarding the network transactions.

Some sample data files are provided in Workbin –
`ascii.dat`: Text file giving a detailed description of each packet (as displayed in `wireshark`).
`hex.dat`: Text file (used as input to your program) giving the same packets in hex format (as shown above).
`packets.dump`: Binary (packet capture) file which can be opened by `wireshark`.

The text files are rather large, so use a more powerful text editor to open them, such as `Geany` [5] or `emacs`. Disable line wrapping in your editor to make the file loading process much faster. `Emacs` can open the file quickly even with line wrapping enabled.

You can use the sample data files to test and check your program. I will test your program on a much larger packet trace. If your program is correct, it should run properly with the large trace too.

REQUIREMENTS

You may implement each requirement in a different source file (so that if one crashes, it will not crash the other). The input file may contain other irrelevant packets which may be ignored by the program. Suppose the requirements are implemented in Python consisting of two files: `count.py` and `dns.py`.

The requirements for these programs are described as follows –

1. The `count.py` program should print out the protocol/frame types and the corresponding packet count. The basic protocol/frame types include Ethernet, IP, ARP, ICMP, UDP, TCP, Ping, DNS and DHCP. An example output is given below:

```
$ ./count.py hex.dat
total number of Ethernet (IP + ARP) packets =  4021
total number of IP packets =  4011
total number of ARP packets =  10
total number of ICMP packets =  1098
total number of TCP packets =  1645
total number of UDP packets =  1262
total number of Ping packets =  1097
total number of DHCP packets =  14
total number of DNS packets =  1171
```

2. The `dns.py` program prints the total number of DNS packets, and the total number of DNS transactions. By transaction, we mean that there is a response to the request (note that some requests may not have responses). In addition, the program must print out the Answer Resource Records of the DNS Responses for 'A' Queries (you can ignore other query types). An example output is given below:

```
$ ./dns.py hex.dat
total number of DNS packets =  1171
total number of DNS transactions =  584
----------------------
DNS Transaction
----------------------
transaction_id =  81ca
Questions =  1
Answer RRs =  1
Authority RRs =  3
Additional RRs =  3
Queries:
       Name =  ntp.ubuntu.com.
       Type =  1
       Class =  1
Answers:
       Name =  ntp.ubuntu.com.
       Type =  1
       Class =  1
       Time to live =  600
       Data length =  4
       Addr =  91.189.94.4

... Other transactions ...
----------------------
DNS Transaction
----------------------
transaction_id =  fdcf
Questions =  1
Answer RRs =  2
Authority RRs =  3
Additional RRs =  1
Queries:
       Name =  bt.mit.edu.
       Type =  1
       Class =  1
Answers:
       Name =  bt.mit.edu.
       Type =  5
       Class =  1
       Time to live =  21600
```

```
        Data length =  17
        CNAME =  SCRIPTS-VHOSTS.mit.edu.

        Name =  SCRIPTS-VHOSTS.mit.edu.
        Type =  1
        Class =  1
        Time to live =  21600
        Data length =  4
        Addr =  18.181.0.46
... Other transactions ...
```

If you analyse any other packets found in the trace, you will get bonus points.

GUIDELINES

1. Remember to convert from network order to host order while analysing packets, using the *hton()* and *ntoh()* functions or equivalents.
2. Remember to follow the principle – "Be liberal in what you accept" [6], otherwise your program may crash when I run it on a larger trace.
3. Try to make your program reasonably efficient. I should not have to wait for too long on the small trace just to get output.
4. You can use filter expressions in `wireshark` to find packets easily. For example, to find the DNS packets with Transaction ID of 0xfdcf, use the expression "dns.id==0xfdcf". To see all DNS responses to 'A' queries, use "udp.srcport==53 and dns.qry.type==1".
5. You can generate your own packet trace using `wireshark` to further test your program. Please contact the TA to find out how to generate the `hex.dat` file using `wireshark`.
6. In the analysis of DNS packets, be sure to understand how Fully Qualified Domain Names are encoded and compressed [7].
7. If you go through the `hex.dat` file carefully, you will spot some blocks of bytes that need to be ignored (such as Reassembled TCP blocks). Make sure you ignore these in your program.

SUBMISSION

Please submit a tarball/zip named "CS3103_PA2_YourMatricNumber.zip" in the student submission folder in IVLE by the due date. The tarball should contain the source code and a README file. The README file must have your name(s) and matric number(s), instructions on how to compile and run the code, the platform (OS) you developed on, and most importantly some sample output. You can also mention any extra features you may have implemented for extra credit.

EVALUATION AND DEMO

After submitting the code in IVLE, each person (or group) will have to show a demo. Please note that each of you will be evaluated separately, even if it is a group. Make sure that you have understood the working of the code very well, even if some parts of the code were written by your teammate.

During the demo, please bring a printout of your source code and readme file, with your name(s) and matric number(s) written clearly on top. You are encouraged to bring your laptop for the demo. If you are unable to bring your laptop, then make sure that your code can compile and run correctly on Ubuntu 10.04 (Linux). Please note that during the demo you are expected to download the source code uploaded in IVLE, compile it and execute it.

This assignment counts 10% to your final grade. The total points is 100.

1. **Task 1: Counting packets (40 points)**
2. **Task 2: DNS packet analysis (40 points)**
3. **Answering questions correctly during the demo (20 points)**

There is a bonus of 10 points for any extra packet analysis.

REFERENCES

[1] Wireshark: Packet capturing software for both Windows and Linux
http://www.wireshark.org/
[2] Packet Formats: Header format for TCP/IP packets
http://www.networksorcery.com/enp/topic/ipsuite.htm
[3] Perl: Tutorial of the Perl language
http://www.perl.com/pub/a/2000/10/begperl1.html
[4] Python: Tutorial of the Python language
http://docs.python.org/tutorial/index.html
[5] Geany: Text Editor
http://www.geany.org/
[6] Robustness Principle: Wikipedia
http://en.wikipedia.org/wiki/Robustness_principle
[7] DNS Message Resource Record Field Formats
http://www.tcpipguide.com/free/t_DNSNameNotationandMessageCompressionTechnique.htm
http://www.tcpipguide.com/free/t_DNSNameNotationandMessageCompressionTechnique-2.htm