

Computer Networks
– A Simple Network Analyzer –
PART A – undergraduates and graduates
PART B – graduate students only

Objectives

The main objective of this assignment is to gain an understanding of network activities and network packet formats using an existing packet sniffer program (Ethereal or Wireshark or tcpdump), which uses a normal network card to monitor the network traffic passing your computer.

Overview

Most of the ethernet frames that you will find on an Ethernet are either ARP or IP packets. The ethernet type field in the ethernet frame is used to distinguish what kind of packet (ARP or IP) is encapsulated in the ethernet frame data area.

Packets that run on top of IP (UDP or TCP) are encapsulated in the data section of the IP packets. A field in the IP header specifies the encapsulated protocol.

What To Do – Part A:

Download and install on your computer a packet sniffer (Ethereal or Wireshark in Windows, and tcpdump in Unix). Familiarize yourself with the sniffer program by studying the documentation available on the Internet.

Capturing and Saving Packets:

1. Familiarize yourself with the syntax and options of the [ping](#) program. For instance, open a command window and try


```
ping www.google.com
```
2. Find out your IP address. To do this, type [ipconfig /all](#) at the command prompt in Windows, or [ifconfig -a](#) in Unix. You will get a bunch of configuration information, one piece of which is your IP address.
3. Start capture session on your packet sniffer. Make sure you select a correct interface card. Learn how to control the volume of captured frames using filters, and capture only frames to or from your computer.
4. Ping any remote computer -- for instance, [www.cnn.com](#). You should see in the Ethereal window one ARP Request, one ARP Reply and a bunch of other packets. Stop capturing packets.

- Save the captured packets in a file in format `libpcap (tcpdump)` – this is the format that we will be decoding. Make sure to save the file in a folder where you can find it again. It is easy to save it in the wrong location if you do not pay attention.

Decoding Packets

Implement your own simple network analyzer that reads Ethernet frames from a tcpdump file (the one you have saved in Part II) and prints out information from the Ethernet header of each frame. If the Ethernet frame contains an IP packet, then print the IP header (only the fields described below). Your program should display the fields in the header in the following form:

```
ETHER: ----- Ether Header -----
ETHER:
ETHER: Packet 1
ETHER: Packet size = 210 bytes
ETHER: Destination = 08:00:20:01:3d:94
ETHER: Source = 08:00:69:01:5f:0e
ETHER: Ethertype = 0800 (IP)
ETHER:
IP: ----- IP Header -----
IP:
IP: Version = 4
IP: Header length = 20 bytes
IP: Total length = 196 bytes
IP: Time to live = 255 seconds/hops
IP: Protocol = 17 (UDP)
IP: Source IP address = 128.10.26.116
IP: Destination IP address = 128.10.3.100
...
Number of records processed: 112
Average packet length: 827 bytes
```

Notes:

- Next to the Ethernet type, print out a label identifying the type of data encapsulated in the frame (IP, ARP or unknown):

<u>Ethernet type</u>	<u>Ethernet data</u>
0x0800	IP
0x0806	ARP
otherwise	UNKNOWN

- You will need to use bit operations (`&`, `>>`) to extract bits out of a byte. Examples:
`b & 0x0F` value of the second half of b
`(b & 0xF0) >> 4` value of the first half of b

Format of a tcpdump capture file

The tcpdump capture file is organized as follows. The first line contains 24 bytes that should be skipped. Then, a series of variable-length records follows. Each record starts with a header containing information about how long the record is, how long the Ethernet frame encapsulated in the record is, and so on. The record header is followed by the Ethernet frame itself (that is, Ethernet header and Ethernet data), that we want to analyze. A record has the following layout:

Record header	Ethernet header	Ethernet data
RecHeader	EthHeader	unused here

In this assignment you will process the record header, the Ethernet header and the Ethernet data. The record header contains the following information:

RecHeader:

Timestamp	– an unsigned 64 bit integer
Frame length	– an unsigned 32 bit integer
Captured data length	– an unsigned 32 bit integer

- Timestamp is the time that the NIC driver sees the packet – ignore it
- Frame length is the length of Ethernet header + Ethernet data
- Captured data length – ignore it

The Ethernet header contains the following information:

EthHeader:

MAC address of destination	– on 6 unsigned bytes (48 bits)
MAC address of source	- on 6 unsigned bytes (48 bits)
The protocol type	- on 2 unsigned bytes (16 bits)

The IP header contains the following information:

IP_Header:

Version and header length	on 1 unsigned byte
Type of service	on 1 unsigned byte
Total packet length	on 2 unsigned bytes
Datagram identifier	on 2 unsigned bytes
Flags and fragment offset	on 2 unsigned bytes
Time to live (gateway hops)	on 1 unsigned byte
Type of protocol	on 1 unsigned byte
Header checksum	on 2 unsigned bytes
Source IP address	on 4 unsigned bytes
Destination IP address	on 4 unsigned bytes
(No options)	

Integer fields in the IP packets that are longer than one byte must be converted from network byte order to host byte order.

Note: Next to the Protocol type (IP header), print out a label identifying the type of data encapsulated in the IP datagram:

<u>Protocol type</u>	<u>Protocol data</u>
17	UDP
1	ICMP
6	TCP
otherwise	UNKNOWN

Implementation suggestions

The key to success in developing a program is to build in incrementally, in stages. I suggest that you go through the following implementation stages:

Stage 1. Write a program that does the following:

- open the input data file in binary
- skip the first 24 bytes
- read the first record header
- print out the length of the first record
- load the data file in Ethereal and compare the value printed by your program with the length value given by Ethereal – should be the same

Stage 2. Extend the program from Stage 1 to do the following :

- read the rest of first the record (that is, the Ethernet frame)
- read the second record header
- print out the length of the second record
- load the data file in Ethereal and compare the value printed by your program with the length value given by Ethereal – should be the same

Once you make sure that you correctly jump to the second record in the file, go to Stage 3 below.

Stage 3. Modify the program from Stage 2 to do the following:

- open the input data file and skip the first 24 bytes
- in a loop (until end of file is encountered) do the following:
 - read the next record header
 - print out the length of the record
- print out the number of records in the input file
- compare the number of records displayed by your program against the number of records given by Ethereal – should be the same

A successful implementation of Stage 3 ensures that you process the records in the input file properly.

Stage 4. Extend the program from Stage 3 to read each Ethernet header following a record header. For each Ethernet header, print out:

- the MAC address of the destination node
- the MAC address of the source node

- the protocol type

Notes for C programmers

Unlike Java, C is platform dependant. There are two representations for storing integers and floating point numbers: big-endian and little-endian. Consider, for instance, the integer 91329, which is 00 01 64 C1 in hexadecimal. This could be stored as:

Big endian:	00	01	64	C1
Little endian:	C1	64	01	00

Each processor chooses its own format, for example Sun Spark uses big-endian and Intel Pentiums use little-endian.

Computer networks are big endian. This means that when little-endian computers (like Pentium based computers running Windows) are going to pass integers over the network, they need to convert them to big endian. Likewise, when they receive integer values over the network, they need to convert them back to their own native representation.

Integer fields that are longer than one byte must be converted from network byte order to host byte order.

What To Do – Part B:

In this part you will extend the network analyzer you wrote in part A to print out the TCP and UDP headers of the captured packets. Print the Ethernet and the IP header as before.

- If the IP datagram contains an UDP packet, then print the UDP header
- If the IP datagram contains a TCP packet, then print the TCP header

The UDP header contains the following information (see textbook, page 602):

UDP_Header:

Source port	on 2 unsigned bytes
Destination port	on 2 unsigned bytes
Checksum	on 2 unsigned bytes
Length	on 2 unsigned bytes

The TCP header contains the following information (see textbook, page 608, or quick reference handout):

TCP_Header:

Source port	on 2 unsigned bytes
Destination port	on 2 unsigned bytes
Sequence number	on 4 unsigned bytes
ACK number	on 4 unsigned bytes
Hlen and Code bits	on 2 unsigned bytes
Window size	on 2 unsigned bytes
... other info that you won't decode ...	

Your program should display the fields of the captured packets as shown below:

```
ETHER: ----- Ether Header -----
ETHER:
ETHER: Packet 1
ETHER: Packet size = 285 bytes
ETHER: Destination = 00:00:c0:51:ad:9c
ETHER: Source = 00:80:3e:4e:76:e0
ETHER: Ethertype = 0x0800 (IP)
ETHER:
IP: ----- IP Header -----
IP:
IP: Version = 4
IP: Header length = 20 bytes
IP: Total length = 271 bytes
IP: Time to live = 253 seconds/hops
IP: Protocol = 17 (UDP)
IP: Source IP address = 153.104.136.2
IP: Destination IP address = 153.104.202.161
IP:
UDP: ----- UDP Header -----
UDP:
UDP: Source port = 53
UDP: Destination port = 3718
UDP: Checksum = 0x1dc4
UDP: Length = 251
```

```
ETHER: ----- Ether Header -----
ETHER:
ETHER: Packet 2
ETHER: Packet size = 62 bytes
ETHER: Destination = 00:80:3f:f4:03:ad
ETHER: Source = 00:06:5b:de:58:60
ETHER: Ethertype = 0x0800 (IP)
ETHER:
IP: ----- IP Header -----
IP:
IP: Version = 4
IP: Header length = 20 bytes
IP: Total length = 48 bytes
IP: Time to live = 128 seconds/hops
IP: Protocol = 6 (TCP)
IP: Source IP address = 153.104.202.98
IP: Destination IP address = 153.104.202.12
IP:
TCP: ----- TCP Header -----
TCP:
TCP: Source port = 4312
TCP: Destination port = 23 (TELNET)
TCP: Sequence number = 1966876696
TCP: Acknowledgement number = 0
TCP: Header length = 28 bytes
TCP: Flags = 0x0002 (SYN)
TCP:    ..0. .... = No urgent pointer
TCP:    ...0 .... = Acknowledgement
```

TCP: 0... = No push
TCP:0.. = No reset
TCP:1. = SYN
TCP:0 = No Fin
TCP: Window size = 64240
...

Number of records processed: 42
Average packet length: 167 bytes

Notes:

- a. Well known port numbers are:

<u>Port number</u>	<u>Application</u>
21	FTP
23	TELNET
25	SMTP (mail server)
80	HTTP (web server)

- b. Use Ethereal to examine the options section of the TCP header. This activity will help you understand the information passed between client and server during connection establishment. You need not decode the options section of the TCP header.

Submission Guidelines

Hand in a printed copy of your program, a sample output, and a short README file that explains any bugs and/or deviations in your program from the assignment specification.

If possible, please bring your laptop to class for a live demonstration of your code.

You may work in teams or individually on this assignment. If you choose to work in teams, please report in the README file the contribution of each team member to the assignment (equal contribution, or approximate percentage contribution). Team members may be asked oral questions in order to evaluate their contribution and understanding of the assignment, before being assigned a grade.

Grading

The majority of your grade for this assignment will depend upon how well your implementation works. I will run your program on a capture file created by myself. In cases of equal contribution, each team member's grade will be determined by:

- 90%: Correct Execution
- 10%: Program structure and Documentation

Use good indentation, meaningful variable names and helpful comments. Start early!

Have Fun!

