Ben Watson

Report for Lab 1

Introduction:

The aim of this project was to build a packet sniffer to successfully interpret various protocols, deal with them appropriately, and print the packet to the command line in a nicely structured form.

Design:

My sniffer code was really quite simple in design, and I reformatted it multiple times in order to achieve this. Firstly, the program runs by collecting 'tcpdump' information and piping it through as a 'packet'. My methods then strip off the Ethernet header, collecting and printing to screen all the various information, and send the packet on to methods dealing with IP layer headers, depending on whether the packet is IPv4, IPv6, or some other, unknown, protocol. From there, the IP header is stripped off, and the data collected and printed, and the packet is sent to the transport layer methods, depending again on what protocol is specified; ICMP, ICMPv6, TCP, UDP, or 'unknown'. Finally, from here the header is, again, removed, and the packet is sent to the final printing methods. Here it is refactored as printed, with spacing, colouring, and tabs to buffer it nicely and present it on screen.

I also included quite substantial testing code, and a python script to run some packets that I created against my sniffer. I tailored the test packets to each protocol and made sure that they test the correctness of the protocols thoroughly.

Apparatus and Procedures:

I developed this in stages. Firstly I dealt only with IPv4, and started out with UDP. I then moved on to TCP once that was working, and then ICMP and other. After IPv4 was complete, I moved on to dealing withIPv6, in the same manner. After finishing this, I formatted it nicely, adding character space buffering and letter colouring. Finally, I created my test script and generated the packets, testing my sniffer against each packet. I found that my ICMP solution had a few bugs and proceeded to fix them. I believe I have tested each protocol pretty comprehensively with the use of my test script and custom packets.

Results and Discussion:

I managed to generate, and run some test output files against my code and got it passing, with results looking similar to this:

```
Packet ID: 1
Ether Type: IPv6
Source IP: 0000:0000:0000:0000:0000:0000:0000:0001
Destination IP: 0000:0000:0000:0000:0000:0000:0000:0001
Protocol: TCP
Source Port: 32769
Destination Port: 9600
Packet Length: 35
================================================================================
00000 01 01 00 08 01 02 00 00   00 00 12 30 00 00 12 31    ...........0...1
00016 00 00 12 32 00 00 12 33   00 00 12 34 d7 cd ef 00    ...2...3...4....
00032 00 12 35                                            ..5
================================================================================
```

I found it quite hard to start this project because I didn't really know what I was doing, and we hadn't yet covered any related material in the lectures. However, once I started, it began making more and more sense and the lectures helped me to understand it more thoroughly. IPv6 was a little bit of an issue, being harder for me than IPv4, with the new addresses and IP header length not being included in the 'length' field. Another issue I ran into was managing to understand when and where to use ntohs and ntohl methods, and figuring out the proper way to obtain the source and destination ports.

Conclusion and Recommendations:

I met all the requirements for this project. I handled TCP, UDP, ICMP, and unknown traffic for both IPv4 and IPv6, Extension headers for IPv6, and handled other Ethernet types (by printing out clearly, rather than simply crashing). I would extend this by increasing the amount of protocols I handle directly, for example, including some code to handle ARP etc. I would also potentially make a better interface and maybe extend my test suite even more.

Acknowledgements:

David C Harrison (david.harrison@ecs.vuw.ac.nz) July 2015

http://www.tcpdump.org/sniffex.c, Version 0.1.1 (2005-07-05), Copyright (c) 2005 The Tcpdump Group