CS 5434 HW 2 – Fall 2015

Due Friday Oct 2nd @ 23:59.

In this homework you will work in groups of two to build a C program capable of parsing packets and storing them in a simple but realistic data structure. The next homework will build on this code to do actual detection tasks.

First, download the VM from the following URL (it's a huge download, so start early). **Do not re-use the VM from homework 1**.

http://www.cs.cornell.edu/courses/CS5434/2015fa/HW2.zip

Some skeleton code and a Makefile has been set up for you in the HW2 directory. The password for this VM is "**peanut**".

Objective

The objective of this assignment is to identify complete TCP connections and count packets within each connection session. Your program should use a hash table to track this state and must be able to handle arbitrarily large (within reason) files and number of connections without failing. It must also be free from memory errors.

Input

Your program will read a series of pcap files using libpcap. Input pcap files are specified on the command line after the "-r" option, as in the following format:

```
./hw2 -r file1.pcap file2.pcap file3.pcap
```

An arbitrary number of files may be specified; your program should parse the pcap files in the order given on the command line. Within each file, all packets should be sequentially read and processed until there are no more packets remaining in the file. Then, **without clearing or modifying the existent in-memory data structures**, the next pcap file should be processed. The overall effect of inputting sequential pcap files is to simulate the operation of your program over an extended input.

The skeleton code provided in hw2.c already does some command line parsing for you – check out the inline program comments for details on how to use it.

Data structures and connection state

Your code should have a flow table data structure that stores information about each TCP connection found in the input pcap files. Write a hash table with chaining and choose an appropriate table size. Third party libraries are not allowed. **You must write the hash table code yourself**. The hash key should be based on the 4-tuple <src-ip, dst-ip, src-port, dst-port> (see hints below). For each new connection, as defined by seeing the first `SYN` packet, you should create a new entry in the hash table. As additional packets arrive, you should maintain a count of the number of packets in that connection, as well as the state of the connection. For the purpose of this exercise, you must use a single entry for both sides of the TCP conversation. In other words, a packet going from host A:port A to host B:port B should be considered part of the same flow as a packet going from host B:port B to host A:port A.

For a connection to be considered fully closed, a `FIN` packet must be sent in **both** directions. The connection-closing handshake is considered complete on the first `FIN` packet sent by the non-initiator. For example, if host A sends a `FIN` to B to initiate connection closure, the connection is not considered closed until B has sent its first `FIN` packet. Note that host A can retransmit `FIN`s; the connection is not considered closed until B sends its first `FIN`. Packets seen after connection closure should be ignored. However, a subsequent `SYN` seen for the same 4-tuple after connection closure should be treated as the start of a new connection.

Once a connection is closed, you should immediately output the timestamp of the `FIN` packet that caused the connection to close, along with the connection 4-tuple and the number of packets seen on that connection. The number of packets seen include every packet from the `SYN` to the `FIN` that completed the connection closure. Subsequently, you should free the entry corresponding to that particular connection in order to conserve memory, while being careful not to lose track of any other flow entries in the connection chain.

Your program should log to stdout in the following format:

```
hh:mm:ss.us Flow src-ip:src-port -> dst-ip:dst-port second fin after N
packets.
```

Sample output

The HW2 directory contains a sample pcap file called nytimes.pcap, which consists of packets obtained by capturing the activity of a browser loading the New York Times website. The result of running your program on that pcap file should look like this (the program output is truncated for brevity):

```
./hw2 -r nytimes.pcap
09:07:54.880872 Flow 10.148.1.8:59180 -> 170.149.161.130:80 second fin
after 16 packets
09:07:54.897589 Flow 10.148.1.8:59180 -> 170.149.161.130:80 second fin
after 17 packets
09:07:54.970421 Flow 10.148.1.8:59214 -> 170.149.161.130:80 second fin
after 63 packets
09:07:54.986803 Flow 10.148.1.8:59214 -> 170.149.161.130:80 second fin
after 64 packets
09:07:56.499824 Flow 10.148.1.8:59226 -> 170.149.161.130:80 second fin
after 23 packets
```

Note that there should be a newline (‘`\n`’) printed after every line.

We may provide more pcap files at a later time.

Assumptions

For this homework, you should make the following assumptions:

1. Packets will not be malformed.
2. Non-TCP packets should be ignored (eg. UDP, ICMP, DHCP, etc).
3. Incomplete flows that do not start with a `SYN` or have not fully transitioned into connection closure should be ignored.
4. Any stray packets that show up after connection closure (see definition above) should be ignored.
5. `FIN` packets may be retransmitted. Be careful to check for the exact `FIN` that causes the connection closure.

Submission

Fill in the README file with answers to the questions within. Be concise with your answers and keep your responses within the allocated space.

If you have additional source files, you need to update the Makefile accordingly to reflect these sources in the build process. Your Makefile needs to be correct – a broken Makefile that results in an unsuccessful compile will incur a penalty to your score.

Zip up only the README, Makefile and all your source/header files before submission. Do not turn in anything else – this includes pcap files, binaries, object files or temporary files. **Failure to follow these instructions precisely will result in a score penalty**.

When submitting, make sure to check that your files have been successfully uploaded onto CMS and that you have indeed uploaded the right set of files.

Grading

We will test the correctness of your program using pcap files that you will not have access to. Some of these pcap files may be quite large. You need to ensure that your program can run with reasonable performance on reasonably large pcap files without crashing or leaking memory. Your program should also not consume an inordinate amount of memory.

We will also use the valgrind tool to check for leaks or bad memory accesses, and apply a score penalty as appropriate.

Hints

1. Consider endianness. Byte ordering is different on the network than on Intel-based systems. See section 3 of the manpages on byte order ("man 3 byteorder") for macros you should use to cope with this. Nothing will work correctly until you get this part down.
2. Think carefully about how to make sure you can relate packets from both sides of the connection to the same flow entry.
3. For the flow hash function, we suggest XOR-ing together the two IP addresses and ports, then taking the modulus to the number of buckets of your hash table.

Homework updates / supplementary material

From time to time, we may post clarifying notes pertaining to the homework. It is your responsibility to monitor Piazza for these updates. In addition, while attendance for the supplementary lecture is optional, the material and slides covered are not optional. Please review all materials before attempting the homework.

Questions

If you have questions regarding the homework, please post them on Piazza.

For feedback and other non-homework related issues, please e-mail zteo@cs.cornell.edu.

Best of luck!