CSc 450/550: Computer Communications and Networks (Summer 2007)

Lab Project 3: A Simple Network Traffic Analyzer

Spec Out: July 6, 2007
Demo Due: July 25, 2007
Code Due: July 27, 2007

1 Introduction

In this project, students will build a simple network traffic analyzer to process given `tcpdump` files. The project will allow students to become familiar with the `pcap` programming library, and to better understand the design and implementation of TCP/IP protocols.

2 Background

2.1 `pcap` Library

`pcap` library provides a platform-independent interface to capturing packets from network interfaces with traffic filters when necessary. `pcap` can also dump captured packets to files in `pcap` format. In addition, `pcap` library can be used to read packets from `pcap` files.

To read packets from binary `tcpdump` files in `pcap` format, the following `pcap` library calls are often used.

- `pcap_open_offline()`: to open a `pcap` file and create a capture session
- `pcap_dispatch()` or `pcap_loop()`: to read and process packets with callback function
- `pcap_next()` or `pcap_next_ex()`: to read and process the next packet
- `pcap_compile()`: to create a traffic filter according to the given expression string
- `pcap_setfilter()`: to apply the traffic filter
- `pcap_freecode()`: to release the resource allocated for the traffic filter
- `pcap_close()`: to close the capture session

To use `pcap` library, you need to `#include <pcap.h>` and link with `-lpcap`
For more information on `pcap`, please `man pcap` and check `http://www.tcpdump.org/pcap.htm`
For a sample program using `pcap`, please see `http://www.tcpdump.org/sniffex.c`

2.2 Filter Expression

`pcap` supports traffic filters to be applied when capturing or reading packets; with traffic filters, `pcap` appears to only capture specific packets on the network interface or read them from the capture file. Users can describe traffic filters in expression strings, which are compiled by `pcap`.

The following expression strings are often used:

- `host`: host address
- `port`: port number
- `proto`: network protocol

Certain abbreviations of the expression strings are allowed. For example, ‘‘`tcp port 80 and host 192.168.0.1`’’ specifies TCP packets with source or destination port number 80 and source or destination host address 192.168.0.1

For more information on expression strings, `man tcpdump` and check the “expression” section.

2.3 Header Definition

Common network protocol headers are defined in the following header files:

- `/usr/include/net/ethernet.h`: Ethernet header
- `/usr/include/netinet/ether.h`: Ethernet MAC address format conversion
- `/usr/include/netinet/ip.h`: IP header
- `/usr/include/arpa/inet.h`: IP address format conversion
- `/usr/include/netinet/ip_icmp.h`: ICMP header
- `/usr/include/netinet/tcp.h`: TCP header
- `/usr/include/netinet/udp.h`: UDP header

Be aware that some protocol headers may contain header options of variable length.

3 Design

Students will have the freedom of designing their simple network traffic analyzer to be implemented in C or C++ with `pcap` library. The following is just a possible design.

The analyzer can first open the `tcpdump` file and create a `pcap` session, if no error occurs. Then the analyzer compiles the expression string, if necessary, and applies the traffic filter to the `pcap` session. The analyzer should maintain some data structures to collect certain statistics about the TCP connections in the `tcpdump` file, the TCP packets for a particular TCP connection, or a particular TCP packet, when reading and processing packets after applying the filter.

4 Requirements

4.1 Basic Features

Basic features are required in all implementations in order to get the full marks for this lab project.

4.1.1 TCP connections summary

The network traffic analyzer should print out the summary information about the TCP connections in the given `tcpdump` file, when being invoked as:

```
./nta <tcpdump_filename>
```

If being successfully invoked, the network traffic analyzer should print out the TCP connections summary in the follow format:

```
conn_num: init_IP:init_port init_pkts rSF - resp_IP:resp_port resp_pkts rSf
conn_num: init_IP:init_port init_pkts rSF - resp_IP:resp_port resp_pkts rSF
...
conn_num: init_IP:init_port init_pkts Rsf - resp_IP:resp_port resp_pkts Rsf
```

That is,

- `conn_num`: connection number, incrementing from 1 and indicating the order of the first packet of the connections in the `tcpdump` file
- `init_IP`: the IP address of the TCP connection initiator. The connection initiator is the TCP endpoint sending the first connection establishment packet with only SYN flag set. If no such packets are observed for a connection, the initiator is assumed to be the source endpoint of the first packet of the connection
- `init_port`: the port number of the TCP connection initiator
- `init_pkts`: the number of packets sent by the TCP connection initiator
- `rSF`: connection management flags (i.e., reset, synchronization, and finish, respectively) sent by the initiator throughout the connection , and capitalized letters indicating observed flags
- `resp_IP`: the IP address of the TCP connection responder
- `resp_port`: the port number of the TCP connection responder
- `resp_pkts`: the number of packets sent by the TCP connection responder
- `rSf`: connection management flags (i.e., reset, synchronization, and finish, respectively) sent by the responder throughout the connection, and capitalized letters indicating observed flags

4.1.2 TCP connection details

The network traffic analyzer should print out the detailed information about a particular TCP connection, when being invoked as:

```
./nta <tcpdump_filename> <init_IP> <init_port> <resp_IP> <resp_port>
```

If being successfully invoked, the network traffic analyzer should print out the detailed information about the specified TCP connection in the following format:

```
pkt_num: src_IP:src_port > dst_IP:dst_port uaprSf seqno ackno win
pkt_num: src_IP:src_port > dst_IP:dst_port uAprSf seqno ackno win
...
pkt_num: src_IP:src_port > dst_IP:dst_port uAprsf seqno ackno win
```

That is,

- `pkt_num`: packet number, incrementing from 1 and indicating the order of the packet observed in the connection
- `src_IP`: source IP address
- `src_port`: source port number
- `dst_IP`: destination IP address
- `dst_port`: destination port number
- `uaprSf`: TCP flags, capitalized letters indicating flags that are set in this packet
- `seqno`: TCP sequence number, relative from the first sequence number of this connection
- `ackno`: relative TCP acknowledgment number
- `win`: TCP window size

4.1.3 Packet round-trip time

The network traffic analyzer should print out the round-trip time information for a particular TCP packet, when being invoked as:

```
./nta <tcpdump_filename> <src_IP> <src_port> <dst_IP> <dst_port> <seqno>
```

If being successfully invoked, the network traffic analyzer should print out the round-trip information about the specified TCP packet in the following format:

```
dat_pkt_time src_IP:src_port > dst_IP:dst_port uaprsf seqno ackno win
ack_pkt_time src_IP:src_port > dst_IP:dst_port uAprsf seqno ackno win
```

That is,

- `dat_pkt_time`: the time when the data packet is captured, in the format of `HH:MM:SS:microsecond`
- `ack_pkt_time`: the time when the corresponding acknowledgment packet is captured

Be aware that TCP adopts cumulative acknowledgment and may adopt delayed acknowledgment, and data packets might get retransmitted.

4.2 Bonus Features

The simple network traffic analyzer only reveals very little and coarse information about the traffic contained in the `tcpdump` file, e.g., TCP connections and packets. There is much more information can be revealed and inferred, e.g., the round-trip timeout value, congestion window at the sender.

If you want to design and implement a bonus feature, you should contact the course instructor for permission at least one week before the code due date and clearly indicate the feature during project demo and in code submission. The credit for correctly implemented bonus features will not exceed 20% of the full marks for this lab project.

5 Self-Testing

A `tcpdump` file will be provided for your self-testing. You can use Ethereal (Wireshack), TCPdump, or TCPtrace to examine the `tcpdump` file, and test your design and implementation.

However, your traffic analyzer might be tested against different `tcpdump` files during project demo and code inspection.

6 Demonstration

Your traffic analyzer should be demonstrated in the lab section for which you have registered on the demo due date. **Lab projects not demonstrated will have their code submission not marked. It is expected that your lab project be working at the time of demonstration.** During the project demo, lab instructors will go through a demo checklist together with the student, and then provide the checklist to the student. Students will have the chance to improve their design and implementation until the code due date.

The demo is intended to allow students to demonstrate their projects and to help them improve their design and implementation, not to be coding or debugging assistance, and only focuses on required features. There is no guarantee on the correctness and grade of the project, which can only be determined after the code inspection.

7 Submission

The entire lab project, including the code and documentation, should be submitted electrically through `csc450l` (`l` is for letter L) at `http://www.csc.uvic.ca/~submit/index.cgi` on or before the code due date.

Only the source code (including header files and `Makefile`) and documentation (including `readme.txt`) should be included in a single `tar.gz` file to be submitted. No object or binary files are included in the submission. If `directory` is your project directory, to create such a gzipped tarball, you can

```
cd direcoty
tar -zcvf p3.tar.gz .
```

This packing and naming convention should be strictly followed to allow your submission to be properly located for grading.

In `directory`, you need to include a `Makefile`, which compiles and builds the final binary executable (`nta`) automatically by typing

`make`

The same `Makefile` also removes all object and executable files when you type in

`make clean`

All projects will be tested on `linux.csc.uvic.ca`

In `directory`, you need to include `readme.txt` in plain text format, which contains your student number, registered lab section and a brief description of your code structure.

The code itself should be sufficiently self-documented. For more information on acceptable coding style, please see [1].

IMPORTANT: All submitted work should be yours. If you have used anything out there, even a small component in your design and implementation, you should credit and reference properly, and your contribution can be determined accordingly. For academic integrity policies, please see [2].

8 Marking

This lab project is worth 15% in the final grade of this course for CSc 450 students, and 10% for CSc 550 students.

For mark posting and appeal policies, please see the official course outline at [2].

References

[1] `http://www.csc.uvic.ca/~csc450l/references/Code-Style.html`

[2] `http://courses.seng.engr.uvic.ca/courses/2007/summer/csc/450`