Project #2 : Traceroute Next Generation

CS155: Computer and Network Security
Due: May 4th, by 11:59pm (Part 1),
Due: May 11th, by 11:59pm (Part 2)

Spring 2009

Contents

1 Introduction

Traceroute is one of the key tools used for network troubleshooting and scouting. It has been available since networking's early days. Because traceroute is based on TTL header modification, it crafts its own network packets.

The goal of this project is to re-implement a traceroute with cutting edge techniques designed to improve its scouting capabilities and bypass SPI firewalls. In this project you will learn about packet injection and sniffing. The project also teaches about more subtle topics such as network latency, packet filtering, and Q.O.S.

1.1 Instructions

This project has to be completed sequentially as each part depends on the previous ones. You are required to work in groups of at most 2 people (one-person teams, while allowed, are discouraged). The required coding language is C. You should consider using libnet [1] and libpcap [2], however using raw sockets is permitted as well.

Note that the exercises become progressively more difficult, with Exercise 5 likely to consume the greatest amount of effort. Please plan your work accordingly: there is no penalty for early submission.

1.2 Submission and Grading

The project is due in two parts. The first part includes exercises 1 and 2, and the second part should be fully functional, including exercises 1, 2, 3, 4, and 5. The first part will be worth 40% of the project grade, which means that exercises 1 and 2 will be graded entirely based on your part 1 submission. In all, exercises will have equal weight, about 20% of the project grade each.

Submission is by email to **cs155ta@cs.stanford.edu**, and is due by 11:59pm on **May 4th** for Part 1 and by 11:59pm on **May 11th** for Part 2. Provide a tgz archive that contains your code. When extracted, you project has to be in a directory named project2_name1[_name2] where name1 (and name2 when present) is(are) the last name(s) of the student(s) it the team. The subject line of the email needs to be "CS155 project2". Please include in the body of the e-mail the name of each member of the team, as well as their email addresses.

Your code will be compiled under Linux Ubuntu with the lastest version of libpcap and libnet installed. To compile your project, the script will use the command `make`, executed in the project directory specified above. Then it will launch our tests, using an executable name "traceng"—you have to make sure your make file results in this executable name, in the same project directory.

Our tests will involve tracing from Stanford to **www.apple.com**, **www.google.de**, **www.amazon.co.uk**. We will test every feature you implemented against these hosts.

Make sure that the command line options and the output format are correctly implemented as they will be used by our script during the grading process. Specific examples of how we are going to test your executable are provided later in this document, for each exercise.

Your project directory should also contain a file called README which is an ASCII text file describing the highlights of your solution to each exercise—anything special that you did, information required in the specific exercise, what is the most interesting thing you learned in the exercise, and anything else we need to take into account. Plan on a couple of paragraphs per exercise (excluding exercise-specific info). Your README could account for as much as 20% of the project grade, so you need to take the time and write meaningful, concise answers.

Along with README, include a log file called LOG, that contains the output of your implementation for all the functionality you have implemented. This will be helpful as we grade your work in case the network is volatile, or due to differences in your environment and ours. Note that the output from each exercise starts with a dump of the arguments used—this is so a log containing many runs will still be readable.

Feel free to include network traces that you captured to demonstrate your working implementation: while not required, they will help us better evaluate your project submission. Your .tgz submission should not exceed 2MB.

1.3 Exercise Overview

There are five exercises in the project:

- **Exercise 1** "Standard Traceroute": covered in Section 2.
- **Exercise 2** "Versatile Traceroute": The goal is to implement various types of probes so your tool is able to switch when a router does not respond to the current type of probe or when packets are lost (Section 3).
- **Exercise 3** "Statistics Reporting": The statistics are similar in spirit to those reported by mtr [4] (Section 4).
- **Exercise 4** "Path Diagnostics": The goal is to be able to analyze network problems and load-balancing routing (Section 5).
- **Exercise 5** "Advanced Techniques": Finally you will implement a set of techniques designed to handle packet filtering devices such as firewalls (Section 6). These techniques are inspired by Firewalk [3] as well as the established traceroute technique introduced by Michal Zalewski.

1.4 Recommended Reading

You need a really good understanding of the TCP/IP protocol stack to complete this project. If you want a good book about TCP/IP, we suggest "TCP/IP Illustrated, Volume 1" [6] by R. Stevens. If you want a book that describes in detail how libpcap and libnet work, use the book by the author of libnet, Mike Schiffman, called "Building Open Source Network Security Tools: Components and Techniques" [5].

2 Standard Traceroute

The goal of this first part is to make you familiar with the notion of packet injection and packet sniffing. Accordingly you will have to code a traceroute that sends ICMP packets and outputs the results.

- Look at libnet [1], libpcap [2] and raw socket APIs to understand the differences.
- Send a simple ICMP echo packet on the network. The destination IP is passed as a command line argument **-d**. You should test that the packet is correctly sent and answered by using tcpdump [2].
- Implement the sniffing process in your code. Use two threads: one for sending packets, one for reading the responses from the network.
- Implement a TTL increment loop, with TTL starting at 1. Increment TTL until a response from the targeted host is received.
- By default your program uses the first network interface. However add the ability to select another one using the command switch **-i**, e.g. as "-i eth0".
- Add a file output option **-f** (always append to the file, do not clear its contents).

2.1 Sample Invocations

- `traceng -f log.txt -d <targetHostName>`
- `traceng -d <targetHostName> -i <ifSrc>`

2.2 Sample Output

```
<arguments f="log.txt" d="72.32.146.3" i="eth1">
</arguments>
<traceroute>
<host distance="1"
      ip="192.168.1.1"
      probeType="ICMP">
</host>
<host distance="2"
      ip="72.32.146.3"
      probeType="ICMP">
</host>
</traceroute>
```

3 Versatile Traceroute

The goal of this part is to add probe flexibility to accommodate router and firewall policy.

- Add the ability to send TCP **-T** probes with various ports at the source and destination **-sp and -dp**.
- Add the ability to send UDP **-U** probes with various ports at the source and destination **-sp and -dp**.
- Add the ability to send ICMP **-I** probes with various ICMP codes **-c**. (Note that the implementation for Exercise 1 should be equivalent to using "-I -c 8".

3.1 Sample Invocations

- `traceng -d <targetHostName> -sp <portSrc> -dp <portDest> -T`
- `traceng -U -sp <portSrc> -i <ifSrc> -d <targetHostName> -dp <portDest>`
- `traceng -I -d <targetHostName> -c <code>`

3.2 Sample Output

```
<arguments ...>
</arguments>
<traceroute>
<host distance="1"
      ip="192.168.1.1"
      probeType="TCP/UDP/ICMP"
      port="42">
</host>
<host distance="2"
      ip="72.32.146.3"
      probeType="TCP/UDP/ICMP"
      port="42">
</host>
</traceroute>
```

4 Statistics Reporting

The goal of this section is to extend traceng to be a tool for gathering statistics. This is similar in spirit to mtr [4]. Once the path to the target host is determined the tool starts to "ping" each router that belongs to the path, and to gather statistics. This section's objectives are:

- Modify the probe engine to ping each router on a regular basis.
- Display probe results on the standard output.
- Add latency evolution information.
- Enable statistics reporting when the **-S** option is provided (otherwise, the behavior is as before). The option takes as an argument the number of iterations you have to run.

4.1 Sample Invocations

- `traceng -S 10 -f t.log -T -d <targetHostName> -sp <portSrc> -dp <portDest>`
- `traceng -d <targetHostName> -sp <portSrc> -dp <portDest> -S 10 -U`
- `traceng -d <targetHostName> -I -c <code> -S 2 -f my_stats.log`

4.2 Sample Output

```
<arguments ...>
</arguments>
<traceroute>
<host distance="1"
      ip="192.168.1.1"
      probeType="TCP/UDP/ICMP"
      port="42"
      pktSent="1"
      pktLost="0"
      minTime="2.300"
      maxTime="2.300"
      avgTime="2.300"
      iteration="1">
</host>
<host distance="2"
      ip="72.32.146.3"
      probeType="TCP/UDP/ICMP"
      port="42"
      pktSent="1"
      pktLost="0"
      minTime="7.120"
      maxTime="7.120"
      avgTime="7.120"
      iteration="1">
</host>
...
...
</traceoute>
```

```
<traceroute>
<host distance="1"
      ip="192.168.1.1"
      probeType="TCP/UDP/ICMP"
      port="42"
      pktSent="10"
      pktLost="0"
      minTime="1.101"
      maxTime="5.200"
      avgTime="1.327"
      iteration="10">
</host>
<host distance="2"
      ip="72.32.146.3"
      probeType="TCP/UDP/ICMP"
      port="42"
      pktSent="10"
      pktLost="1"
      minTime="5.051"
      maxTime="10.311"
      avgTime="8.732"
      iteration="10">
</host>
.....
</traceroute>
```

Add one line per host per probe type per ping cycle. Note that the "per host" and "per probe type" requirements will become relevant in the next exercises. So far, you have only dealt with a single host, and a single probe type per invocation of traceng.

5 Path Diagnostics

Here are the three questions that your tool needs to address:

- Packet loss: is the loss due to a rate limit on the router, or congestion?
- Route load balancing: is there a load balancer between you and the targeted host?
- Does your targeted host use multiple servers? How does the load balancing work?

All of this functionality is controlled by the **-G** option. If it is missing, the behavior of traceng is unchanged.

5.1 Loss

To analyze the cause for packet loss you will implement an algorithm that performs the following tests and analyzes their results when loss is detected on a router.

- Try to change the delay between probes for this particular router.
- Try to change the probe protocol.
- Try to change the packet TOS.
- Try to probe the router and its successor in a short interval to see if both show loss.
- Do this also for the router and its predecessor.
- Try multiple protocols.

Add the resulting interpretation to the output of the tool. The format for the output is

```
<diag type="loss">your interpretation</diag>
```

In the README file, explain your interpretation codes in detail. You are free to add other tests if they are needed.

5.2 Route Load Balancing

To detect route load balancing you will implement an algorithm that performs the following tests and analyzes their results. The goal is to discover whether the route is load balanced and how the load balancing is done.

- Run the path discovery with multiples protocols
- Run the path discovery with multiples ports source and destination.
- If the route is load balanced try to determine if the load balancing is done
  - by protocol
  - port destination
  - packets: this can be done by observing the packet repartition
  - the tuple port source / port destination
  - other?

Add the result interpretation to the output of the tool. The format for the file is

```
<diag type="routelb">your interpretation</diag>
```

In the README file, explain your interpretation codes in detail. You are free to add other tests if they are needed.

5.3 Server Load Balancing

Some host names may resolve to multiple servers. One such example is www.google.com. In case of multiple servers per host name, your tool needs to trace the route to all servers, and gather (and report) statistics for each.

Recall that in Section 4 you were required to output one line per host. In case of multiples servers for the same host, you have to treat each server as its own host, and provide separate lines for each IP and at each iteration ping them separately to compute their latency. *HINT: Look at the command* `host` *if you want to understand how to find out all server IPs for a host name.*

5.4 Sample Invocations

- `traceng -f t.log -G -T www.google.com -sp <portSrc> -dp <portDest>`

5.5 Sample Output

```
<traceroute>
<arguments ...>
</arguments>
<host distance="1"
      ip="192.168.1.1"
      probeType="TCP/UDP/ICMP"
      port="42">
<host distance="2"
      ip="72.3.12.198"
      probeType="TCP/UDP/ICMP"
      port="42">
<host distance="3"
      ip="164.32.9.8"
      probeType="TCP/UDP/ICMP"
      port="42">
<host distance="4"
      ip="164.32.9.1"
      probeType="TCP/UDP/ICMP"
      port="42">
<host distance="4"
      ip="164.32.9.2"
      probeType="TCP/UDP/ICMP"
      port="42">
<host distance="5"
      ip="74.125.19.99"
      probeType="TCP/UDP/ICMP"
      port="42">
<host distance="5"
      ip="74.125.19.103"
      probeType="TCP/UDP/ICMP"
      port="42">
<host distance="5"
      ip="74.125.19.104"
      probeType="TCP/UDP/ICMP"
      port="42">
<host distance="5"
      ip="74.125.19.147"
      probeType="TCP/UDP/ICMP"
      port="42">
</host>
</traceroute>
<diag type="loss">Packets dropped by 164.32.9.8 due to congestion.</diag>
<diag type="routelb">Route load balancing detected for TCP traffic after
IP 164.32.9.8 at hop 4.</diag>
```

6 Firewall Handling

6.1 Firewalk

Implement the Firewalking **-F** algorithm [3]. That enumerates firewall ACL by targeting a host **-d** which is behind the firewall **-g** and trying to reach it with various TCP or UDP packet with a TTL one greater than the targeted firewall. If there is a response, then the traffic is considered allowed.

6.1.1 Sample Invocation

```
traceng -F -g <gatewayip> -d <hostbehingthegateway> -f my_stats.log
```

6.1.2 Sample Output

```
<arguments ...>
</arguments>
<firewalk>
  <port num="1" proto="TCP">open</proto>
  <port num="2" proto="TCP">filtered</proto>
</firewalk>
```

6.2 Established Method

Implement the established traceroute technique **-E** used to by pass stateful packet filters. This technique works as a normal TCP traceroute, excepts that, before tracerouting, a TCP connection is established with the destination host. You are not allowed to use the socket library for TCP: you have to implement the three-way handshake using hand-crafted packets. Make sure your ISN is random.

6.2.1 Sample Invocation

```
traceng -E -d <targetHostName> -sp <portSrc> -dp <portDest> -f my_stats.log
```

6.2.2 Sample Output

```
<arguments ...>
</arguments>
<established>
  <port num="1" proto="TCP">open</proto>
  <port num="2" proto="TCP">filtered</proto>
</established>
```

6.3 Ghost Traceroute

This part allows you to understand better the link between ARP addresses and IP addresses, and how LANs work.

Implement a ghost traceroute which allows you to traceroute from a spoofed IP. The key is to handle ARP packets, as you need to see the responses to whatever you send, and switches along the way need to know to send those responses to you. Make sure that the IP you spoof is not already in use. Use the **-P** option to activate this functionality, along with passing the requested source IP in the **-s** argument.

6.3.1 Sample Invocation

```
traceng -P -s <sourceIP> -d <targetHostName> -sp <portSrc> -dp <portDest> -f my_stats.log
```

6.3.2 Sample Output

```
<arguments ...>
</arguments>
<traceroute>
<host distance="1"
      ip="192.168.1.1"
      probeType="TCP/UDP/ICMP"
      port="42">
</host>
<host distance="2"
      ip="72.32.146.3"
      probeType="TCP/UDP/ICMP"
      port="42">
</host>
</traceroute>
```

NOTE: Even though this output is indistinguishable from that in Exercise 2, tcpdump will capture the different IP that you are using, as well as your ARP traffic.

7 Wrap-Up

Note that you can assume the -S, -G, -E, -F, and -P options are mutually exclusive: we will not use more than one of them at a time, even though there are ways to meaningfully mix these options.

References

[1] Libnet homepage: http://libnet.sourceforge.net/.

[2] Tcpdump and libpcap official homepage: http://www.tcpdump.org/.

[3] David Goldsmith and Michael Schiffman. Firewalk http://www.packetfactory.net/firewalk/firewalk-final.pdf, 1998.

[4] Kimball Matt. Mtr: My traceroute (http://en.wikipedia.org/wiki/mtr_(my_traceroute)).

[5] Schiffman Mike. *Building Open Source Network Security Tools: Components and Techniques*. John Wiley, Nov. 2002.

[6] W. Richard Stevens. *TCP/IP Illustrated, Volume 1 : The Protocols*. Addison-Wesley, 1994.