BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

MEASUREMENT OF PACKET LOSS AND RTT

MĚŘENÍ ZTRÁTOVOSTI A RTT

PROJECT DOCUMENTATION
PROJEKTOVÁ DOKUMENTACE

AUTHOR **GABRIELA PACÁKOVÁ**
AUTOR PRÁCE

LOGIN **XPACAK01**

BRNO 2017

Contents

Chapter 1

Preface

This document serves as a project documentation for Network Applications and Network Administration project called Measurement of Packet Loss and RTT. The basic purpose of the described application is to test a number of chosen hosts in the network by sending echo packets and waiting for the replies from the hosts. The earlier chapters will describe the concept of the problematics, whereas the latter chapters will include a mildly detailed description of implementation and a few examples of the usage of the described application.

Chapter 2

Basic Concept

In order to grab the basic concept of this application, there are a few utilities and notions the reader must be able to distinguish between. The reader should already be familiar with the OSI model and have the basic understanding of the seven layers. The application mostly operates on layer two, three and four. The application supports both IPv4 and IPv6 families.

2.1 Protocol ICMP(v6)

ICMP[1] and ICMPv6[2] protocols are Internet Message Control Protocols for IPv4 and IPv6 respectively. Their purpose is to send control messages with the appropriate answer code, to reply to requests. It is compulsory for each host to have these implemented and to reply appropriately. *ICMP is actually an integral part of IP, and must be implemented by every IP module.*[1]

There are several types of ICMP messages. We'll be concerned only about the `ECHO` and `ECHO_REPLY` types. To test a certain host, the application is going to construct an ICMP header with an `ECHO` type of message inside, and after that, it is going to send the packet including the aforementioned ICMP header to the host. Then, it will wait a certain amount of time for the host to reply by an `ECHO_REPLY` type of message. This is very similar to how Ping utility works.

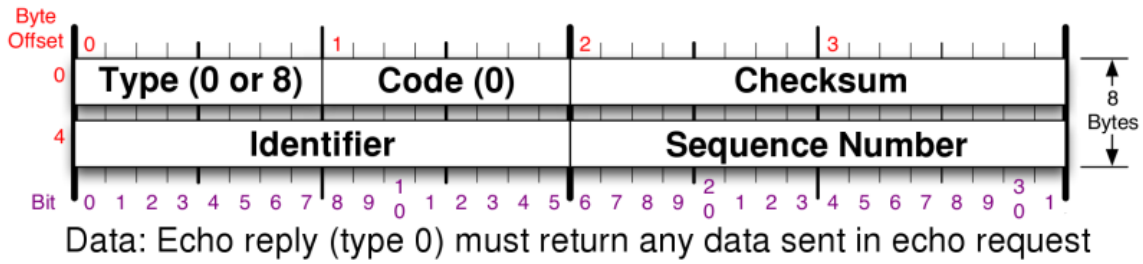

Figure 2.1: ICMP header. http://www.nmap.org/

[1] RFC792 https://tools.ietf.org/html/rfc792

2.2 Parallel Ping

Our application, as opposed to a simple Ping utility, is able to test a large number of hosts all at the same time, among other things. That means it is capable of sending `ECHO` packets and receiving `ECHO_REPLY` packets from multiple hosts parallelly. This allows us to gather statistics about the hosts more effectively than testing them one by one.

2.3 Round Trip Time (RTT)

Round-trip time (RTT) is the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledgment of that signal to be received.[2] In this case, it is basically the time difference of the `ECHO` message sent time and the `ECHO_REPLY` message received time. This time difference tells us how fast can the host reply to our requests.

2.4 Packet Loss And Statistics

There are a few types of statistics the application will provide. First and foremost, every `-t` seconds, it will output the number of lost packets and their percentage among all packets sent to each tested host. The user can adjust the time interval in the options.

Secondly, every hour it will output a statistics based on the RTT of all the packets sent to the host. The statistics include minimum, maximum and average RTT values in addition to the standard deviation of the RTTs.

The last part can be turned on by an optional argument `-r`. It will set a RTT threshold time for the packets. Each packet that wasn't lost and its RTT is greater than the threshold will be reported every `-t` seconds with the percentage mentioned earlier.

2.5 UDP Protocol

According to specification, a part of this application is supposed to be able to send and receive UDP packets with random data and also act as a concurrent UDP server that listens for UDP packets and replies with the same content back to the senders, if the user sets it this way. The detailed description will be discussed in the implementation part.

[2] Definition of RTT - Wikipedia https://en.wikipedia.org/wiki/Round-trip_delay_time

Chapter 3

Implementation

The application's name is`testovac`. It is implemented using `C language` and is intended to work under various Linux distributions. The source code is divided into several modules which I'm going to describe in the following sections.

3.1 Modules

All of the following modules have a `.c` source and a `.h` header file.

3.1.1 testovac

This module contains the `main()` function. It controls the basic program flow, calculates the intervals for the next packet to be sent, measures RTT and provides data to calculate statistics. It's heavily dependent on the other modules.

3.1.2 argparse

This module handles argument parsing using function `parse_args()`. It's a custom made argument processing function. Apart from parsing all the parameters, it checks if the nodes that the user provided are valid. That means it calls the `getaddrinfo()` function that attempts to translate every node, before any packets are prepared to be sent.

3.1.3 utils

This module contains some useful utility functions such as the translation from host name to IP, reverse translation from IP to host name and some other functions that make use of `gettimeofday()`.

3.1.4 memops

This modules handles memory operations and memory allocations. In addition it handles the linked list functions. Linked list is used by the parent program to keep track of all the hosts that it's testing.

3.1.5 socket

This module contains functions to handle socket creation, setting socket options using `setsockopt()`, also contains a function `socket_timeout()` that uses `select()` to establish a proper timeout.

3.1.6 icmp

This module contains all the operations needed to prepare an ICMP echo header, to prepare a packet, calculate its checksum, send the packet and wait for ICMP reply.

3.1.7 udp

This module contains all the UDP related functions, creation of the UDP packet, and also a function to listen for UDP packets and send a reply to them.

3.2 Core

3.2.1 Parallel Testing

As already mentioned before, the program is written so it can test multiple hosts at the same time. There are many alternatives on how to achieve this. I decided to implement this by forking. A parent program parses all the nodes given by the user input, and makes a linked list for all the valid ones. For each node, a `fork()` is called and a child is created. Parent continues on to make children while there are still nodes in the list, and at the same time, created children start to test their given host.

When parent is done forking, it waits for the children to terminate before terminating itself. The program core for children is an infinite loop, as the specification doesn't include when to stop testing the host. The child processes will continue on to test hosts until `SIGINT` from the user is caught. After that the children will attempt to terminate gracefully, printing out statistics just before termination.

If the user also included `-l` parameter on use, the parent will start listening for UDP packets on a port given by `-l`. For each incoming connection, the parent will again `fork()` itself and their child will reply and terminate. This allows the application to reply on multiple connections all at the same time.

3.2.2 Sockets And ICMP Header

The application uses RAW sockets for ICMP, and therefore the user must have root privileges to run it. The socket is created on the `IPPROTO_ICMP` layer, so it's already encapsulated in the IP layer. The packet construction is fairly simple. The ICMP header is filled out and checksum is calculated using `ip_checksum()`. If some random data are to be included, they are generated using `generate_rnd_data()`, added after the header, and checksum is calculated after that.

3.2.3 Calculating RTT

The application uses two ways to calculate RTT. If the size of the data in the packet allows a timestamp of the time sent to be inserted, it's inserted as the first `sizeof(double)`[1] bytes.

[1] 8 bytes https://en.wikipedia.org/wiki/Double-precision_floating-point_format

Otherwise, the time sent is measured right after the `send_icmp()` or `send_udp()` operation and saved into a variable. The RTT is then calculated either by decoding the time stamp from the reply and subtracting the time when it was received, or just two variables are subtracted to get the final RTT. The latter one is less precise.

3.2.4 Calculating MDEV

The `mdev` could mean either mean deviation or the standard deviation. As it would be problematic to keep track of all the packets RTT's separately and add them all together in the end, I decided to take a look at how Ping utility calculates mdev. It calculates the standard deviation progressively, after each packet the RTT of that packet is processed. This way the program doesn't have to remember an extensive amount of RTT values.

```
double tsum, tsum2, tmdev, rtt;
int nreceived;
...
/*Processing the RTT after each packet */
tsum += rtt;
tsum2 += rtt * rtt;
...
/* Calculating the mdev before we print it out. */
tsum /= nreceived;
tsum2/= nreceived;
tmdev = sqrt(tsum2 - tsum * tsum);
```

3.3 Known Bugs And Limitations

3.3.1 LIMITATION: Packet Size

The variable data size that can be set up by `-s` is set to a maximum of MTU which is 1500 bytes over Ethernet. That includes all the headers, so the maximum effective size of the data that can be sent is `1456` bytes using IPv4 and `1436` bytes using IPv6. If the user sets the data to be larger, it will be adjusted appropriately according to description above.

3.3.2 BUG: Exiting The Program

The application will terminate when CTRL+C is pressed, but sometimes, when testing a large number of hosts, the parent and child processes don't terminate in the right order. In that case, CTRL+C must be pressed again.

3.3.3 BUG: UDP Packets

If `-u` and `-p` is set by the user and the host is not responding to UDP packets sending UDP packets back, the application might send an excessive amount of packets to that port in a short amount of time.

Chapter 4

Examples Of Usage

Usage: ./testovac [options] [target list...]
Note: Requires root access if you're not using UDP (-u)
Options:
`[-h]` - displays this help
`[-u]` - UDP protocol will be used for testing
`[-t <interval> s]` interval after which the packet loss statistic is displayed, default 300
`[-i <interval> ms]` wait interval milliseconds between sending each packet, default 100
`[-p <port>]` is used with UDP, sets which port to use to send UDP packets to
`[-l <port>]` program will listen on this port and reply on incoming UDP messages
`[-s <size>]` size of packet data, ICMP default 56 bytes, UDP default 64 bytes
`[-r <value> ms]` RTT treshold. Packets over treshold are reported if -r is set
`[-w <timeout> s]` sets the timeout for the first packet, after that it's 2xRTT
`[-v]` is verbose mode, displays received packets (similar to ping)
Target list:
`<target>` - IPv4 / IPv6 / hostname of the target, multiple targets must be separated by whitespace

```
                                                sudo ./testovac -v -i 1000 -r 5 www.google.com www.yahoo.com
2017-11-19 14:16:56.25 56 bytes from prg03s05-in-f196.1e100.net (172.217.23.196) time=24.51 ms
2017-11-19 14:16:56.26 56 bytes from media-router-fp1.prod.media.vip.ir2.yahoo.com (188.125.80.144) time=64.30 ms
2017-11-19 14:16:57.18 56 bytes from media-router-fp1.prod.media.vip.ir2.yahoo.com (188.125.80.144) time=54.09 ms
2017-11-19 14:16:57.21 56 bytes from prg03s05-in-f196.1e100.net (172.217.23.196) time=13.76 ms
2017-11-19 14:16:58.20 56 bytes from prg03s05-in-f196.1e100.net (172.217.23.196) time=17.08 ms
2017-11-19 14:16:59.16 56 bytes from media-router-fp1.prod.media.vip.ir2.yahoo.com (188.125.80.144) time=87.25 ms
2017-11-19 14:16:59.19 56 bytes from prg03s05-in-f196.1e100.net (172.217.23.196) time=24.73 ms
^C2017-11-19 14:16:59.41 media-router-fp1.prod.media.vip.ir2.yahoo.com: 25.000% packet loss, 1 packet lost
2017-11-19 14:16:59.41 media-router-fp1.prod.media.vip.ir2.yahoo.com: 75.000% (3) packets exceeded RTT treshold 5.0 ms
2017-11-19 14:16:59.41 prg03s05-in-f196.1e100.net: 0.000% packet loss, 0 packet lost
2017-11-19 14:16:59.41 media-router-fp1.prod.media.vip.ir2.yahoo.com: 25.000% packet loss, rtt min/max/avg/mdev 54.090/87.254/68.548/13.868 ms
2017-11-19 14:16:59.41 prg03s05-in-f196.1e100.net: 100.000% (4) packets exceeded RTT treshold 5.0 ms
2017-11-19 14:16:59.41 prg03s05-in-f196.1e100.net: 0.000% packet loss, rtt min/max/avg/mdev 13.756/24.730/20.020/4.750 ms
```

Figure 4.1: Example of `testovac` usage.

Chapter 5

Conclusion

It could be stated that the application is an extended version of Ping utility. It can potentially be used to collect various data of the tested hosts, or network. The source files can be compiled using `Makefile`, which requires `gcc` to be installed on the system. The application was tested on lubuntu 16 and centOS7.

Bibliography

[1] Postel, J.: *RFC792*. https://tools.ietf.org/html/rfc792. September 1981.

[2] Society, T. I.: *RFC4443*. https://tools.ietf.org/html/rfc4443. March 2006.