CS542 PROJECT
PING COMMAND IMPLEMENTATION

May 6, 2022

Duan, Bin
Illinois Institute of Technology
Department of Computer Science

Contents

1 Implemented Algorithm

Ping command-ICMP echo request/reply python implementation.
Tasks:

- The sender sends out 2 ICMP requests constantly with the length of 256 bytes;
- Receiver sends back replies and print the requests (messages, not the statistics) it receives from the sender to a .txt file;
- When the sender receives replies, it prints the reply messages (not the statistics) on the screen and in a .txt file as well.

2 Running Environment

2.1 Workspace

- macOS Monterey
- Python 3.7.2

2.2 Python Libraries

- matplotlib
- os
- select
- signal
- socket
- struct
- sys
- time
- argparse

2.3 Command

```
sudo python -u ping.py
```

Note that, the code needs to be executed using root (i.e., with sudo).

3 Code Snippets

It is worth noting that we annotate each function and block with the source code "ping.py". Here, we list some important code snippets and their explanation for emphasis.

3.1 Ping Arguments

```
parser = argparse.ArgumentParser(description="Ping Command Python
                                    Implementation for CS542 Project.")

parser.add_argument("--hostname", type=str, default="www.google.com",
                                    help="Address to be pinged.")
parser.add_argument("--log_file", type=str, default="log.txt", help="
                                    Path to save ping log.")
parser.add_argument("--timeout", type=int, default=1000, help="timeout
                                    for receiving a ping (ms).")
parser.add_argument("--count", type=int, default=2, help="How many ICMP
                                     requests constantly.")
parser.add_argument("--packet_size", type=int, default=256, help="
                                    Packet size.")

opt = parser.parse_args()
print_options(opt)
```

We define common arguments for the ping command that user can easily alter the arguments for different scenarios. All arguments are annotated with "help" description, which should pretty straightforward for easy readability. All arguments will be print out into console.

3.2 Logging Class

The design of logging class is to record the statistics or important data (e.g., maximum round-trip time) when pinging. Our design follows:

```
def calc_cksum(src_str):
    """Calculate checksum"""
    total_count = int(len(src_str) / 2) * 2
    sum = 0

    for i in range(0, total_count, 2):
        if (sys.byteorder == "little"):
            low_byte = src_str[i]
            high_byte = src_str[i + 1]
        else:
            low_byte = src_str[i + 1]
            high_byte = src_str[i]
        sum = sum + high_byte * 256 + low_byte

    # Handle last byte if applicable (odd-number of bytes)
    if total_count < len(src_str):
        low_byte = src_str[len(src_str) - 1]
        sum += low_byte

    # Truncate sum to 32 bits
    sum &= 0xffffffff

    # Add high 16 bits to low 16 bits
    sum = (sum >> 16) + (sum & 0xffff)
    sum += (sum >> 16)
    answer = ~sum & 0xffff

    # Invert and truncate to 16 bits
    answer = socket.htons(answer)

    return answer
```

To calculate the correct checksum, we first make a dummy header with checknum=0, and then pad the segment into a multiple of 16 bits and computes the 16-bit checksum over the entire result. At the receiving sitem TCP software performs the same computation to verify that the segment arrived intact.

3.3 IP Address Translation

```
def to_ip(addr):
    """interpolate hostname to valid IPV4 address"""
```

```
    def is_valid_ip4_address(addr):
        parts = addr.split(".")
        if not len(parts) == 4:
            return False
        for part in parts:
            try:
                number = int(part)
            except ValueError:
                return False
            if number > 255:
                return False
        return True

    if is_valid_ip4_address(addr):
        return addr
    return socket.gethostbyname(addr)
```

We can use different representations for the hostname, for example, "www.google.com" and its physical ip address. We first validate whether it is valid IPV4 address and if not we convert the semantic hostname to its physical counterpart.

3.4 ICMP ECHO REQUEST

```
def send_one_ping(self, current_socket):
    """
    Send one ICMP ECHO_REQUEST
    """
    # Header is type (8), code (8), checksum (16), id (16), sequence (
                                            16)
    checksum = 0

    # Make a dummy header with a 0 checksum.
    header = struct.pack("!BBHHH", ICMP_ECHO, 0, checksum, self.own_id,
                                             self.seq_number)

    padBytes = []
    startVal = 0x42
    for i in range(startVal, startVal + (self.packet_size)):
        padBytes += [(i & 0xff)]  # Keep chars in the 0-255 range
    data = bytes(padBytes)

    # Calculate the checksum on the data and the dummy header.
```

```
    checksum = calc_cksum(header + data)

    header = struct.pack("!BBHHH", ICMP_ECHO, 0, checksum, self.own_id,
                                            self.seq_number)

    packet = header + data

    send_time = timer()

    try:
        current_socket.sendto(packet, (self.destination, 1))  # Port
                                                number is irrelevant for
                                                ICMP
    except socket.error as e:
        self.response.output.append("General failure (%s)" % (e.args[1]
                                                ))
        current_socket.close()
        return

    return send_time
```

Implmentation of ICMP Echo request message. We first create a dummy header with 0 checksum and then create a certain size packet sent to the ping destination.

3.5 ICMP ECHO REPLY

```
def receive_one_ping(self, current_socket):
    """
    Receive the ping from the socket. timeout = in ms
    """
    timeout = self.timeout / 1000.0

    while True:  # Loop while waiting for packet or timeout
        select_start = timer()
        inputready, outputready, exceptready = select.select([
                                                current_socket], [], [],
                                                timeout)
        select_duration = (timer() - select_start)
        if inputready == []:  # timeout
            return None, 0, 0, 0, 0

        receive_time = timer()
```

```
        packet_data, address = current_socket.recvfrom(ICMP_MAX_RECV)

        icmp_header = self.header2dict(
            names=[
                "type", "code", "checksum",
                "packet_id", "seq_number"
            ],
            struct_format="!BBHHH",
            data=packet_data[20:28]
        )

        if icmp_header["packet_id"] == self.own_id:  # Our packet
            ip_header = self.header2dict(
                names=[
                    "version", "type", "length",
                    "id", "flags", "ttl", "protocol",
                    "checksum", "src_ip", "dest_ip"
                ],
                struct_format="!BBHHHBBHII",
                data=packet_data[:20]
            )
            packet_size = len(packet_data) - 28
            ip = socket.inet_ntoa(struct.pack("!I", ip_header["src_ip"]
                                              ))

            return receive_time, packet_size, ip, ip_header,
                                              icmp_header

        timeout = timeout - select_duration
        if timeout <= 0:
            return None, 0, 0, 0, 0
```

Implementation of ICMP echo reply. We first create a loop for waiting for the packet and create a reply message if a request/packet is received.

3.6 Handling One Ping

```
def do_work(self):
    """
    Send one ICMP ECHO_REQUEST and receive the response until self.
                                        timeout
```

```
    """
    try:
        if self.udp:
            current_socket = socket.socket(socket.AF_INET, socket.
                                           SOCK_DGRAM, socket.
                                           getprotobyname("icmp"))
            if self.source_address:
                current_socket.bind((self.source_address, 1))
        else:
            current_socket = socket.socket(socket.AF_INET, socket.
                                           SOCK_RAW, socket.
                                           getprotobyname("icmp"))
    except socket.error as e:
        if e.errno == 1:
            # error trace
            etype, evalue, etb = sys.exc_info()
            evalue = etype(
                "%s - ICMP messages can only be send from processes
                                                running as ROOT/
                                                ADMINISTRATOR." %
                                                evalue
            )
            raise etype(evalue).with_traceback(etb)
        raise  # raise the original error

    send_time = self.send_one_ping(current_socket)
    if send_time == None:
        return
    self.send_count += 1

    receive_time, packet_size, ip, ip_header, icmp_header = self.
                                        receive_one_ping(current_socket
                                        )
    current_socket.close()

    if receive_time:
        self.receive_count += 1
        self.ttl = ip_header["ttl"]
        delay = (receive_time - send_time) * 1000.0
        self.total_time += delay
        if self.min_time > delay:
            self.min_time = delay
```

```
        if self.max_time < delay:
            self.max_time = delay

        self.print_success(delay, ip, packet_size, ip_header,
                                                icmp_header)
        return delay
    else:
        self.print_failed()
```

This is a helper function that encapsulates one pair of sending ICMP echo request and receiving echo reply for easier use of constantly pinging.

3.7 Constantly PING

```
def run(self, count=None, deadline=None):
    """
    send and receive pings in a loop. Stop if count or until deadline.
    """
    if not self.logging:
        self.setup_signal_handler()

    while True:
        delay = self.do_work()

        self.seq_number += 1
        if count and self.seq_number >= count:
            break
        if deadline and self.total_time >= deadline:
            break

        if delay == None:
            delay = 0

        if (MAX_SLEEP > delay):
            time.sleep((MAX_SLEEP - delay) / 1000.0)

    self.print_exit()
    if self.logging:
        return self.response
```

A constantly pinging helper function runs ping by a count specified by the argument – count. The user can change the count parameter to make the program to perform

different times of ping.

4 Running and Observation

```
(base) dhcp115:CS542 duanbin$ sudo python -u ping.py
------------ Ping Options ---------------
                  count: 2
               hostname: www.google.com
               log_file: log.txt
            packet_size: 256
                timeout: 1000
----------------- End --------------------
Ping www.google.com at (142.250.190.132): 256 data bytes
Receive 68 bytes from www.google.com (142.250.190.132): icmp_seq=0 ttl=118 time=22.06 ms
Receive 68 bytes from www.google.com (142.250.190.132): icmp_seq=1 ttl=118 time=22.46 ms
----www.google.com PING Statistics----
2 packets transmitted, 2 packets received, 0.00 % packet loss
round-trip min/avg/max = 22.06(ms)/22.26(ms)/22.46(ms)
```

Figure 1: Running with default parameters.

```
(base) dhcp115:CS542 duanbin$ sudo python -u ping.py --count 4 --packet_size 32
------------ Ping Options ---------------
                  count: 4                                  [default: 2]
               hostname: www.google.com
               log_file: log.txt
            packet_size: 32                                 [default: 256]
                timeout: 1000
----------------- End --------------------
Ping www.google.com at (142.250.190.132): 32 data bytes
Receive 32 bytes from www.google.com (142.250.190.132): icmp_seq=0 ttl=118 time=14.58 ms
Receive 32 bytes from www.google.com (142.250.190.132): icmp_seq=1 ttl=118 time=22.28 ms
Receive 32 bytes from www.google.com (142.250.190.132): icmp_seq=2 ttl=118 time=22.24 ms
Receive 32 bytes from www.google.com (142.250.190.132): icmp_seq=3 ttl=118 time=125.49 ms
----www.google.com PING Statistics----
4 packets transmitted, 4 packets received, 0.00 % packet loss
round-trip min/avg/max = 14.58(ms)/46.15(ms)/125.49(ms)
```

Figure 2: Running with changing parameters. Here, we change the constant ping count from 2 to 4, and packet size from 256 to 32.

Observation 1: The delay for a single ICMP message varies. We can easily notice in Figure. 3. The time is changing during the process of constant pinging. Since we are using TCP, the sequence numbers are ordered.

```
(base) dhcp115:CS542 duanbin$ sudo python -u ping.py --count 8 --packet_size 32
------------ Ping Options ---------------
                   count: 8                                  [default: 2]
                hostname: www.google.com
                log_file: log.txt
             packet_size: 32                                 [default: 256]
                 timeout: 1000
----------------- End --------------------
Ping www.google.com at (142.250.190.132): 32 data bytes
Receive 32 bytes from www.google.com (142.250.190.132): icmp_seq=0 ttl=118 time=15.04 ms
Receive 32 bytes from www.google.com (142.250.190.132): icmp_seq=1 ttl=118 time=60.86 ms
Receive 32 bytes from www.google.com (142.250.190.132): icmp_seq=2 ttl=118 time=105.87 ms
Receive 32 bytes from www.google.com (142.250.190.132): icmp_seq=3 ttl=118 time=16.31 ms
Receive 32 bytes from www.google.com (142.250.190.132): icmp_seq=4 ttl=118 time=16.65 ms
Receive 32 bytes from www.google.com (142.250.190.132): icmp_seq=5 ttl=118 time=16.47 ms
Receive 32 bytes from www.google.com (142.250.190.132): icmp_seq=6 ttl=118 time=15.75 ms
Receive 32 bytes from www.google.com (142.250.190.132): icmp_seq=7 ttl=118 time=20.49 ms
----www.google.com PING Statistics----
8 packets transmitted, 8 packets received, 0.00 % packet loss
round-trip min/avg/max = 15.04(ms)/33.43(ms)/105.87(ms)
```

Figure 3: Running with changing parameters. Here, we change the constant ping count from 2 to 8, and packet size from 256 to 32.

```
(base) dhcp115:CS542 duanbin$ sudo python -u ping.py --hostname localhost
------------ Ping Options ---------------
                   count: 2
                hostname: localhost                          [default: www.google.com]
                log_file: log.txt
             packet_size: 256
                 timeout: 1000
----------------- End --------------------
Ping localhost at (127.0.0.1): 256 data bytes
Receive 256 bytes from localhost (127.0.0.1): icmp_seq=0 ttl=64 time=0.29 ms
Receive 256 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.54 ms
----localhost PING Statistics----
2 packets transmitted, 2 packets received, 0.00 % packet loss
round-trip min/avg/max = 0.29(ms)/0.41(ms)/0.54(ms)
```

Figure 4: Ping localhost by changing the hostname argument.

5 Performance and Observation

We conduct a performance test of the ping program by changing the count arguments and plot the min/avg/max round trip time to study its performance.

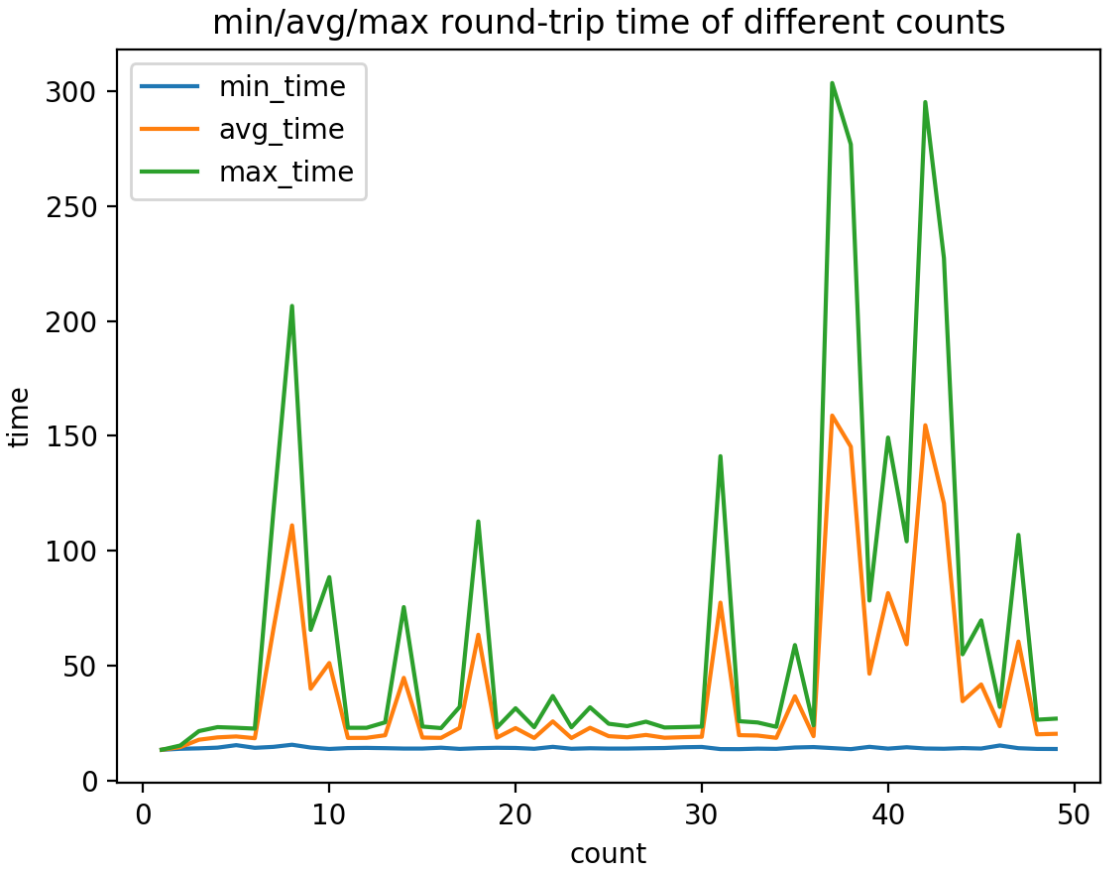

Figure 5: Min/Avg/Max round-trip time under different counts [ICMP echo request] for constant ping *www.google.com*. Unit: ms.

Observation 2: The min round-trip time of different counts is quite consistent. The tendencies of min and max round time are consistent to each other. The maximum round-trip time fluctuates maybe due to the network and connection.

6 References

[1] https://github.com/Akhavi/pyping

[2] Lecture 20 of CS542 course by Dr. Lan Yao