IIT CS542: Computer Network Fundamentals

Final Project

Team Member:

Muhao Chen, A20456889, mchen69@hawk.iit.edu

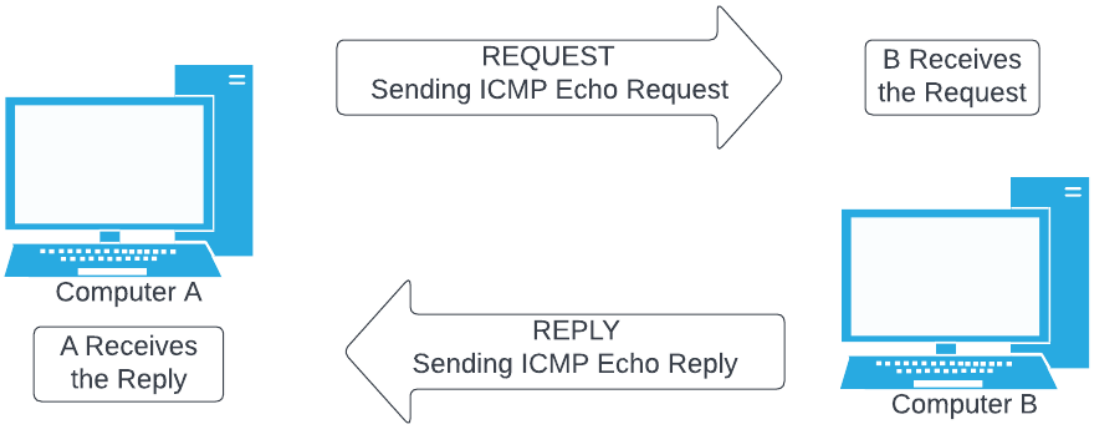
Xiaoxu Li, A20500966, xli280@hawk.iit.edu

Remark: All of work is done by us together. We use **Python** and **Mac**/**Windows** OS.

1. Overview

1.1 hosts information and interaction

In this project, we created an application program to implement a PING command by sending and receiving ICMP messages by using Row Socket in python. We accomplished the requirements of sender sends out 2 ICMP requests constantly with the length of 256 bytes (data portion/payload), the receiver sends back replies and print the requests it receives from the sender to a .txt file and when the sender receives replies, it prints the reply messages on the screen and in a .txt file as well.



The information of HostA and HostB, and their interactions.

Name	OS	Role	Ip Address	Action/Function
HostA	Mac	Sender	192.168.50.133	Request(step1)
				Receive Reply(step4)
HostB	Windows	Receiver	192.168.50.202	Receive Request(step2)
				Reply(step3)

Limited in the IP address, we let two hosts in the same LAN, due to the private IP address.

1.2 ICMP introduction

- ICMP

We can use PING command through ICMP protocol to explore the target IP address and calculate the round trip time. The ICMP include type, code, checksum, indentifier, sqence number and data. The ICMP structure is shown below.

8 bit	8 bit	8 bit	8 bit
TYPE	CODE	CHECKSUM	
IDENTIFIER		SEQUENCE NUMBER	
DATA			

- The Type and Code of Echo Request/Reply are shown below.

	Type	Code
Request	8	0
Reply	0	0

- Checksum

The Checksum is used to check if the ICMP header is validate. The checksum is a 16 bit long which contains a one's complement of the ones complement sum of the ICMP message's header, starting with the ICMP type. The checksum should be set to 0, when calculating the checksum.

- Identifier

Identifier is used to identify the request packet. In order to keep the different requests and replies, the identifier will be echoed back in the reply.

- Sequence Number

Sequence number usually starts at 1, and incremented by 1 after each transmission for each packet. It is a number to show the identifier of the sending message.

In the project, I give an example of request struct.pack to show ICMP structure.

And I will detailly introduce in the following part.

```
# pack icmp using struct, 8 is request, code is 0
type_request = 8
type_code = 0
# pack icmp using struct
icmp_before_checksum = struct.pack('!BBHHH',type_request,type_code,0,identify_num,sequence_number)+payload_add_time
```

1.3 Result Overview

Step 1: Set Destination IP address, and you can rewrite other parameters.

```
# define the parameters
# 1. the bit length of payload data
payload_data_bit_length = 256
# 2. define the type of message
message_type = "ICMP"      .
# 3. reveiver ip address
rec_ip_address = "192.168.50.202"
```

Step 2: Receiver should listen all the time, to wait for ICMP message. Therefore, type "python reve_icmp.py" in the terminal(Windows, receiver host).

The graph shows the result while receiving ICMP messages. (actually, after step 3)

```
S:\program test\test_22_5_06_1\ping_fin>python reve_icmp.py
start to keep listening:

****************************************receive icmp****************************************
sequence_number:  1
id_address_reveiver:  192.168.50.133
content/payload:
57896044618658097711785492504343953926634992332820282019728792003956564819968
********************************************************************************************


****************************************receive icmp****************************************
sequence_number:  2
id_address_reveiver:  192.168.50.133
content/payload:
57896044618658097711785492504343953926634992332820282019728792003956564819968
********************************************************************************************

start to reply:
====================================sending=============================================
start sending icmp,round:  1
================================================================================
start sending, sequence number:  1
finish sending, sequence number:  1
start sending, sequence number:  2
finish sending, sequence number:  2
================================================================================
```

Step 3: Sender should send 2 requests ICMP messages. Therefore, type "python send_icmp.py" in the terminal(Mac, sender host).

The graph shows the sending messages, receiver replies and the statistics.

```
(base) elonchen@MacBook-Pro share_mac % sudo python3 send_icmp.py
====================================sending=============================================
start sending icmp,round:  1
================================================================================
start sending, sequence number:  1
finish sending, sequence number:  1
start sending, sequence number:  2
finish sending, sequence number:  2
================================================================================

****************************************receive reply*****************************************
sequence_number:  1
id_address_reveiver:  192.168.50.202
time_period:  6.2551  ms
content/payload:
57896044618658097711785492504343953926634992332820282019728792003956564819968
**********************************************************************************************


****************************************receive reply*****************************************
sequence_number:  2
id_address_reveiver:  192.168.50.202
time_period:  73.2131  ms
content/payload:
57896044618658097711785492504343953926634992332820282019728792003956564819968
**********************************************************************************************

++++++++++++++++++++++++++++++++++++statistics++++++++++++++++++++++++++++++++++++
total sending number:  2
total reply number:  2
loss rate:  0.0
max pass time:  73.2131
min pass time:  6.2551
average pass time:  39.734
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

2. Sender's Program Detail

1) Set Parameters

There are a lot of parameters, including IP address, message style, length of the payload data and statistics initial parameters. The only thing you need to change is the IP address.

```
# define the parameters
# 1. the bit length of payload data
payload_data_bit_length = 256
# 2. define the type of message
message_type = "ICMP"
# 3. reveiver ip address
rec_ip_address = "192.168.50.202"
# 4. how many times you wanna to send, sum_times = icmp_nums*2, because two icmp one time
icmp_nums = 1
# 5. sending number
sending_num = 0
# 6. reveving number
receving_num = 0
# 7. time_list
time_list = []
```

2) Create Socket

This function is used to implement creating the raw socket. The message type in this function is ICMP which is defined at the beginning of the program.

```
# define and create the socket
def create_socket(message_type):
    obj_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.getprotobyname(message_type))
    return obj_socket
```

3) Generate Payload Data

This function is used to generate the payload data. Based on the requirement of this project, the payload data is 256 bytes long. In the code, the payload data is an array of integers which is generated gradually until reach the 256 bytes. And then, we convert the integers to bytes by using the mode (%) operation to make sure the data is in even bytes.

```
# generate the data through func, based on bit length required, such as 256.
def generate_payload_data(bit_length):
    payload_data = 1
    while(True):
        if payload_data.bit_length() == 256:
            break
        payload_data = payload_data * 2
    # change the int to bytes
    payload_data = b'%d'%(payload_data)
    return payload_data
```

4) Sending ICMP

First of all, we need to pass the following parameters into the function. ICMP Type = 8, ICMP, Code = 0, Identifier Number = 3400 and Sequence Number = 1. And then, we calculate the sending time by record the current time. Next, we encapsulate payload includes the sending time and data by using struct.pack.

```
def send_icmp(obj_socket, rec_ip_address, identify_num, payload_data):
    # set sequence number
    sequence_number = 1
    icmp_nums_sum = icmp_nums * 2
    while sequence_number < icmp_nums_sum+1:
        print("=====================================sending=============================================")
        print("start sending icmp,round: ",str(sequence_number//2+1))
        print("=========================================================================================")
        for i in range(2):
            print("start sending, sequence number: ",sequence_number)
            # record the current time
            time_start = time.time()
            # add the time to the payload
            payload_add_time = struct.pack('!d', time_start) + payload_data
            # pack icmp using struct, 8 is request, code is 0
            type_request = 8
            type_code = 0
            # pack icmp using struct
            icmp_before_checksum = struct.pack('!BBHHH',type_request,type_code,0,identify_num,sequence_number)+payload_add_time
            # compute checksum based on raw icmp
            checksum = get_checksum(icmp_before_checksum)
            # add checksum to the final sending icmp
            icmp = icmp_before_checksum[:2] + checksum + icmp_before_checksum[4:]
            # send the icmp to the deitination
            obj_socket.sendto(icmp, 0, (rec_ip_address, 0))
            print("finish sending, sequence number: ",sequence_number)
            # sequence ++
            sequence_number = sequence_number + 1
            # send num plus 1
            global sending_num
            sending_num = sending_num + 1
        print("=========================================================================================")
        time.sleep(3)
    return None
```

5) Get Checksum

This function is do the calculation to get the result of the checksum value. Firstly, we obtain the length of ICMP before the checksum. If the length is odd, we need add a space at the end of the ICMP. Then we compute how many 2 bytes words in the ICMP. Next, we capture the words by two bytes, then transfer it to checksum and convert the final checksum value.

```
def get_checksum(icmp_before_checksum):
    # set checksum value
    checksum = 0
    # get the legth of icmp before checksum
    icmp_before_checksum_length = len(icmp_before_checksum)
    # if check sum could not be divided by 2, add \00
    if (icmp_before_checksum_length % 2):
        icmp_before_checksum = icmp_before_checksum + b'\00'
    # compute how many two btyes word
    len_two_B_icmp = len(icmp_before_checksum)//2
    # get every two bytes word
    for i in range(len_two_B_icmp):
        byte_content = struct.unpack('!H', icmp_before_checksum[2*i:2*i+2])[0]
        checksum += byte_content
    # transfer into checksum
    more_than_16bit = checksum >> 16
    while more_than_16bit:
        more_than_16bit = checksum >> 16
        checksum = (checksum & 0xffff) + more_than_16bit
    # reverse the code
    checksum = ~checksum & 0xffff
    checksum = struct.pack('!H', checksum)
    return checksum
```

6) Receive the reply

In this process, the sender will keep listening the reply from the receiver. Once the receiver sends out the reply, the sender will unpack the struct to get the data in binary, then convert to decimal. And then the sender need verify if the type is 0, the code is 0, any return , and check if the identifier from the reply is same as the sender's sending identifier. Recording the sending time from receiver to calculate the period of its sending. At last, read and unpack the payload.

```
def receive_icmp(obj_socket, identify_num):
    while True:
        try:
            # listen the port, wait the reply icmp
            icmp_reply, reply_address_tuple = obj_socket.recvfrom(1000)
            print(" ")
            print("*****************************************receive reply*****************************************")
            reply_address = reply_address_tuple[0]
            # unpack the struct to get the data
            icmp_reply_process = icmp_reply[20:]
            reply_type, reply_code, v1, current_ident, sequence_number, = struct.unpack('!BBHHH',icmp_reply_process[:8])
            # judge reply type is 0, whether code is 0
            if (reply_type != 0) or (reply_code != 0):
                pass
            payload = icmp_reply_process[8:]
            #judge whether there are any return , judge whether there are the same identify number, true rply, not other icmp
            if (current_ident != identify_num) and current_ident:
                pass
            # get the sending time from the reply, compute the time period
            sending_time, = struct.unpack('!d', payload[:8])
            period_time = (time.time()-sending_time) * 1000
            period_time = round(period_time,4)
            print("sequence_number: ",sequence_number)
            print("id_address_reveiver: ",reply_address)
            print("time_period: ",period_time," ms")
            # get the content
            print("content/payload: ")
            print_payload = str(payload[8:])[2:-1]
            print(print_payload)
            print("***********************************************************************************************************")
            print(" ")
            # write into the txt
            with open('sender_receve_replies_data_icmp'+str(sequence_number)+'.txt','w') as f:
                f.write(print_payload)
            # rece num plus 1
            global receving_num
            receving_num = receving_num + 1
            time_list.append(period_time)
            if sequence_number == 2:
                # show the statistics
                gene_stats()
                break
        except:
            continue
```

7) Statistic Message

The statistic message includes the number of the sending messages, the number of the reply messages, the loss rate, the maximum transmission time, the minimum transmission time and the average transmission time.

```
# generate statistics
def gene_stats():
    print("++++++++++++++++++++++++++++++++++++statistics++++++++++++++++++++++++++++++++++++")
    print("total sending number: ",sending_num)
    print("total reply number: ",receving_num)
    print("loss rate: ",round((sending_num-receving_num)/receving_num,2))
    print("max pass time: ",max(time_list))
    print("min pass time: ",min(time_list))
    print("average pass time: ",round(sum(time_list)/len(time_list),3))
    print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++")
```

3. Receiver's Program Detail (meanly show differences)

1) Define the Socket

Because Windows is a little different from Mac, the socket raw API requires socket Bind a static IP address, and set the initial parameters for the socket creation.

```
# define and create the socket
def create_socket(message_type):
    obj_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.getprotobyname(message_type))
    if sys.platform == 'win32':
        # windows require the binding, i do not know why
        # https://docs.microsoft.com/en-us/windows/win32/winsock/tcp-ip-raw-sockets-2
        obj_socket.bind(("192.168.50.202", 0))
        obj_socket.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)
    return obj_socket
```

2) Receive ICMP Messages

Keep listening through socket.recvfrom, and while getting the useful messages, which means have the same identify number. Then Print the sequence number, ip address and payload data, meanwhile, saving the data into the txt file.

```
def receive_icmp(obj_socket, identify_num):
    while True:
        try:
            # listen the port, wait the reply icmp
            icmp_reply, reply_address_tuple = obj_socket.recvfrom(1000)
            print(" ")
            print("**********************************************receive icmp**********************************************")
            reply_address = reply_address_tuple[0]
            reply_address = "192.168.50.133"
            # unpack the struct to get the data
            icmp_reply_process = icmp_reply[20:]
            reply_type, reply_code, v1, current_ident, sequence_number, = struct.unpack('!BBHHH',icmp_reply_process[:8])
            payload = icmp_reply_process[8:]
            #judge whether there are any return , judge whether there are the same identify number, true rply, not other icmp
            if (current_ident != identify_num) and current_ident:
                pass
            # get the sending time from the reply, compute the time period
            sending_time, = struct.unpack('!d', payload[:8])
            print("sequence_number: ",sequence_number)
            print("id_address_reveiver: ",reply_address)
            # get the content
            print("content/payload: ")
            print_payload = str(payload[8:])[2:-1]
            print(print_payload)
            print("*************************************************************************************************************")
            print(" ")
            # write into the txt
            with open('reveiver_receive_data_icmp'+str(sequence_number)+'.txt','w') as f:
                f.write(print_payload)
            if sequence_number == 2:
                return reply_address,print_payload,sending_time
        except:
            continue
```

3) Send/Reply ICMP Messages

The receiver will reply the ICMP message with the same data payload, but change the type from 8 to 0, because it is a reply for the request. And we should keep the sending time recording is not changed.

```
def send_icmp(obj_socket, rec_ip_address, identify_num, payload_data,sending_time):
    # set sequence number
    sequence_number = 1
    icmp_nums_sum = icmp_nums * 2
    while sequence_number < icmp_nums_sum+1:
        print("=====================================sending=============================================")
        print("start sending icmp,round: ",str(sequence_number//2+1))
        print("=================================================================================================")
        for i in range(2):
            print("start sending, sequence number: ",sequence_number)
            # add the time to the payload
            payload_add_time = struct.pack('!d', sending_time) + payload_data
            # pack icmp using struct, 8 is request, code is 0
            type_request = 0
            type_code = 0
            # pack icmp using struct
            icmp_before_checksum = struct.pack('!BBHHH',type_request,type_code,0,identify_num,sequence_number,)+payload_add_time
            # compute checksum based on raw icmp
            checksum = get_checksum(icmp_before_checksum)
            # add checksum to the final sending icmp
            icmp = icmp_before_checksum[:2] + checksum + icmp_before_checksum[4:]
            # send the icmp to the deitination
            obj_socket.sendto(icmp, 0, (rec_ip_address, 0))
            print("finish sending, sequence number: ",sequence_number)
            # sequence ++
            sequence_number = sequence_number + 1
        print("=================================================================================================")
        time.sleep(3)
    return None
```

4. Robust:

1) With two threads, it is guaranteed to send and listen at the same time, and it is guaranteed not to miss an ICMP reply.
2) Use If, to figure out whether there are other useless or irrelative ICMP messages. If the identify is not the same, drop it.
3) Use While loop to send the message, and listen to wait for the message, use the try function to avoid the false situations.
4) Use Statistics to show the total situations of the ICMP requests and replies.

5. Performance and Consideration:

1) The performance, in the page 3 is excellent, due to the 0-loss rate. And the latency is low, which means the network and code is excellent.
2) Surprising and Consideration, (1) due to the LAN and private work, we could not send the ICMP messages to public. (2) While network is bad ,and should we set timeout function.