REPORT

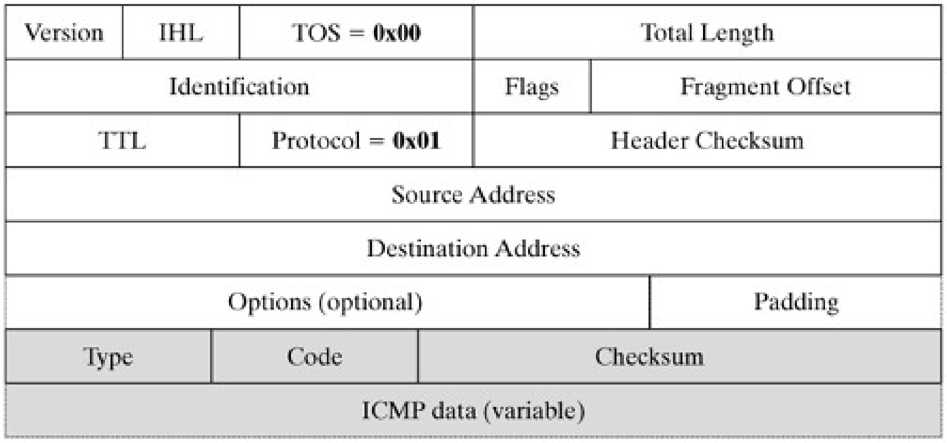
Ping Command- ICMP echo request/reply

Introduction

ICMP stands for Internet Control Message Protocol. It is used to enhance the reliability which provides information about errors, loss of packets, unavailable destinations, etc. It is documented as RFC 792. It is mandatory that every device that implements IP (Internet Protocol), must also implement ICMP.

We have been using IP addresses for communication and IP is an unreliable datagram service. IP packets are incapable of sending error messages if anything goes wrong in the network, like sending a letter to a destination and the letter not getting received at the desired destination and you will never come to know about it. So, to handle this kind of issue an IP has an assistant called ICMP (Internet Control Message Protocol).

In ICMP any destination or router that detects any problem in handling a received IP packet, generated ICMP message addressed to the originating station of IP packet. ICMP message can be analysed by network management systems to generate network reports for the network administrators.



<table>
<tr><td>Version</td><td>IHL</td><td colspan="2">TOS = <b>0x00</b></td><td colspan="4">Total Length</td></tr>
<tr><td colspan="4">Identification</td><td>Flags</td><td colspan="3">Fragment Offset</td></tr>
<tr><td colspan="2">TTL</td><td colspan="2">Protocol = <b>0x01</b></td><td colspan="4">Header Checksum</td></tr>
<tr><td colspan="8">Source Address</td></tr>
<tr><td colspan="8">Destination Address</td></tr>
<tr><td colspan="6">Options (optional)</td><td colspan="2">Padding</td></tr>
<tr><td colspan="2">Type</td><td colspan="2">Code</td><td colspan="4">Checksum</td></tr>
<tr><td colspan="8">ICMP data (variable)</td></tr>
</table>

IP packet containing ICMP

ICMP messages are send as IP packets. The protocol field of IP header is set to 0x01 to indicate that this packet contains ICMP message.

In this report, the use of ICMP for Request and Reply messages is illustrated.

Concept of Request and Reply

In Request and Reply, Sender should send a request packet and get an ICMP reply packet either from the receiver or router along the way.
Request and Reply messages are used in different ways.

- Echo request and reply
- Router Solicitation and advertisement
- TimeStamp request and reply
- Network mask request and reply

Here, Echo request and reply are highlighted.
ICMP packets of echo request and reply are used to test the network layer of destination. In other words, the host at the destination is switched on or off. We can also check the network layer of all the devices on the way from sender to destination.

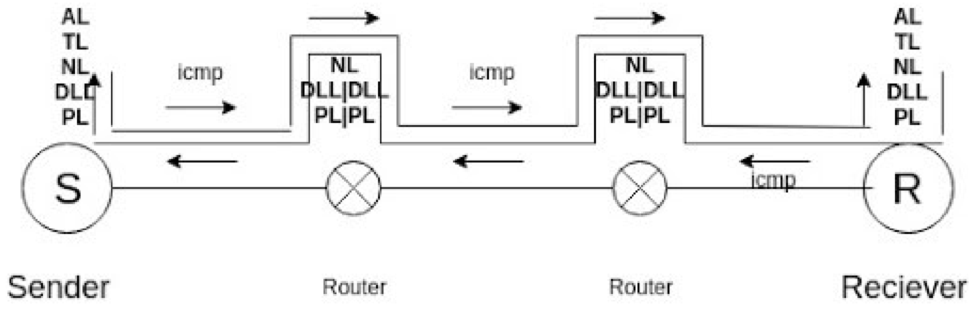


Figure: Sender & Receiver sending and getting ICMP packets via network layer

In case, any network layer fails, the message will be discarded. ICMP packets are used at the network layer whenever we send an echo request. At the application layer, this echo request will never appear. So, no one will find out that there is actually a packet sent. Because of these, various attacks are possible. Generally, in our computer systems, the program called Ping (Packet Internet Groper) is implemented using echo request and reply.

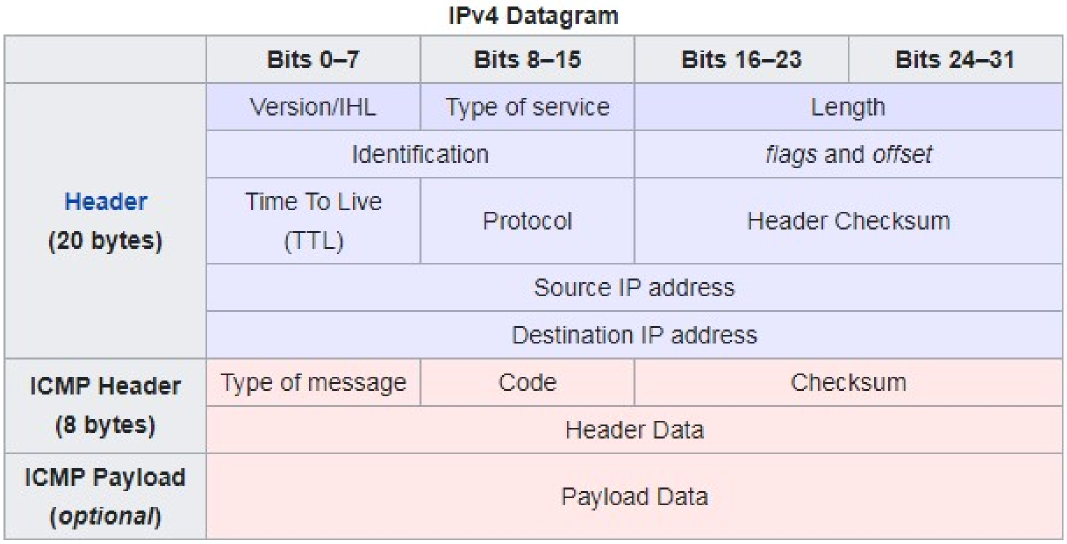
Observation

The different ICMP message types are:

ICMP Type	ICMP Code	Description
0	0	Echo Reply (used by ping)
3	0	Destination Network Unreachable
3	1	Destination Host Unreachable
3	3	Destination Port Unreachable
8	0	Echo request (used by ping)
11	0	TTL Expired (used by traceroute)

Basic ICMP message consists of three fields — type, code and checksum (used for error detection). Extension field is used with some of the messages.
A) type field: Specifies the message type (Ex: Destination unreachable)
B) code field: Describes the type (Ex: reason why the destination was unreachable)

When an IP packet containing ICMP echo request. (Type 8) is sent to a host, the host returns IP packet with ICMP echo reply (TYPE 0).



IPv4 Datagram

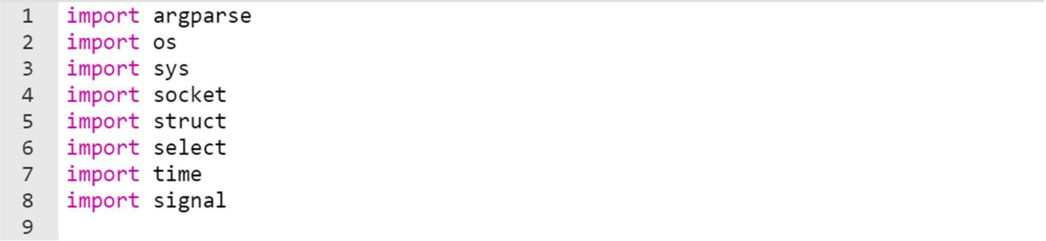
<table>
<tr><th></th><th>Bits 0–7</th><th>Bits 8–15</th><th>Bits 16–23</th><th>Bits 24–31</th></tr>
<tr><td rowspan="5"><b>Header</b>
<b>(20 bytes)</b></td><td>Version/IHL</td><td>Type of service</td><td colspan="2">Length</td></tr>
<tr><td colspan="2">Identification</td><td colspan="2"><i>flags</i> and <i>offset</i></td></tr>
<tr><td>Time To Live
(TTL)</td><td>Protocol</td><td colspan="2">Header Checksum</td></tr>
<tr><td colspan="4">Source IP address</td></tr>
<tr><td colspan="4">Destination IP address</td></tr>
<tr><td rowspan="2"><b>ICMP Header</b>
<b>(8 bytes)</b></td><td>Type of message</td><td>Code</td><td colspan="2">Checksum</td></tr>
<tr><td colspan="4">Header Data</td></tr>
<tr><td><b>ICMP Payload</b>
<b>(<i>optional</i>)</b></td><td colspan="4">Payload Data</td></tr>
</table>

The payload of the packet is generally filled with ASCII characters, as the output of the tcpdump command shows in the last 32 bytes of the following example (after the eight-byte ICMP header starting with 0x0800).

Ping (packet internet groper): It is an application of ICMP echo. It is used for estimation of round-trip delay, packet loss, and other parameters, Delay is measured by starting a timer at the time of sending the echo request and nothing the time when echo is received. Several PINGs are sent one after the other and round-trip time is expressed as minimum, maximum and average values.
Packet loss is estimated based on number of echo replies not on received. If out of 1000 echo requests, only 900 are replied to, the packet loss is 1%.

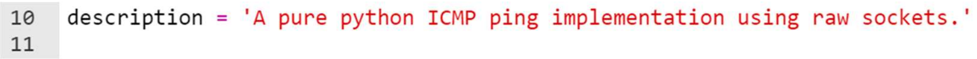
Code & Explanation

Importing the required libraries:



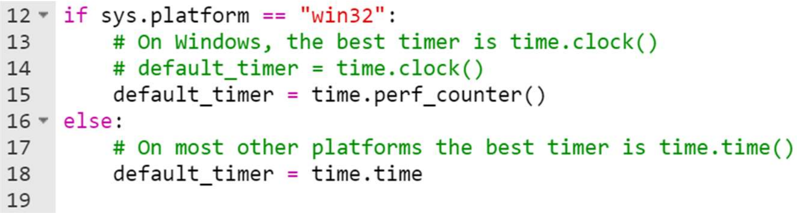
```python
import argparse
import os
import sys
import socket
import struct
import select
import time
import signal

```

This Python program is for the implementation of ICMP echo request and reply using raw sockets.



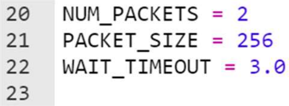
```python
description = 'A pure python ICMP ping implementation using raw sockets.'

```

Defining the timer based on the architecture of the system:



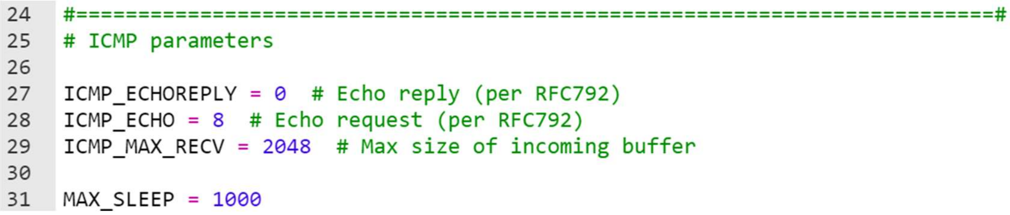
```python
if sys.platform == "win32":
    # On Windows, the best timer is time.clock()
    # default_timer = time.clock()
    default_timer = time.perf_counter()
else:
    # On most other platforms the best timer is time.time()
    default_timer = time.time

```

Here, the sender is supposed to send out 2 ICMP requests constantly with the length of 256 bytes; given as:



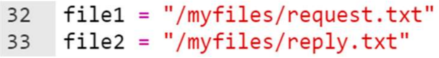
```python
NUM_PACKETS = 2
PACKET_SIZE = 256
WAIT_TIMEOUT = 3.0

```

Defining the ICMP parameters:



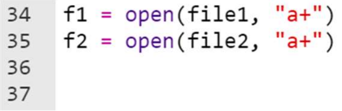
```python
#============================================================================#
# ICMP parameters

ICMP_ECHOREPLY = 0  # Echo reply (per RFC792)
ICMP_ECHO = 8  # Echo request (per RFC792)
ICMP_MAX_RECV = 2048  # Max size of incoming buffer

MAX_SLEEP = 1000
```

Defining the files to write the requests and replies to:



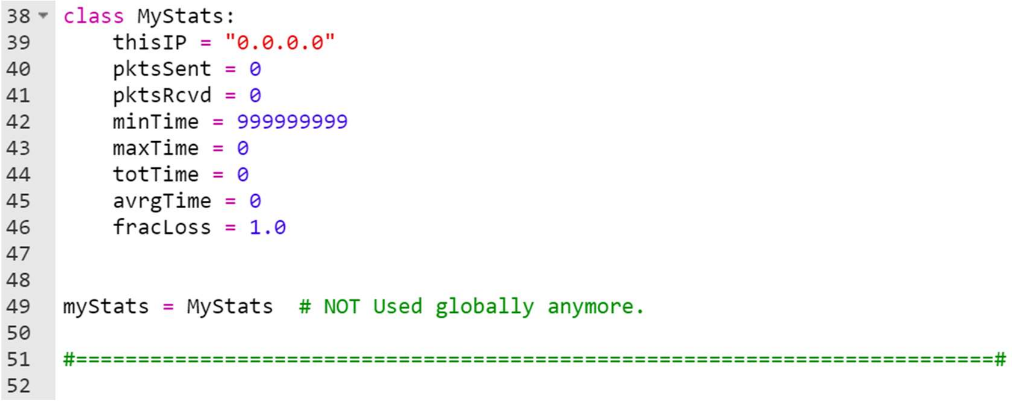
```python
file1 = "/myfiles/request.txt"
file2 = "/myfiles/reply.txt"
```

Opening the files in append mode for appending the messages on the respective text files:



```
f1 = open(file1, "a+")
f2 = open(file2, "a+")
```

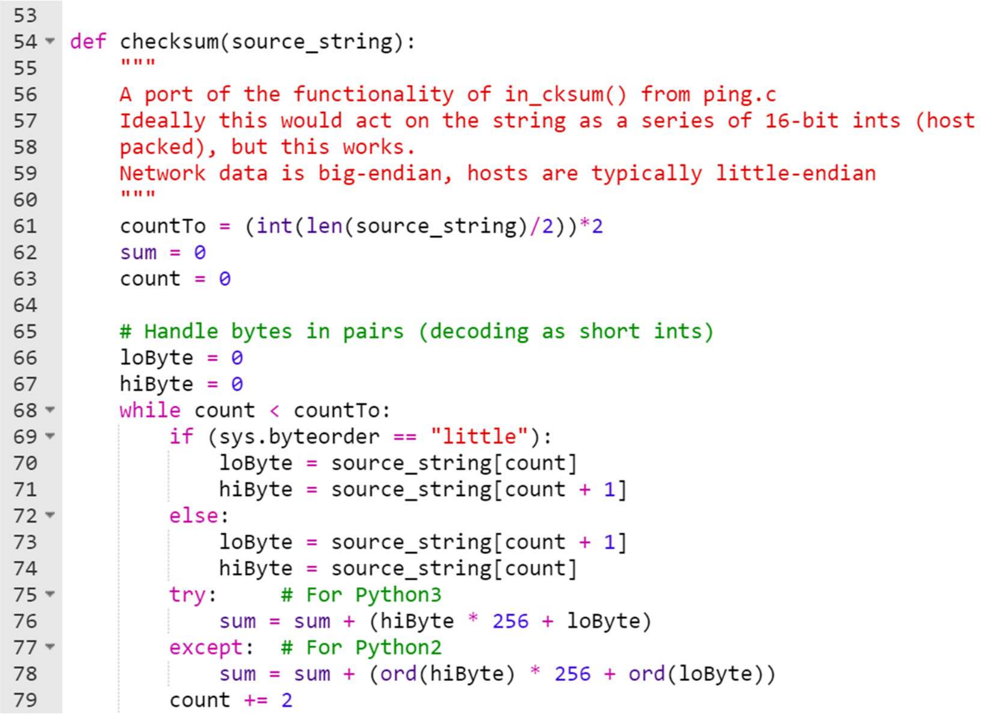
Defining the MyStats class for the statistics of the message exchange to be proceeded further in the program:



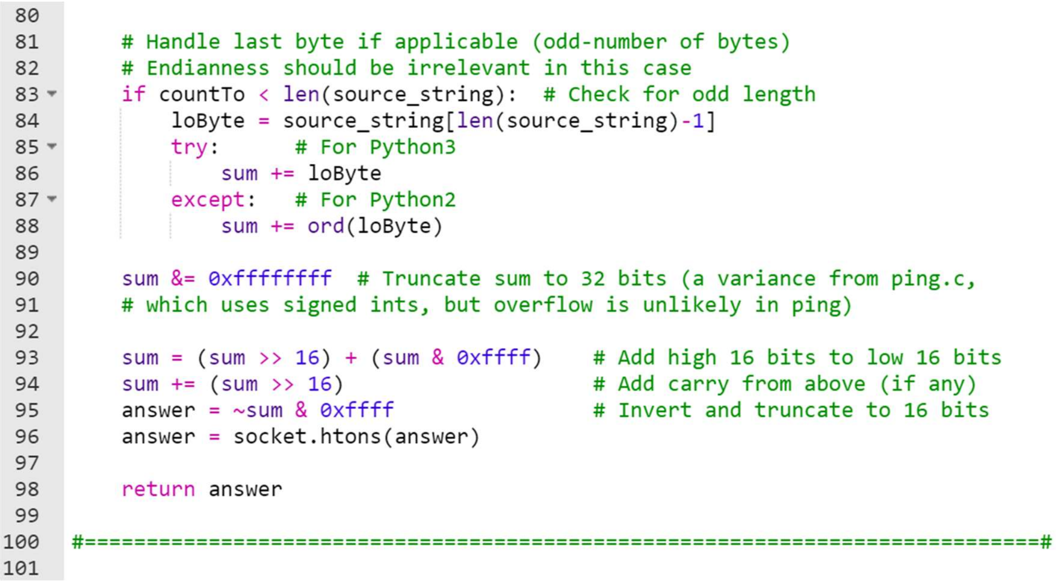
```
class MyStats:
    thisIP = "0.0.0.0"
    pktsSent = 0
    pktsRcvd = 0
    minTime = 999999999
    maxTime = 0
    totTime = 0
    avrgTime = 0
    fracLoss = 1.0


myStats = MyStats  # NOT Used globally anymore.

#==========================================================================#
```

The checksum verifies the validity of the ICMP header. When the data packet is transmitted, the checksum is computed and inserted into this field. When the data packet is received, the checksum is again computed and verified against the checksum field. If the two checksums do not match then an error has occurred.

Defining the checksum function:

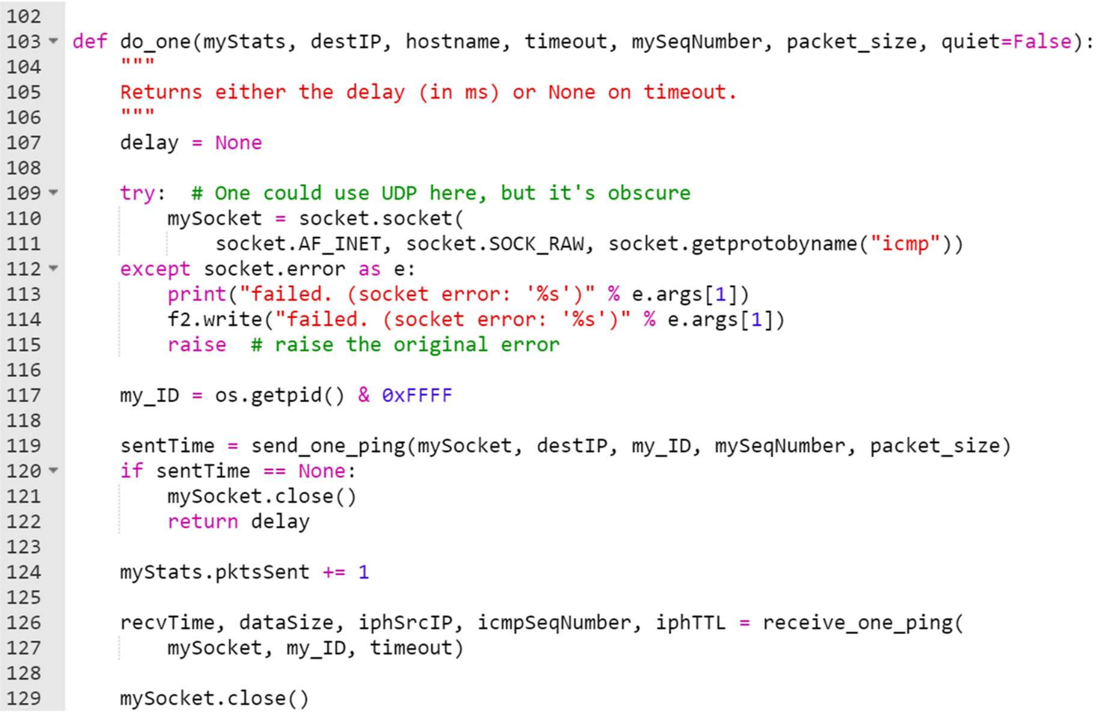


```

def checksum(source_string):
    """
    A port of the functionality of in_cksum() from ping.c
    Ideally this would act on the string as a series of 16-bit ints (host
    packed), but this works.
    Network data is big-endian, hosts are typically little-endian
    """
    countTo = (int(len(source_string)/2))*2
    sum = 0
    count = 0

    # Handle bytes in pairs (decoding as short ints)
    loByte = 0
    hiByte = 0
    while count < countTo:
        if (sys.byteorder == "little"):
            loByte = source_string[count]
            hiByte = source_string[count + 1]
        else:
            loByte = source_string[count + 1]
            hiByte = source_string[count]
        try:     # For Python3
            sum = sum + (hiByte * 256 + loByte)
        except:  # For Python2
            sum = sum + (ord(hiByte) * 256 + ord(loByte))
        count += 2
```

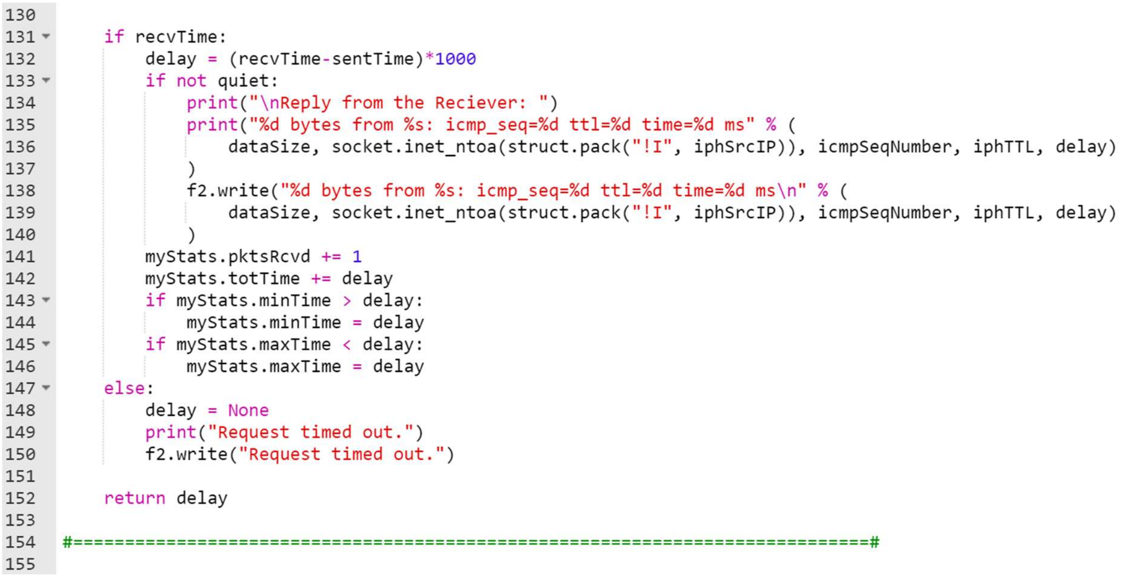


```

    # Handle last byte if applicable (odd-number of bytes)
    # Endianness should be irrelevant in this case
    if countTo < len(source_string):  # Check for odd length
        loByte = source_string[len(source_string)-1]
        try:      # For Python3
            sum += loByte
        except:   # For Python2
            sum += ord(loByte)

    sum &= 0xffffffff  # Truncate sum to 32 bits (a variance from ping.c,
    # which uses signed ints, but overflow is unlikely in ping)

    sum = (sum >> 16) + (sum & 0xffff)    # Add high 16 bits to low 16 bits
    sum += (sum >> 16)                    # Add carry from above (if any)
    answer = ~sum & 0xffff                # Invert and truncate to 16 bits
    answer = socket.htons(answer)

    return answer

#==============================================================================#

```

Checking for delay (in ms) or None on timeout:

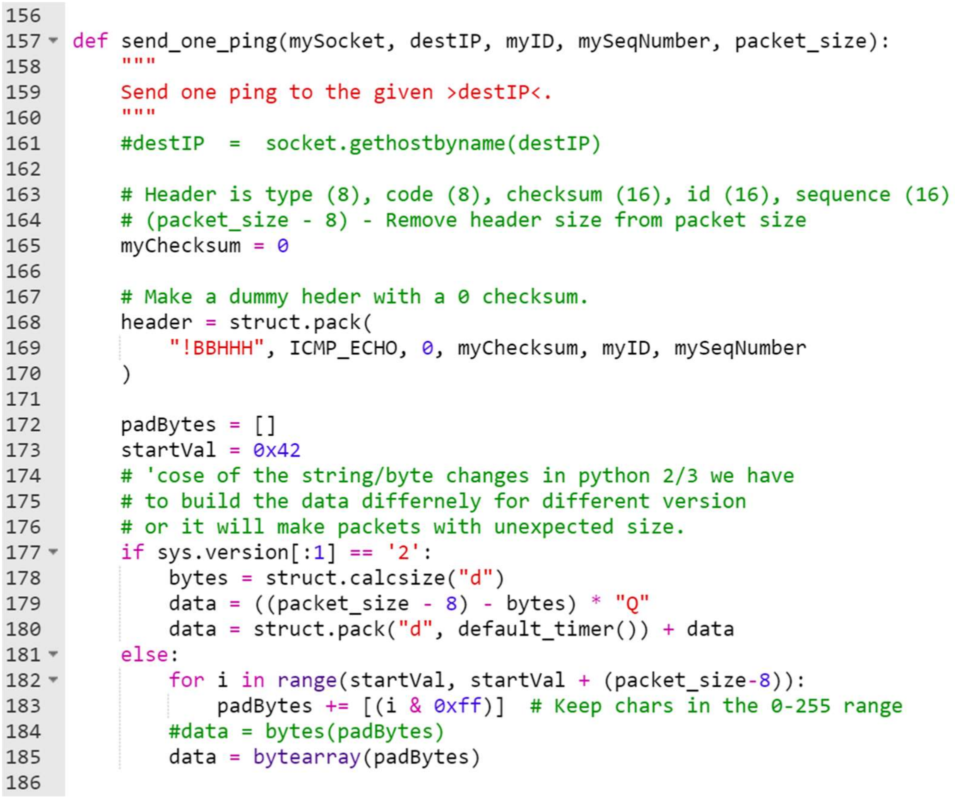


```

def do_one(myStats, destIP, hostname, timeout, mySeqNumber, packet_size, quiet=False):
    """
    Returns either the delay (in ms) or None on timeout.
    """
    delay = None

    try:  # One could use UDP here, but it's obscure
        mySocket = socket.socket(
            socket.AF_INET, socket.SOCK_RAW, socket.getprotobyname("icmp"))
    except socket.error as e:
        print("failed. (socket error: '%s')" % e.args[1])
        f2.write("failed. (socket error: '%s')" % e.args[1])
        raise  # raise the original error

    my_ID = os.getpid() & 0xFFFF

    sentTime = send_one_ping(mySocket, destIP, my_ID, mySeqNumber, packet_size)
    if sentTime == None:
        mySocket.close()
        return delay

    myStats.pktsSent += 1

    recvTime, dataSize, iphSrcIP, icmpSeqNumber, iphTTL = receive_one_ping(
        mySocket, my_ID, timeout)

    mySocket.close()
```

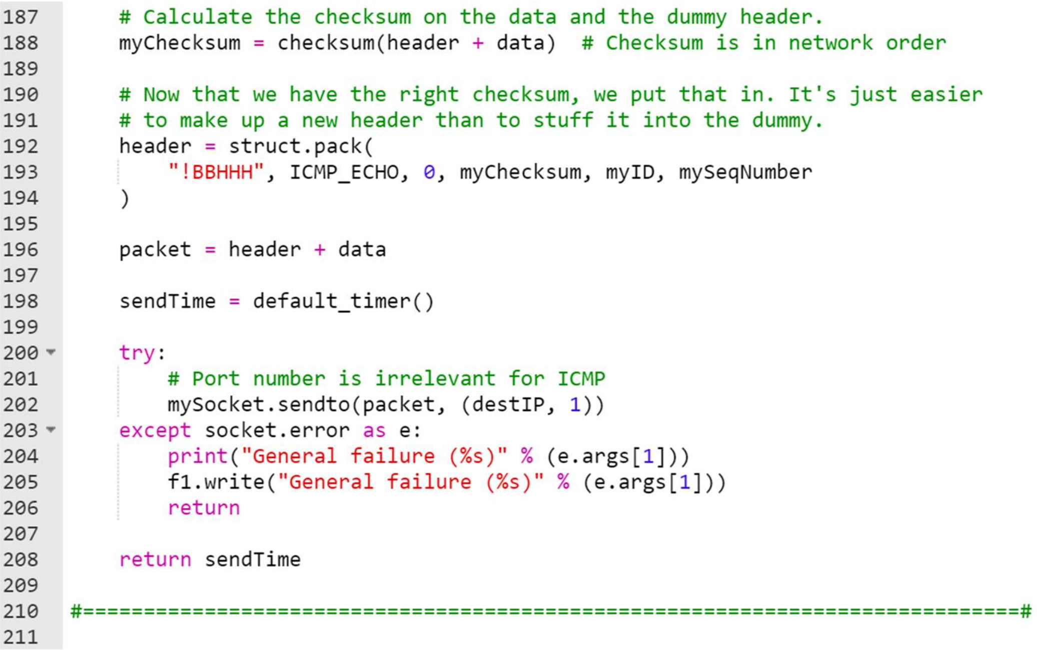


```python

    if recvTime:
        delay = (recvTime-sentTime)*1000
        if not quiet:
            print("\nReply from the Reciever: ")
            print("%d bytes from %s: icmp_seq=%d ttl=%d time=%d ms" % (
                dataSize, socket.inet_ntoa(struct.pack("!I", iphSrcIP)), icmpSeqNumber, iphTTL, delay)
            )
            f2.write("%d bytes from %s: icmp_seq=%d ttl=%d time=%d ms\n" % (
                dataSize, socket.inet_ntoa(struct.pack("!I", iphSrcIP)), icmpSeqNumber, iphTTL, delay)
            )
        myStats.pktsRcvd += 1
        myStats.totTime += delay
        if myStats.minTime > delay:
            myStats.minTime = delay
        if myStats.maxTime < delay:
            myStats.maxTime = delay
    else:
        delay = None
        print("Request timed out.")
        f2.write("Request timed out.")

    return delay

#==============================================================================#

```

Now, lets ping the destination IP address:

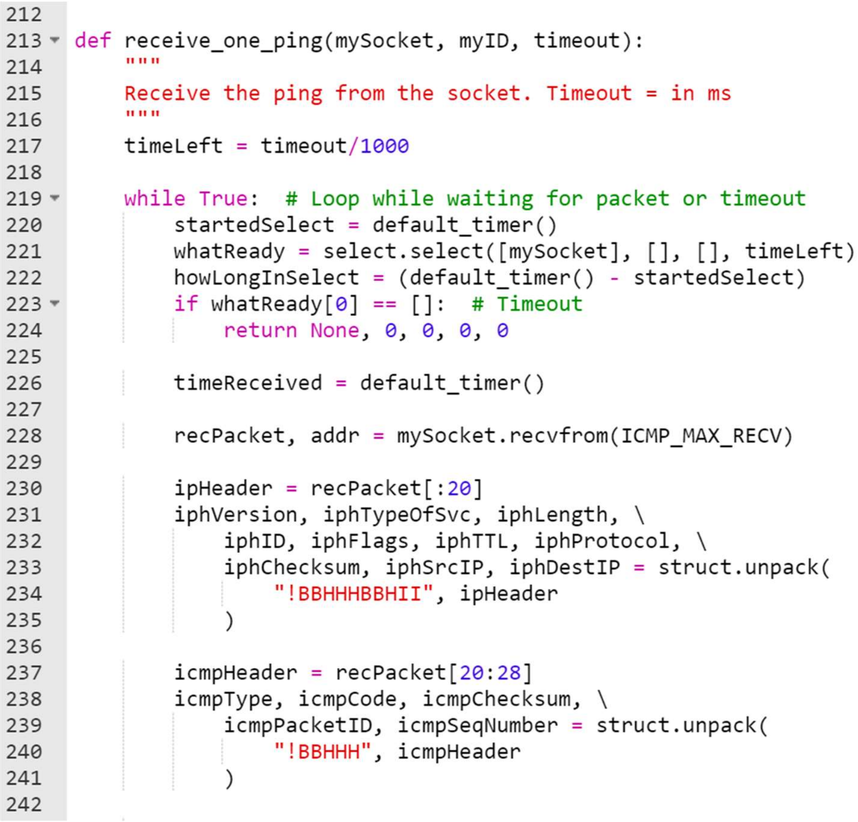


```python

def send_one_ping(mySocket, destIP, myID, mySeqNumber, packet_size):
    """
    Send one ping to the given >destIP<.
    """
    #destIP  =  socket.gethostbyname(destIP)

    # Header is type (8), code (8), checksum (16), id (16), sequence (16)
    # (packet_size - 8) - Remove header size from packet size
    myChecksum = 0

    # Make a dummy heder with a 0 checksum.
    header = struct.pack(
        "!BBHHH", ICMP_ECHO, 0, myChecksum, myID, mySeqNumber
    )

    padBytes = []
    startVal = 0x42
    # 'cose of the string/byte changes in python 2/3 we have
    # to build the data differnely for different version
    # or it will make packets with unexpected size.
    if sys.version[:1] == '2':
        bytes = struct.calcsize("d")
        data = ((packet_size - 8) - bytes) * "Q"
        data = struct.pack("d", default_timer()) + data
    else:
        for i in range(startVal, startVal + (packet_size-8)):
            padBytes += [(i & 0xff)]  # Keep chars in the 0-255 range
        #data = bytes(padBytes)
        data = bytearray(padBytes)

```

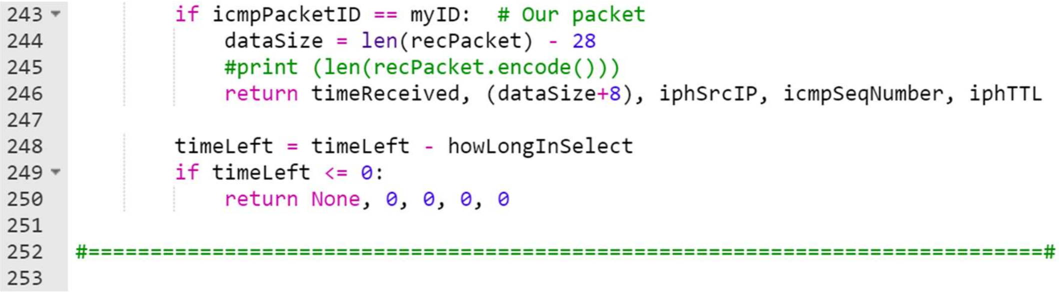


```python
    # Calculate the checksum on the data and the dummy header.
    myChecksum = checksum(header + data)  # Checksum is in network order

    # Now that we have the right checksum, we put that in. It's just easier
    # to make up a new header than to stuff it into the dummy.
    header = struct.pack(
        "!BBHHH", ICMP_ECHO, 0, myChecksum, myID, mySeqNumber
    )

    packet = header + data

    sendTime = default_timer()

    try:
        # Port number is irrelevant for ICMP
        mySocket.sendto(packet, (destIP, 1))
    except socket.error as e:
        print("General failure (%s)" % (e.args[1]))
        f1.write("General failure (%s)" % (e.args[1]))
        return

    return sendTime

#=============================================================================#

```

On the other end, receiving the ping from the socket:

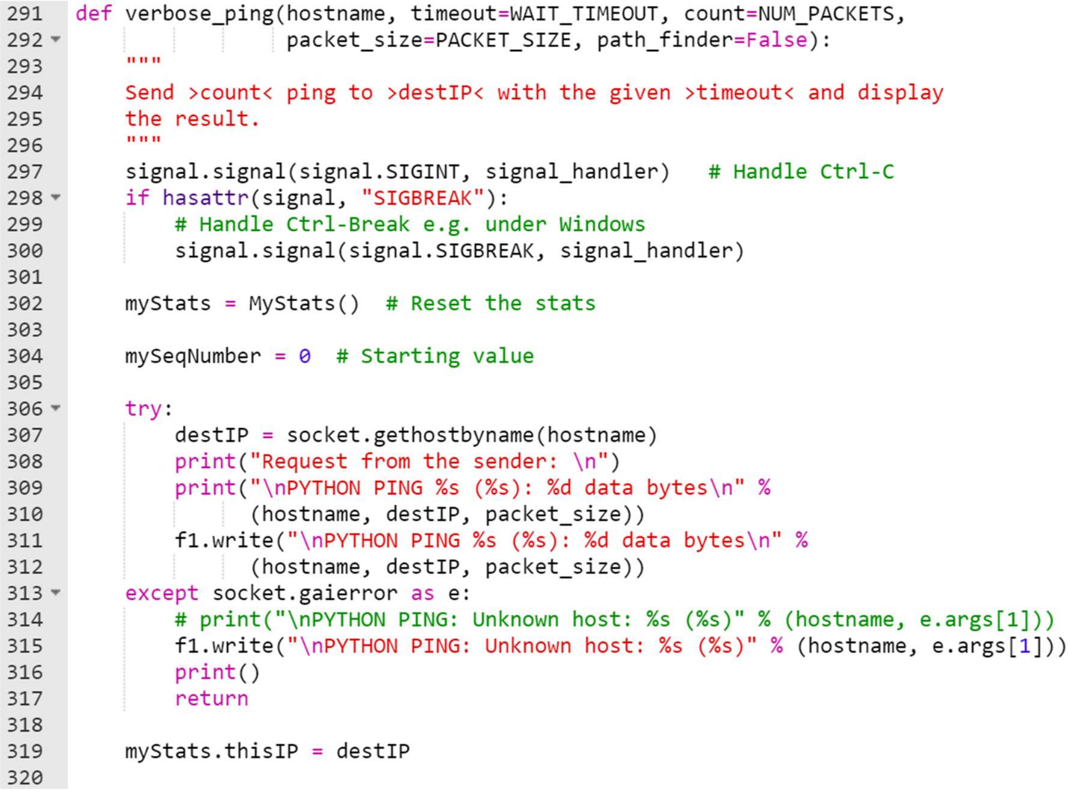


```python

def receive_one_ping(mySocket, myID, timeout):
    """
    Receive the ping from the socket. Timeout = in ms
    """
    timeLeft = timeout/1000

    while True:  # Loop while waiting for packet or timeout
        startedSelect = default_timer()
        whatReady = select.select([mySocket], [], [], timeLeft)
        howLongInSelect = (default_timer() - startedSelect)
        if whatReady[0] == []:  # Timeout
            return None, 0, 0, 0, 0

        timeReceived = default_timer()

        recPacket, addr = mySocket.recvfrom(ICMP_MAX_RECV)

        ipHeader = recPacket[:20]
        iphVersion, iphTypeOfSvc, iphLength, \
            iphID, iphFlags, iphTTL, iphProtocol, \
            iphChecksum, iphSrcIP, iphDestIP = struct.unpack(
                "!BBHHHBBHII", ipHeader
            )

        icmpHeader = recPacket[20:28]
        icmpType, icmpCode, icmpChecksum, \
            icmpPacketID, icmpSeqNumber = struct.unpack(
                "!BBHHH", icmpHeader
            )

```

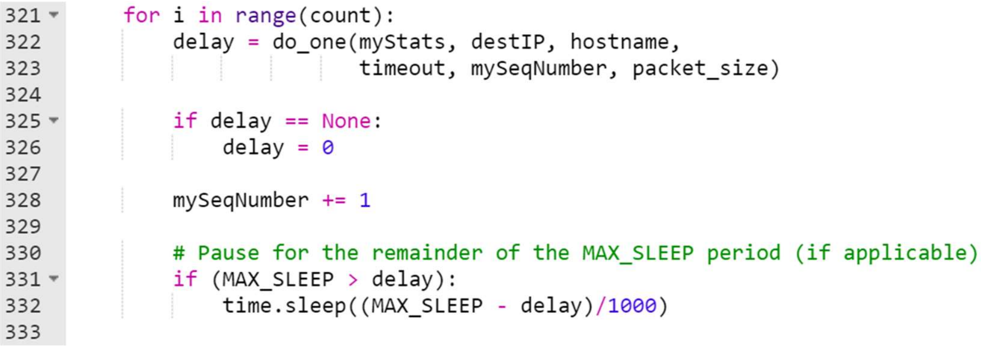


```
        if icmpPacketID == myID:  # Our packet
            dataSize = len(recPacket) - 28
            #print (len(recPacket.encode()))
            return timeReceived, (dataSize+8), iphSrcIP, icmpSeqNumber, iphTTL

        timeLeft = timeLeft - howLongInSelect
        if timeLeft <= 0:
            return None, 0, 0, 0, 0

#=============================================================================#

```

Now, the main function occurs here. Sending out 2 ICMP requests constantly with the length of 256 bytes to the destination IP address and the receiver sends back replies and print the requests (messages, not the statistics) it receives from the sender to a requests.txt file:

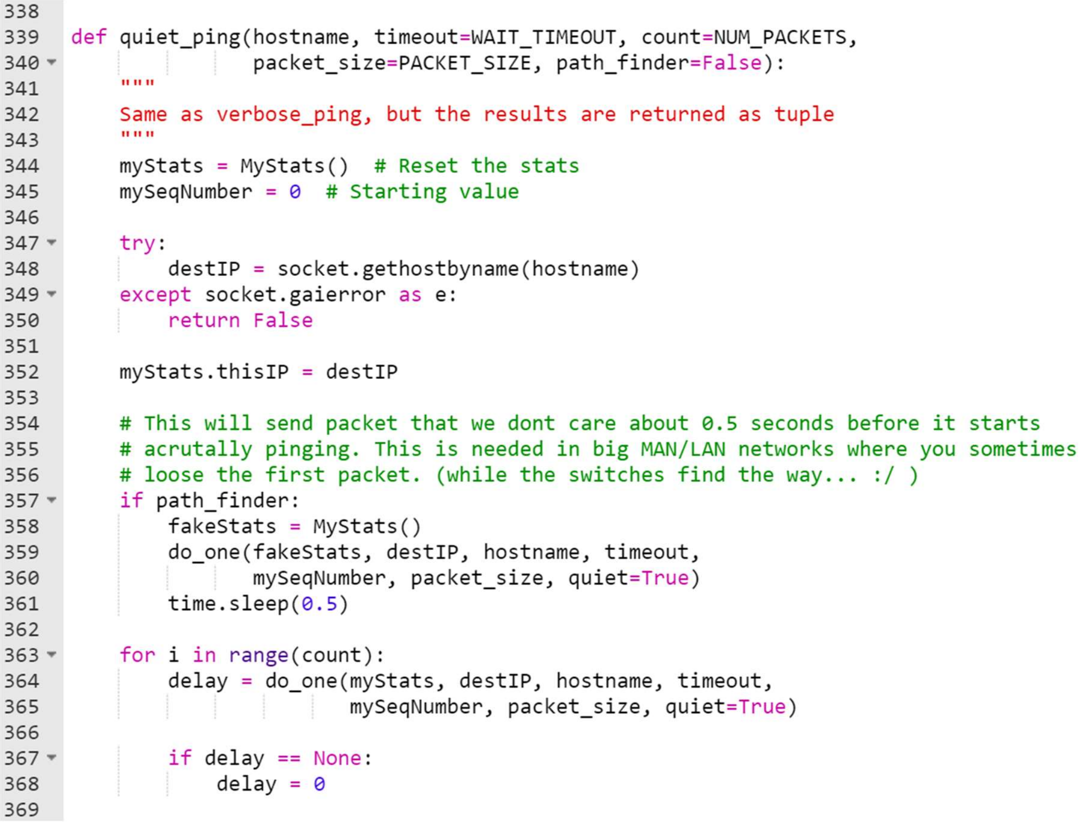


```
def verbose_ping(hostname, timeout=WAIT_TIMEOUT, count=NUM_PACKETS,
                 packet_size=PACKET_SIZE, path_finder=False):
    """
    Send >count< ping to >destIP< with the given >timeout< and display
    the result.
    """
    signal.signal(signal.SIGINT, signal_handler)   # Handle Ctrl-C
    if hasattr(signal, "SIGBREAK"):
        # Handle Ctrl-Break e.g. under Windows
        signal.signal(signal.SIGBREAK, signal_handler)

    myStats = MyStats()  # Reset the stats

    mySeqNumber = 0  # Starting value

    try:
        destIP = socket.gethostbyname(hostname)
        print("Request from the sender: \n")
        print("\nPYTHON PING %s (%s): %d data bytes\n" %
              (hostname, destIP, packet_size))
        f1.write("\nPYTHON PING %s (%s): %d data bytes\n" %
              (hostname, destIP, packet_size))
    except socket.gaierror as e:
        # print("\nPYTHON PING: Unknown host: %s (%s)" % (hostname, e.args[1]))
        f1.write("\nPYTHON PING: Unknown host: %s (%s)" % (hostname, e.args[1]))
        print()
        return

    myStats.thisIP = destIP

```

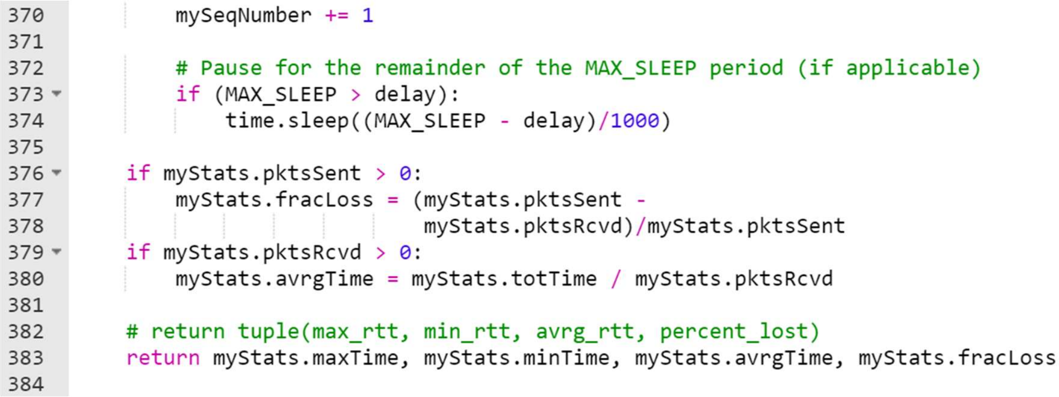


```
    for i in range(count):
        delay = do_one(myStats, destIP, hostname,
                       timeout, mySeqNumber, packet_size)

        if delay == None:
            delay = 0

        mySeqNumber += 1

        # Pause for the remainder of the MAX_SLEEP period (if applicable)
        if (MAX_SLEEP > delay):
            time.sleep((MAX_SLEEP - delay)/1000)

```

The sender receives replies, it prints the reply messages (not the statistics) on the screen and in a reply.txt file as well:

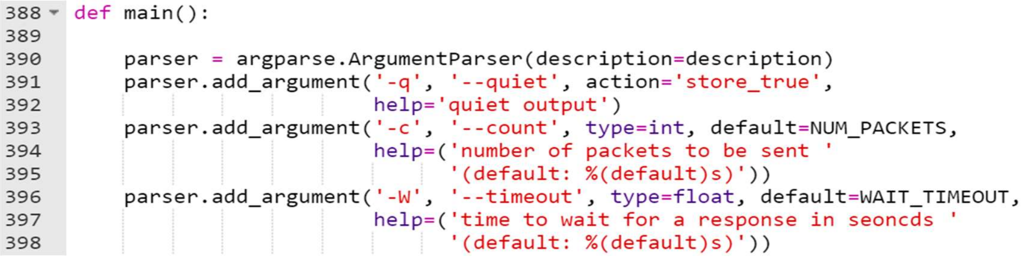


```python

def quiet_ping(hostname, timeout=WAIT_TIMEOUT, count=NUM_PACKETS,
               packet_size=PACKET_SIZE, path_finder=False):
    """
    Same as verbose_ping, but the results are returned as tuple
    """
    myStats = MyStats()  # Reset the stats
    mySeqNumber = 0  # Starting value

    try:
        destIP = socket.gethostbyname(hostname)
    except socket.gaierror as e:
        return False

    myStats.thisIP = destIP

    # This will send packet that we dont care about 0.5 seconds before it starts
    # acrutally pinging. This is needed in big MAN/LAN networks where you sometimes
    # loose the first packet. (while the switches find the way... :/ )
    if path_finder:
        fakeStats = MyStats()
        do_one(fakeStats, destIP, hostname, timeout,
               mySeqNumber, packet_size, quiet=True)
        time.sleep(0.5)

    for i in range(count):
        delay = do_one(myStats, destIP, hostname, timeout,
                       mySeqNumber, packet_size, quiet=True)

        if delay == None:
            delay = 0

```

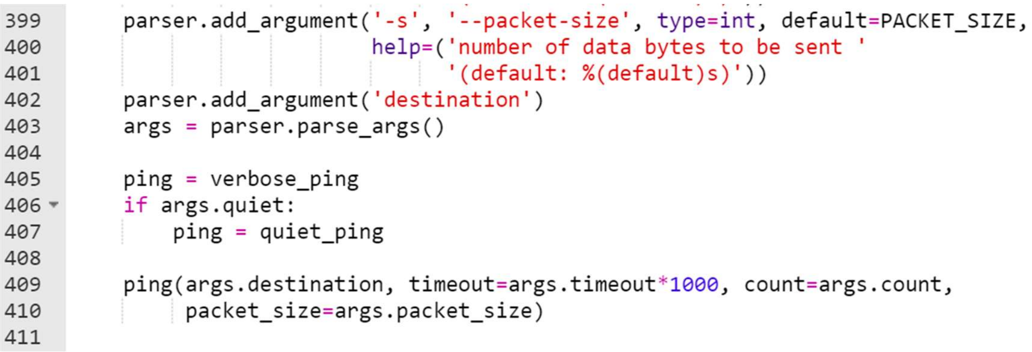


```python
        mySeqNumber += 1

        # Pause for the remainder of the MAX_SLEEP period (if applicable)
        if (MAX_SLEEP > delay):
            time.sleep((MAX_SLEEP - delay)/1000)

    if myStats.pktsSent > 0:
        myStats.fracLoss = (myStats.pktsSent -
                            myStats.pktsRcvd)/myStats.pktsSent
    if myStats.pktsRcvd > 0:
        myStats.avrgTime = myStats.totTime / myStats.pktsRcvd

    # return tuple(max_rtt, min_rtt, avrg_rtt, percent_lost)
    return myStats.maxTime, myStats.minTime, myStats.avrgTime, myStats.fracLoss

```

Finally, calling the functions in the main function for the implementation of ICMP echo request and reply:

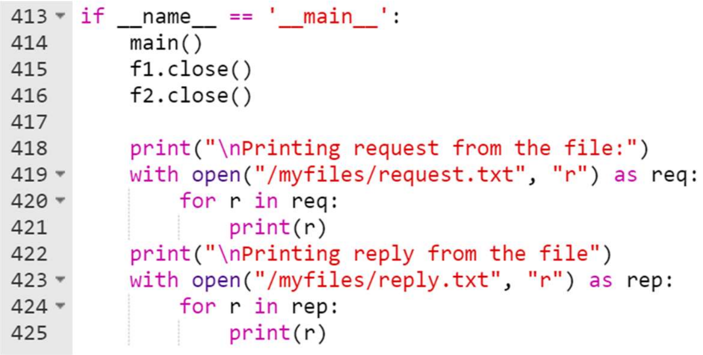


```python
def main():

    parser = argparse.ArgumentParser(description=description)
    parser.add_argument('-q', '--quiet', action='store_true',
                        help='quiet output')
    parser.add_argument('-c', '--count', type=int, default=NUM_PACKETS,
                        help=('number of packets to be sent '
                              '(default: %(default)s)'))
    parser.add_argument('-W', '--timeout', type=float, default=WAIT_TIMEOUT,
                        help=('time to wait for a response in seoncds '
                              '(default: %(default)s)'))
```



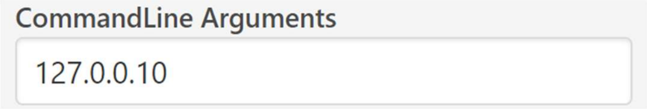
```
    parser.add_argument('-s', '--packet-size', type=int, default=PACKET_SIZE,
                        help=('number of data bytes to be sent '
                              '(default: %(default)s)'))
    parser.add_argument('destination')
    args = parser.parse_args()

    ping = verbose_ping
    if args.quiet:
        ping = quiet_ping

    ping(args.destination, timeout=args.timeout*1000, count=args.count,
         packet_size=args.packet_size)

```

Finally:



```
if __name__ == '__main__':
    main()
    f1.close()
    f2.close()

    print("\nPrinting request from the file:")
    with open("/myfiles/request.txt", "r") as req:
        for r in req:
            print(r)
    print("\nPrinting reply from the file")
    with open("/myfiles/reply.txt", "r") as rep:
        for r in rep:
            print(r)
```

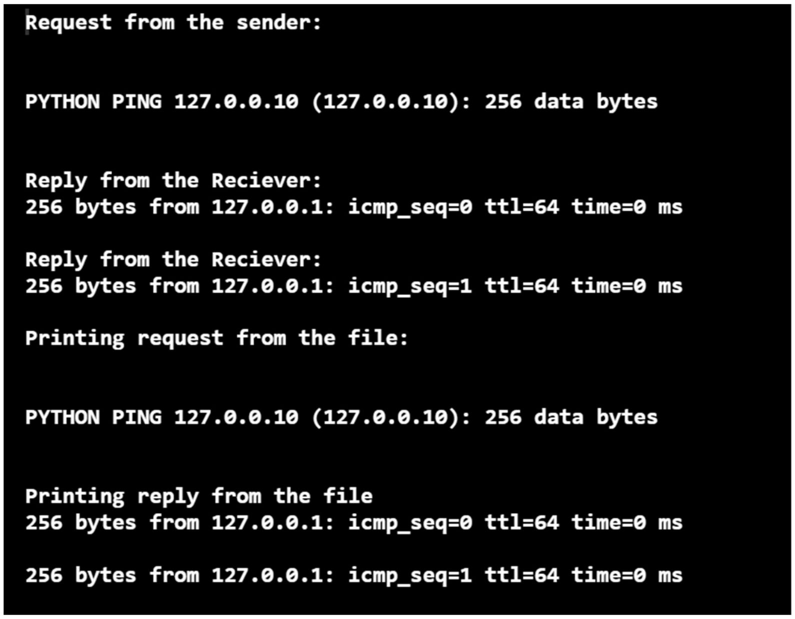
Here, the destination IP address given as input is:



CommandLine Arguments

127.0.0.10

Output screenshot:



```
Request from the sender:


PYTHON PING 127.0.0.10 (127.0.0.10): 256 data bytes


Reply from the Reciever:
256 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0 ms

Reply from the Reciever:
256 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0 ms

Printing request from the file:


PYTHON PING 127.0.0.10 (127.0.0.10): 256 data bytes


Printing reply from the file
256 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0 ms

256 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0 ms
```