Xueyang (Sean) Wang

XW1154

04/09/2017

CS4793

Homework - Program 03

Python Code for Pinger:

import socket

import os

import sys

import struct

import time

import select

import binascii

ICMP\_ECHO\_REQUEST = 8

timeRTT = []

packageSent =0;

packageRev = 0;

def checksum(str):

csum = 0

countTo = (len(str) / 2) \* 2

count = 0

while count < countTo:

thisVal = str[count+1] \* 256 + str[count]

csum = csum + thisVal

csum = csum & 0xffffffff

count = count + 2

if countTo < len(str):

csum = csum + ord(str[len(str) - 1])

csum = csum & 0xffffffff

csum = (csum >> 16) + (csum & 0xffff)

csum = csum + (csum >> 16)

answer = ~csum

answer = answer & 0xffff

answer = answer >> 8 | (answer << 8 & 0xff00)

return answer

def receiveOnePing(mySocket, ID, timeout, destAddr):

global packageRev, timeRTT

timeLeft = timeout

while 1:

startedSelect = time.time()

whatReady = select.select([mySocket], [], [], timeLeft)

howLongInSelect = (time.time() - startedSelect)

if whatReady[0] == []: # Timeout

return ("Request timed out. Not Ready")

timeReceived = time.time()

recPacket, addr = mySocket.recvfrom(1024)

#Fill in start

#Fetch the ICMP header from the IP packet

icmpHeader = recPacket[20:28]

requestType, code, revChecksum, revId, revSequence = struct.unpack('bbHHh',icmpHeader)

if ID == revId:

bytesInDouble = struct.calcsize('d')

timeData = struct.unpack('d',recPacket[28:28 + bytesInDouble])[0]

timeRTT.append(timeReceived - timeData)

packageRev += 1

return timeReceived - timeData

else:

return ("Mismatched!")

#Fill in end

timeLeft = timeLeft - howLongInSelect

if timeLeft <= 0:

return ("Request timed out. Waited")

def sendOnePing(mySocket, destAddr, ID):

# Header is type (8), code (8), checksum (16), id (16), sequence (16)

myChecksum = 0

# Make a dummy header with a 0 checksum.

# struct -- Interpret strings as packed binary data

header = struct.pack("bbHHh", ICMP\_ECHO\_REQUEST, 0, myChecksum, ID, 1)

data = struct.pack("d", time.time())

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

myChecksum = checksum(header + data)

# Get the right checksum, and put in the header

if sys.platform == 'darwin':

myChecksum = socket.htons(myChecksum) & 0xffff

#Convert 16-bit integers from host to network byte order.

else:

myChecksum = socket.htons(myChecksum)

header = struct.pack("bbHHh", ICMP\_ECHO\_REQUEST, 0, myChecksum, ID, 1)

packet = header + data

mySocket.sendto(packet, (destAddr, 1)) # AF\_INET address must be tuple, not str

#Both LISTS and TUPLES consist of a number of objects

#which can be referenced by their position number within the object

def doOnePing(destAddr, timeout):

icmp = socket.getprotobyname("icmp")

#SOCK\_RAW is a powerful socket type. For more details see: http://sock-raw.org/papers/sock\_raw

#Fill in start

#Create Socket here

# try:

mySocket = socket.socket(socket.AF\_INET, socket.SOCK\_RAW, icmp)

# except socket.error as errno:

# if errno == 1:

# raise socket.error(msg)

#Fill in end

myID = os.getpid() & 0xFFFF #Return the current process i

sendOnePing(mySocket, destAddr, myID)

delay = receiveOnePing(mySocket, myID, timeout, destAddr)

mySocket.close()

return delay

def ping(host, timeout=1):

#timeout=1 means: If one second goes by without a reply from the server,

#the client assumes that either the client’s ping or the server’s pong is lost

dest = socket.gethostbyname(host)

print ("Pinging " + dest + " using Python:")

print ("")

#Send ping requests to a server separated by approximately one second

while 1 :

delay = doOnePing(dest, timeout)

print (delay)

time.sleep(1)# one second

return delay

ping("127.0.0.1")

#ping("www.baidu.com")

#China

#ping("www.google.com")

#U.S.

#ping ("Thelocal.de")

#Europe

#ping("www.pichunter.com")

#Africa

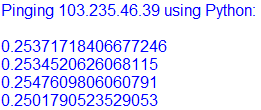


Figure 1. Pinging Asia

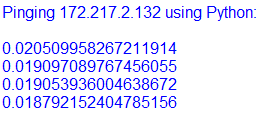


Figure 2. Pinging North America

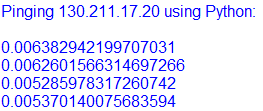


Figure 3. Pinging Europe

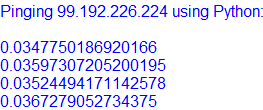


Figure 4. Pinging Africa

Python Code for Trace- Route:

import socket

import os

import sys

import struct

import time

import select

import binascii

label = '\*\*\*\*\*\*\*\*\*\*\*\*\*{0}\*\*\*\*\*\*\*\*\*\*\*\*\*'

ICMP\_ECHO\_REQUEST = 8

MAX\_HOPS = 30

TIMEOUT = 2.0

TRIES = 2

# The packet that we shall send to each router along the path is the ICMP echo

# request packet, which is exactly what we had used in the ICMP ping exercise.

# We shall use the same packet that we built in the Ping exercise

def checksum(str):

# In this function we make the checksum of our packet

csum = 0

countTo = (len(str) / 2) \* 2

count = 0

while count < countTo:

thisVal = str[count+1] \* 256 + str[count]

csum = csum + thisVal

csum = csum & 0xffffffff

count = count + 2

if countTo < len(str):

csum = csum + ord(str[len(str) - 1])

csum = csum & 0xffffffff

csum = (csum >> 16) + (csum & 0xffff)

csum = csum + (csum >> 16)

answer = ~csum

answer = answer & 0xffff

answer = answer >> 8 | (answer << 8 & 0xff00)

return answer

def build\_packet():

#create header and append check sum, Header is type (8), code (8), checksum (16), id (16), seq (16)

myChecksum = 0

myID = os.getpid() & 0xFFFF #Return the current process i

# Make a dummy header with a 0 checksum.

# struct -- Interpret strings as packed binary data

header = struct.pack("bbHHh", ICMP\_ECHO\_REQUEST, 0, myChecksum, myID, 1)

data = struct.pack("d", time.time())

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

myChecksum = checksum(header + data)

# Get the right checksum, and put in the header

if sys.platform == 'darwin':

myChecksum = socket.htons(myChecksum) & 0xffff

#Convert 16-bit integers from host to network byte order.

else:

myChecksum = socket.htons(myChecksum)

header = struct.pack("bbHHh", ICMP\_ECHO\_REQUEST, 0, myChecksum, myID, 1)

packet = header + data

return packet

def get\_route(hostname):

timeLeft = TIMEOUT

for ttl in range(1,MAX\_HOPS):

for tries in range(TRIES):

destAddr = socket.gethostbyname(hostname)

#Fill in start

# Make a raw socket named mySocket

icmp = socket.getprotobyname("icmp")

mySocket = socket.socket(socket.AF\_INET, socket.SOCK\_RAW, icmp)

#Fill in end

mySocket.setsockopt(socket.IPPROTO\_IP, socket.IP\_TTL, struct.pack('I', ttl))

mySocket.settimeout(TIMEOUT)

try:

d = build\_packet()

mySocket.sendto(d, (hostname, 0))

t= time.time()

startedSelect = time.time()

whatReady = select.select([mySocket], [], [], timeLeft)

howLongInSelect = (time.time() - startedSelect)

if whatReady[0] == []: # Timeout

print( " \* \* \* Request timed out.")

recvPacket, addr = mySocket.recvfrom(1024)

timeReceived = time.time()

timeLeft = timeLeft - howLongInSelect

if timeLeft <= 0:

print (" \* \* \* Request timed out.")

except socket.timeout:

continue

else:

#Fill in start

#Fetch the icmp type from the IP packet

icmpHeaderContent = recvPacket[20:28]

type, code, checksum, packetID, sequence = struct.unpack("bbHHh", icmpHeaderContent)

#Fill in end

if type == 11:

bytes = struct.calcsize("d")

timeSent = struct.unpack("d", recvPacket[28:28 + bytes])[0]

print (" %d rtt=%.0f ms %s" %(ttl, (timeReceived -t)\*1000, addr[0]))

elif type == 3:

bytes = struct.calcsize("d")

timeSent = struct.unpack("d", recvPacket[28:28 + bytes])[0]

print (" %d rtt=%.0f ms %s" %(ttl, (timeReceived-t)\*1000, addr[0]))

elif type == 0:

bytes = struct.calcsize("d")

timeSent = struct.unpack("d", recvPacket[28:28 + bytes])[0]

print (" %d rtt=%.0f ms %s" %(ttl, (timeReceived - timeSent)\*1000, addr[0]))

return

else:

print ("error")

break

finally:

mySocket.close()

get\_route("127.0.0.1")

get\_route("www.baidu.com")

#China

get\_route("www.google.com")

#U.S.

get\_route ("Thelocal.de")

#Europe

get\_route("www.pichunter.com")

#Africa



Figure 5. Testing Tracer Route



Figure 6. Tracing Asia



Figure 7. Tracing North America



Figure 8. Tracing Europe



Figure 9. Tracing Africa