|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Traffic intensity | S1 buffer capacity | ping packet loss % | ping min RTT reported | ping avg RTT reported | ping max RTT reported | Observed throughput at h2 |
| Config 0 | 0 | 1000 | 0.0. | 0.014138 | 0.0178200392157 | 0.022685 | 12.6 Kbits/sec |
| Config 1 | 40 | 1000 | 23.5294117647 % | 0.03861 | 0.0352232745098 | 0.051615 | 18.5 |
| Config 2 | 60 | 1000 | 1.96078431373 % | 0.042662 | 0.0453943137255 | 0.051811 | 32.3 Mbits/sec |
| Config 3 | 40 | 500 | 1.96078431373 % | 0.029661 | 0.0366325882353 | 0.047382 | 23.9Mbits/sec |
| Config 4 | 60 | 500 | 29.4117647059 % | 0.0 | 0.0162447254902 | 0.025486 | 33.6 Mbits/sec |

Table when I was executing the client and server multiple times to get a packet loss

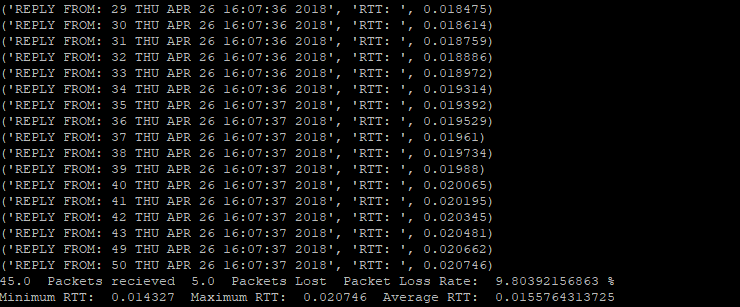
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Traffic intensity | S1 buffer capacity | ping packet loss % | ping min RTT reported | ping avg RTT reported | ping max RTT reported | Observed throughput at h2 |  |
| Config 0 | 0 | 1000 | 0.0. | 0.01808 | 0.025335 | 0.025335 | 12.6 Kbits/sec | To establish the baseline numbers for ping (at zero background traffic) |
| Config 1 | 40 | 1000 | 0% | .02884 | 0.0342283921569 | 0.042881 | 28.5 Mbits/sec | Light traffic |
| Config 2 | 60 | 1000 | 0% | 0.031472 | 0.0362324901961 | 0.04408 | 33.9 Mbits/sec | Heavy traffic |
| Config 3 | 40 | 500 | 0% | 0.025644 | 0.0326547058824 | 0.038445 | 29.5Mbits/sec | Light traffic |
| Config 4 | 60 | 500 | 1.96078431373% | 0.028057 | 0.0365521176471 | 0.051154 | 32.5 Mbits/sec | Heavy  traffic |

Table made when I followed the instructions.

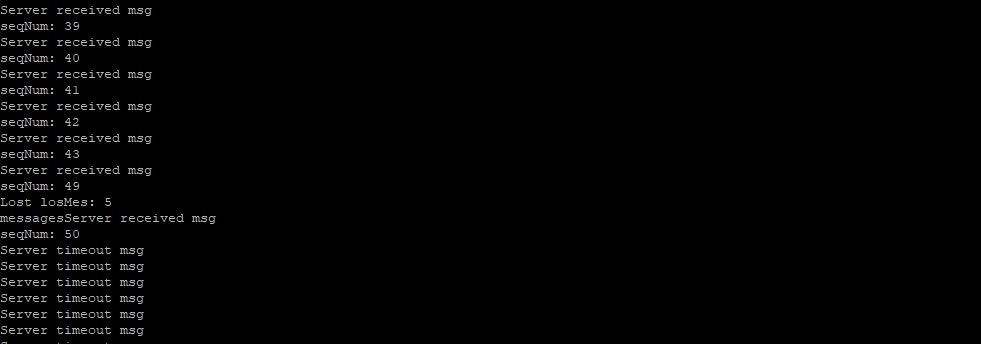
I had problems with the UDP Pinger code. The UDP Pinger client would not always record a packet loss. It had to be run several times to record a packet loss and was unreliable. This only happened when the buffer capacity was relatively high of 200 or more. The only time it could give a reliable packet loss is if the buffer capacity was 100 or less. I checked with the professor the code for part 1 of the UDP Pinger was correct. There was something wrong with mininet. To get the packet loss, I wanted I had to go against requirements and artificially inject them in to the UDP server, but the packet losses were too high, and it went against instructions, so I changed it back to how it was originally. The first topmost table that is filled above is inaccurate because of my rerunning it multiple times so that I can get a packet loss and the fact that i2.out would not always be written despite my typing h2 iperf -s -u > i2.out & . The respective packet loss when the buffer capacity was equal to hundred was 4%, 9%, and 3.9% when testing it with the code I made for part 1 of the UDP Pinger project. Another aspect that was wrong with it was that fact it seemed to only allow users to write one file at a time and no files running in the foreground can run. During the uncommon times where it did write to the file, particularly with the Heartbeat server, it would not write completely. It had to be executed twice for it to finish running when I had that problem. It seems to have something to do with possibly the configuration I have with it. To make 2 files output to a file via mininet I had to execute it several times, which was not rational. For the UDP heartbeat server, I essentially made the executing python file itself write data into a file specified by the user instead of “mininet. In other words, I essentially told the program to write data into a specified file given by the user. Installing mininet and making sure it works properly is a little arduous. The configuration problems may be hard to detect as they sometimes work, and they sometimes do not work. When I was following the instructions of the heartbeat server writing data to a file following the instructions, no data was written. Sometimes it was and sometimes it was not. I suggest that students like me spend more time installing it, and teachers should give tests to make sure it works.



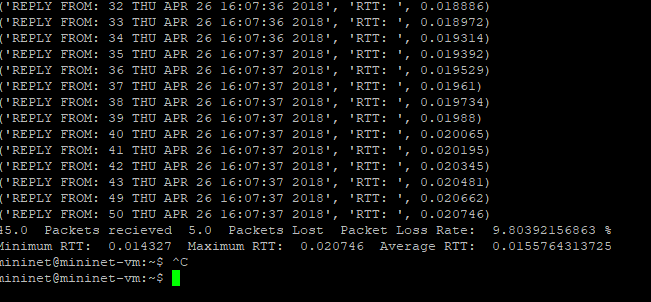
Commands to execute Heatbeat function



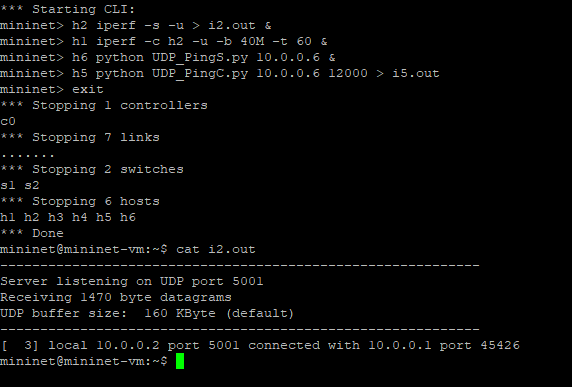
UDP\_HbC.py executing



UDP\_HbS.py executing



UDP\_PingC running at a traffic intensity of 60 and a buffer capacity of 1000



I2.out failing to be written into

#**UDP\_HbS.py**

from socket import \*

import sys

import random

argv = sys.argv

serverIPaddress = argv[1]

messageSplit = "";

words = []

checkSeq = []

lasSeq = 0

preSeq = 0

seqNum = 0

counter = 0

losMes = 1 # number of lost messeges

# Create a UDP socket

# Notice the use of SOCK\_DGRAM for UDP packets

serverSocket = socket(AF\_INET, SOCK\_DGRAM)

time = 1 # When equal 50 or maxTimeOuts server stops

maxTimeOuts = 50

serverSocket.settimeout(1)

# Assign IP address and port number to socket, use port 12000

#print counter

serverSocket.bind((serverIPaddress, 12000))

with open(argv[2], "w") as f:

while (time <= maxTimeOuts):

try:

#Receive the client packet along with the address it is coming from

message, address = serverSocket.recvfrom(1024)

#f.write("packet recieved from before Recv. client\ncounter: %d\n" % counter)

#f.flush()

if(isinstance(message, basestring)):

messageSplit = message

#print "message is a string"

words = messageSplit.split()

seqNum = int(words[2])

#checkSeq.append(seqNum)

preSeq = seqNum

#print checkSeq[counter]

#lostMes = int(checkSeq[counter]) - int(checkSeq[counter - 1])

losMes = (preSeq -1) - lasSeq

#f.flush()

else:

print "message is not a string"

print 'Server received msg ', seqNum

lasSeq = seqNum

f.write("Server received msg\nseqNum: %d\n" % seqNum)

f.flush()

if(losMes > 1):

#print "Lost ", lostMes, " messages"

f.write("Lost losMes: %d\n" % losMes)

f.write("messages")

f.flush()

# Capitalize the message from the client

message = message.upper()

# Server responds

serverSocket.sendto(message, address)

except timeout:

print "Server timeout"

f.write("Server timeout msg\n")

f.flush()

time += 1

#**UDP\_HbC.py**

# UDP\_HbC.py

from socket import \*

import sys

import time

argv = sys.argv

clientIPaddress = argv[1]

n = argv[3]

m = argv[4]

n = int(n)

m = int(m)

if(m < n):

print "the second argument should be less than the third argument"

exit()

#serverName = '10.0.0.6'

serverPort = 12000

clientSocket = socket(AF\_INET, SOCK\_DGRAM)

minRTT = 0.0

maxRTT = 0.0

#tempRTT

RTTAcum = 0.0

avgRTT = 0.0

pktLost = 0.0 # number of packets lost

pktRec = 0.0 # number of packets recieved

pktLoss = 0.0 # packet loss percentage. Initially value given to make sure it stores floating numbers

seqNumTime = time.localtime()

#counter = 0

clientSocket.settimeout(1)

seqNum = 1 # Sequence number intial value

while (seqNum <= 50): # if m 48 and n 44. Don't send message 44 through 48

if(seqNum == n): # 43 already sent and then incremented to 44.

seqNum = m + 1 # only send 48 + 1. If a message is not sent to begin with I do not think there can be a timeout

pktLost += (seqNum - n)

#print "sequence number equals ", seqNum

message = 'reply from: ' + str(seqNum) + ' '+ str(time.asctime(seqNumTime))

seqNumTime = time.localtime() # This will be a little off as it gets the time just before the message is sent

RTT = time.clock() # Hold RTT of a message, which equals the amount of time since the message

# was sent to when client recieved modified message from server.

# It is a little off as it is calculate a little after message is sent

clientSocket.sendto(message,(clientIPaddress,serverPort)) # Send ping message

try:

modifiedMessage, serverAddress = clientSocket.recvfrom(1024)

print(modifiedMessage, "RTT: ", RTT) # print modified message

if(seqNum == 1) :

minRTT = RTT

if(minRTT > RTT) :

minRTT = RTT

if(maxRTT < RTT) : # The max RTT will not be greater or equal to 1 second

maxRTT = RTT

RTTAcum += RTT;

pktRec +=1

except timeout:

print 'Request time out', seqNum

pktLost += 1; # does a ping mean one connection that results in a timout as well?

seqNum += 1; # Sequence number increment every successful ping

avgRTT = RTTAcum / seqNum;

pktLoss = (pktLost / seqNum) \* 100

print pktRec,' Packets recieved ', pktLost, ' Packets Lost ', 'Packet Loss Rate: ', pktLoss,'%' #

print "Minimum RTT: ", minRTT, " Maximum RTT: ", maxRTT, " Average RTT: ", avgRTT

clientSocket.close()

# **UDP\_PingS.py**

from socket import \*

import sys

import random

argv = sys.argv

serverIPaddress = argv[1]

# Create a UDP socket

# Notice the use of SOCK\_DGRAM for UDP packets

serverSocket = socket(AF\_INET, SOCK\_DGRAM)

#serverSocket.settimeout(1)

# Assign IP address and port number to socket, use port 12000

serverSocket.bind((serverIPaddress, 12000))

while True:

# Receive the client packet along with the address it is coming from

message, address = serverSocket.recvfrom(1024)

# Capitalize the message from the client

message = message.upper()

# Server responds

serverSocket.sendto(message, address)

# **UDP\_PingC.py**

from socket import \*

import sys

import time

argv = sys.argv

clientIPaddress = argv[1]

#serverName = '10.0.0.6'

serverPort = 12000

clientSocket = socket(AF\_INET, SOCK\_DGRAM)

minRTT = 0.0;

maxRTT = 0.0;

#tempRTT;

RTTAcum = 0.0;

import sys

import time

argv = sys.argv

clientIPaddress = argv[1]

#serverName = '10.0.0.6'

serverPort = 12000

clientSocket = socket(AF\_INET, SOCK\_DGRAM)

minRTT = 0.0;

maxRTT = 0.0;

#tempRTT;

RTTAcum = 0.0;

avgRTT = 0.0;

pktLost = 0.0; # number of packets lost

pktRec = 0.0 # number of packets recieved

pktLoss = 0.0; # packet loss percentage. Initially value given to make sure it stores floating numbers

seqNumTime = time.localtime()

#counter = 0;

seqNum = 1; # Sequence number intial value

clientSocket.settimeout(1)

while (seqNum <= 50):

message = 'reply from: ' + str(seqNum) + ' '+ str(time.asctime(seqNumTime))

seqNumTime = time.localtime() # This will be a little off as it gets the time just before the message is sent

RTT = time.clock() # Hold RTT of a message, which equals the amount of time since the message

# was sent to when client recieved modified message from server.

# It is a little off as it is calculate a little after message is sent

# clientSocket.sendto(message,(clientIPaddress,serverPort)) # Send ping message

clientSocket.sendto(message,(clientIPaddress,serverPort)) # Send ping message

try:

modifiedMessage, serverAddress = clientSocket.recvfrom(1024)

print(modifiedMessage, "RTT: ", RTT) # print modified message

if(seqNum == 1) :

minRTT = RTT;

if(minRTT > RTT) :

minRTT = RTT;

if(maxRTT < RTT) : # The max RTT will not be greater or equal to 1 second

maxRTT = RTT;

RTTAcum += RTT;

pktRec +=1

except timeout:

print 'Request time out', seqNum

pktLost += 1; # does a ping mean one connection that results in a timout as well?

#clientSocket.settimeout(1)

seqNum += 1; # Sequence number increment every successful ping

avgRTT = RTTAcum / seqNum;

pktLoss = (pktLost / seqNum) \* 100

print pktRec,' Packets recieved ', pktLost, ' Packets Lost ', 'Packet Loss Rate: ', pktLoss,'%' #

print "Minimum RTT: ", minRTT, " Maximum RTT: ", maxRTT, " Average RTT: ", avgRTT

clientSocket.close()