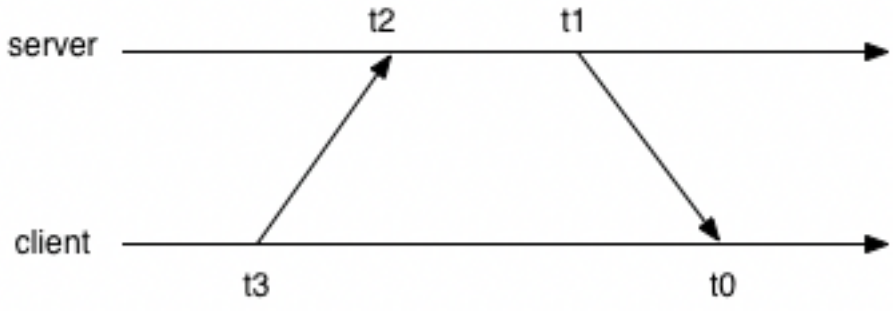
Ella Hayashi
V00184392
CSC462

Simple UDP-based time synchronization client
CSC 462
May 2019

Introduction

The source code below is the NTP client code. The client pings an NTP server, and the data recorded is the calculated average round trip time, the offset, as well as the times t0 (client packet time of arrival), t1 (server packet time of departure), t2 (server packet time of arrival) and t3 (client code time of departure). This may be easier to visualize with the following graph.



The code has a timeout period of 3 seconds, and it pings the server every 10 seconds. Round trip time is calculated as RTT = (t2 - t3) + (t0 - t1), and Offset/Drift is calculated as Θ = ((t2 - t3) - (t0 - t1)) / 2. This document will show the results of the client code running for about 3 hours, pinging the server 1000 times in this duration.

Client Code

```
import datetime
import time
from time import mktime
from datetime import datetime
import socket
import struct
import sys
import time

NTP_SERVER = "0.uk.pool.ntp.org"
TIME1970 = 2208988800
```

Ella Hayashi
V00184392
CSC462

```
def sntp_client(x, c):
   client = socket.socket( socket.AF_INET, socket.SOCK_DGRAM )
   data = '\x1b' + 47 * '\0'
   client.settimeout(3)
   t3 = time.time()
   print('t3 time is: ')
   print datetime.fromtimestamp(t3)
   print('\n')
   client.sendto( data.encode('utf-8'), ( NTP_SERVER, 123 ))
   try:
      data, address = client.recvfrom( 1024 )
      if data:
         t0 = time.time()
         NTP_PF = '!12I'
         unpacked = struct.unpack(NTP_PF, data[0:struct.calcsize(NTP_PF)])
         t1 = unpacked[10] + float(unpacked[11]) /2**32
         t2 = unpacked[8] + float(unpacked[9]) /2**32
         t2 -= TIME1970
         t1 -= TIME1970
         print ('t2 time is: ')
         print datetime.fromtimestamp(t2)
         print('\nt1 time is: ')
         print datetime.fromtimestamp(t1)

      t = struct.unpack( '!12I', data )[10]
      t -= TIME1970
      print ('\nt0 time is: ')
      print datetime.fromtimestamp(t0)

      RTT = (t2-t3) + (t0-t1)
      Drift = ((t2-t3)-(t0-t1))/2
      line =
[str(x),str(RTT),str(Drift),str(datetime.fromtimestamp(t0)),str(datetime.fromtimestamp(t1)),str(datetime.fromtime
stamp(t2)),str(datetime.fromtimestamp(t3))]
      f.write(','.join(line)+'\n')
   except socket.timeout:
      print('\ntimeout\n')
      f.write(str(x) + 'timeout\n')

if __name__ == '__main__':
   x=0
   with open("462_smtp2.csv", 'w') as f:
      f.write('number, RTT, Drift, t0, t1, t2, t3\n')
      while x<1000:
         x+=1
         print("\n\n\n*******\n\n\n")
         sntp_client(x, f)
         time.sleep(10)
```

Ella Hayashi
V00184392
CSC462

Data

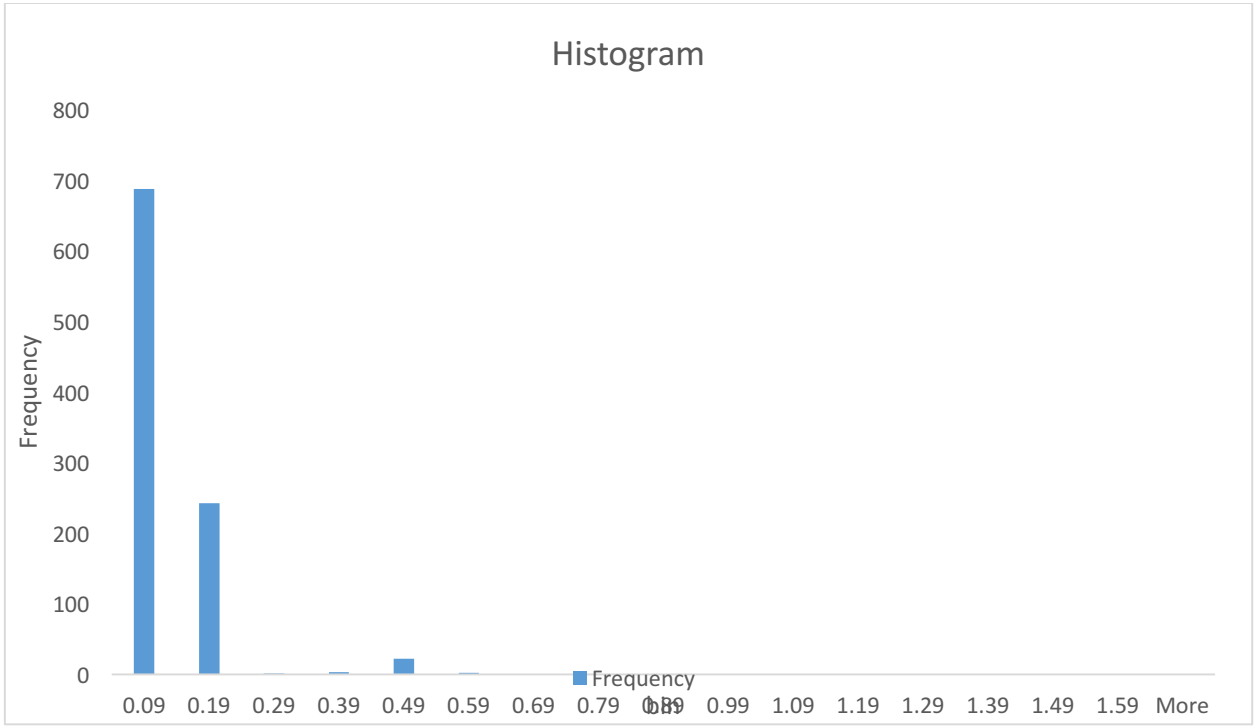
Time outs: There were 34 timeouts over the 1000 iterations, meaning there was an average loss rate of 0.001%.

Average Round trip time: The average round trip time was calculated to be 0.18415476

Average Drift: The average Drift was calculated to be 0.08080955

bin	*Frequency*
0.09	688
0.19	243
0.29	2
0.39	4
0.49	23
0.59	3
0.69	1
0.79	0
0.89	0
0.99	0
1.09	0
1.19	0
1.29	0
1.39	0
1.49	1
1.59	1
More	0

Ella Hayashi
V00184392
CSC462



The histogram above represents the offset averages for the data for the 1000 itterations, with most of the data landing in the range of 0.0 to 0.1.

Questions

1. **how did you calculate your client's average clock drift rate?**

I calculated a drift rate for each log as it was occurring using the equation Θ = ((t2 - t3) - (t0 - t1)) / 2. I then summed up the total of my data then divided the sum by the total number of pings.

2. **what timeout did you pick to detect a failed interaction, and what happens if the server's response packet arrives after that timeout?**

I picked a timeout of 3 seconds. I chose this number after doing some preliminary examining of the data and concluding that it would be unlikely that a packet should arrive outside of this

generous time period (given that most packets were coming in between 0 – 0.1 seconds). If the server should respond after the timeout period I do not believe the packet would be picked up. Looking back with a count of 34 timeouts for my data, perhaps I should have no way of knowing weather or not the timeouts were due to a lost packet or the timeout being too short.

3. **for each successful interaction, calculate the clients' average clock drift rate during the period since the previous successful interaction. Plot a histogram of these "instantaneous" clock drift rate values, and show that plot in your write up. Hypothesize why the histogram is shaped as it is!!!**

The histogram is above under 'Data'. This histogram indicates a large portion of the offset being between 0 and 0.1. There is still a large portion of data that is higher then this and some outliers that lie in the 1.4 to 1.5 region. Though the shape of this histogram is unexpected, one possible reason could be due to the server's clock resetting, or even my own time resetting during this period.