CS 352 Fall 2023

Network Time Protocol Assignment

In this assignment, you will develop a Python3 client that uses the Network Time Protocol (NTP) to compute the current time from a server.

You can work either alone or with 1 other CS 352 student.

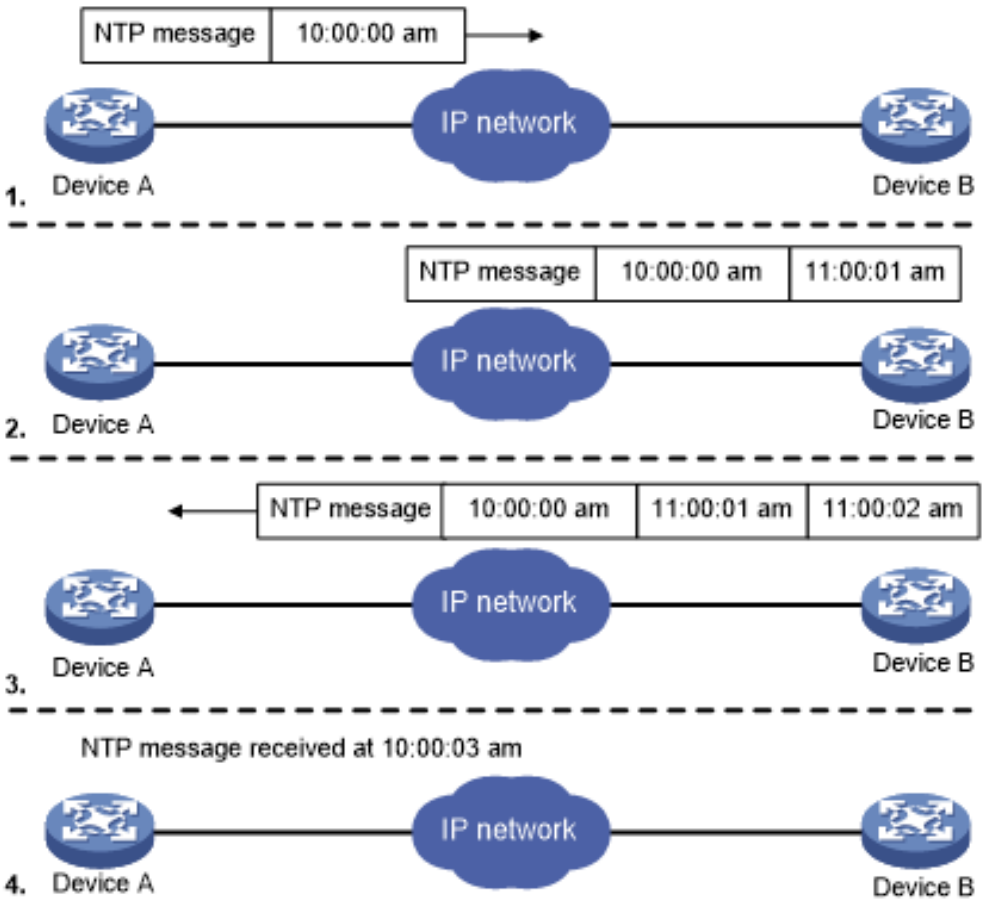
We will use Gradescope's programming project handin in order to both hand-in and grade the project.

1. **Background on the Network Time Protocol**

The Network TIme Protocol is a protocol to allow computers to compute the current time over the Internet. NTP clients can then set their local clocks, or other functions needing time if their local A description of the protocol is at the link below. It is included here to make the document self-contained. See:
https://techhub.hpe.com/eginfolib/networking/docs/switches/5820x-5800/5998-7395r_nmm_cg/content/441755722.htm

The figure below shows the messaging protocol of NTP. Device A and Device B are connected over a network. They have their own independent system clocks, which need to be automatically synchronized through NTP. Assume that:

- Prior to system clock synchronization between Device A and Device B, the clock of Device A is set to 10:00:00 am while that of Device B is set to 11:00:00 am.
- Device B is used as the NTP time server, so Device A synchronizes to Device B.
- It takes 1 second for an NTP message to travel from one device to the other.



The time synchronization process is as follows:

- Device A sends Device B an NTP message, which is timestamped when it leaves Device A. The timestamp is 10:00:00 am (T1).
- When this NTP message arrives at Device B, it is timestamped by Device B. The timestamp is 11:00:01 am (T2).
- When the NTP message leaves Device B, Device B timestamps it. The timestamp is 11:00:02 am (T3).
- When Device A receives the NTP message, the local time of Device A is 10:00:03 am (T4).

Up to now, Device A can calculate the following parameters based on the timestamps:

- The roundtrip delay of NTP message: Delay = (T4–T1) – (T3-T2) = 2 seconds.
- Time difference (the offset) between Device A and Device B: Offset = ((T2-T1) + (T3-T4))/2 = 1 hour.

Based on these parameters, Device A can synchronize its own clock to the clock of Device B. For more information, see RFC 1305.

1.2 What is Unix time?

The Unix operating system and its derivatives, such as Linux, represent time as the number of seconds since Jan. 1, 1970. This assignment will represent Unix time as a Python floating point number. However, both NTP and C use a fixed-point representation of Unix time, which is a fixed point number where the first 32 bits are an integer of the number of seconds, and the second 32 bit integer is the fractions of a second.

1.3 How to get the local computer's current time

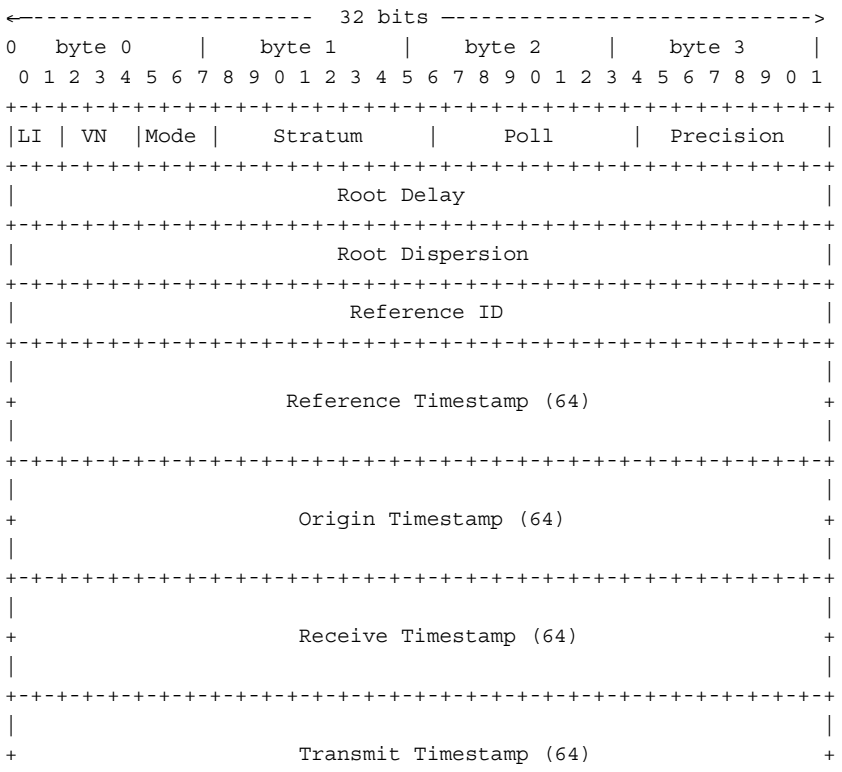
Here is some example Python code to get the the number of seconds since 1970-01-01 from the local clock:

```
from datetime import datetime

time_difference = datetime.utcnow() - datetime(1970, 1, 1, 0, 0, 0)
secs = time_difference.days*24.0*60.0*60.0 + time_difference.seconds
timestamp_float  = secs + float(time_difference.microseconds / 1000000.0)
print("The number of seconds since Jan. 1, 1970 is: %f" % (timestamp_float))
```

1.4 NTP Request Packet format

The format from the NTP client to the server for a time synchronization packet is defined in RFC 5905: https://www.rfc-editor.org/rfc/rfc5905.html



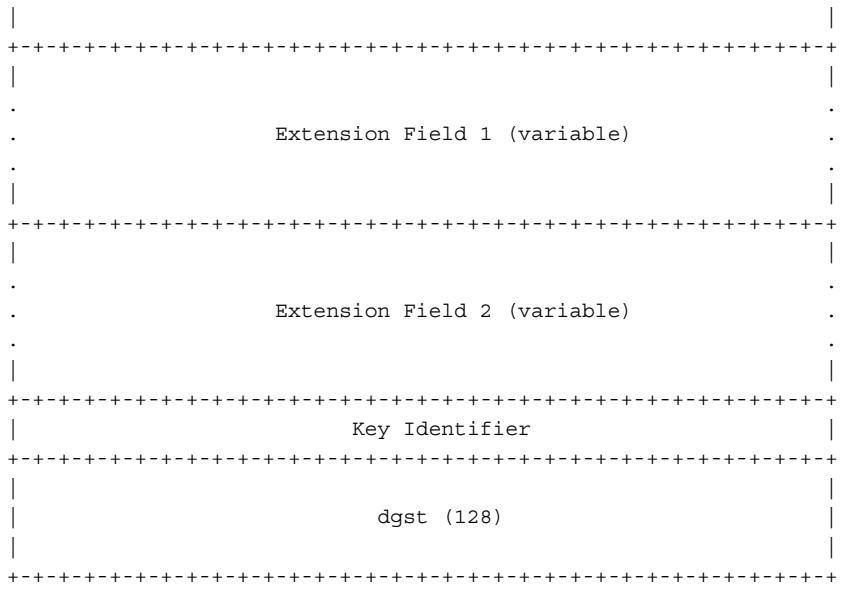


Figure: Packet Header Format

1.5 Packet Fields:

Note: Set the first byte's bits to binary 00,011,011 for LI = 0, VN = 3, and Mode = 3.

- **LI (Leap Indicator)**—A 2-bit leap indicator. When set to 11, it warns of an alarm condition (clock unsynchronized); when set to any other value, it is not to be processed by NTP.
- **VN (Version Number)**—A 3-bit version number that indicates the version of NTP. The latest version is version 4.
- **Mode**—A 3-bit code that indicates the work mode of NTP. This field can be set to these values:
    - **0**—reserved
    - **1**—symmetric active
    - **2**—symmetric passive
    - **3**—client
    - **4**—server
    - **5**—broadcast or multicast
    - **6**—NTP control message
    - **7**—reserved for private use.
- **Stratum**—An 8-bit integer that indicates the stratum level of the local clock, with the value ranging from 1 to 16. Clock precision decreases from stratum 1 through stratum 16. A stratum 1 clock has the highest precision, and a stratum 16 clock is not synchronized and cannot be used as a reference clock.
- **Poll**—An 8-bit signed integer that indicates the maximum interval between successive messages, which is called the poll interval.
- **Precision**—An 8-bit signed integer that indicates the precision of the local clock.

- **Root Delay**—Roundtrip delay to the primary reference source.
- **Root Dispersion**—The maximum error of the local clock relative to the primary reference source.
- **Reference Identifier**—Identifier of the particular reference source.
- **Reference Timestamp**—The local time at which the local clock was last set or corrected.
- **Originate Timestamp**—The local time at which the request departed from the client for the service host.
- **Receive Timestamp**—The local time at which the request arrived at the service host.
- **Transmit Timestamp**—The local time at which the reply departed from the service host for the client.
- **Authenticator**—Authentication information.

1.6 Servers you can use:

These are time servers that provide good throughput for many consecutive requests:

time.cloudflare.com
time.facebook.com
time.apple.com
clock.nyc.he.net

1.7 Packing and unpacking packets

There are two strategies for getting the integers out of the packet. Recall the packet is a byte array (or an immutable byte sequence), and certain parts must be interpreted as 32 or 64 bit integers. To “pack” a byte array (or packet) means to fill in the bytes corresponding to the fields in the packet. To unpack a packet means to extract the field values from the byte array,

One approach is the “C-style” where your code pulls out each byte of the integer and constructs an int from the four bytes. The most significant byte would be shifted left by 24, then the next byte would be shifted by 16, etc.

The second approach is to use the `struct` library in Python. The struct library uses format strings, in the style of C structs,to pack or unpack the data into a byte array. See:

https://docs.python.org/3/library/struct.html

For example:

```
#!/usr/bin/python3

import struct

# this format is 4 8-bit characters (bytes), 2 integers and a 64-bit long long integer
#  This is the example C structure
#  struct packet {
#       char b0;   // a single byte
```

```
#       char b1;
#       char b2;
#       char b3;
#       int  i0;   // 32 bit integer
#       int i1;
#       long long q0;   // 64 bit integer
# };

# this is the format string which describes the data types and their order
# ! = network byte order, c = char, i = int, q = long long int (64-bit)
format_string = '!cccciiq'
# get the total number of bytes in this format
bytesInFormat = struct.calcsize(format_string)
# create a byte sequence with the format and elements of the format
new_packet = struct.pack(format_string,b'3',b'a',b'4',b'6', 7841,89154,9897765654)
print(new_packet)
# given a byte sequence, extract the values given the interpretation of the data types in the
sequence
# python returns a tuple containing the types
(c0,c1,c2,c3,i0,i1,q0) = struct.unpack(format_string,new_packet)
print ("got data " ,c0,c1,c2,c3,i0,i1,q0)
# create a mutable array from the packed bytes:
modified_packet = bytearray(new_packet)
modified_packet[0] = 6
modified_packet[1] = 8

# change the first 32-bit integer to a 1
modified_packet[4] = 0
modified_packet[5] = 0
modified_packet[6] = 0
modified_packet[7] = 1

# print the bytearray that has been modified
print("the modifed packet is", modified_packet)
(c0,c1,c2,c3,i0,i1,q0) = struct.unpack(format_string,modified_packet)
print ("Modified data " ,c0,c1,c2,c3,i0,i1,q0)
```

Create a format string that describes a sequence of 3 bytes, followed by 2 32-bit integers:

```
>>> from struct import *
>>> # fs is the format string
>>> # the format string has a list of types
>>> #!= network byte order , '3c'=3 characters, '2I'=2 integers
>>> fs = '!3c2I' # 3 characters and 2 integers in network byte order
>>> pack(fs,b'a',b'b',b'c',63,654)
       b'abc\x00\x00\x00?\x00\x00\x02\x8e'
 >>> unpack(fs,b'abc\x00\x00\x00?\x00\x00\x02\x8e')
       (b'a', b'b', b'c', 63, 654)
```

2. Functions that must be defined and their pseudocode

Your code must be in a single file called `ntpclient,py`. It must have the following 3 functions, which are defined as below:

(1) `ntpPktToRTTandOffset(pkt,T1,T4)`
(2) `getNTPTimeValue(server, port)`
(3) `getCurrentTime(server,port,iters=20)`

Your client will be imported into a tester program, also written in Python3. The tester program will call the 3 assignment functions with different inputs and check the outputs.

Your code can not use the `ntplib` Python library. If the tester program finds that this library, or any of its functions, are imported, your client will be marked as failing all tests. Your code must use sockets to communicate with a remote NTP server.

2.1 Basic Packet Parsing

This function takes an completed NTP data packet, as Python bytes, and input Unix timestamps, as floating point numbers, and returns the Round Trip Time (RTT) and offset as a Python two-tuple, with the first element being the round-trip time and the second element being the offset. Both are floating point numbers in seconds.

Test 1: (40 points)

This test will create an NTP packet with the time values filled in, and two timestamps from the local client, and call your function, which will return the RTT and offset. This test sees if your code correctly parses an NTP packet.

```
def ntpPktToRTTandOffset(pkt,T1,T4):
    # foreach of the 2 timestamps (T2,T3) in the packet do:
        # get the bytes for the seconds part and convert to a
        # floating point number
        # get the bytes for the fraction part and convert to
        # a floating point number
        # combine the seconds and fraction into 1 number
    # compute the RTT by:  (T4-T1) - (T3-T4)
    # compute the offset by: ((T2-T1) + (T3-T4))/2
    # return a 2-tuple containing the RTT and offset as Python
floats
    # return (RTT, offset)
```

2.2 Communication and Setting the Time

This function tests if you can communicate to a remote NTP server, as well as get the local time needed to set the clock. This test will call your function and make sure the time values in the packet are close to ones sent by the tester code. Return T1 and T2 as Unix time using Python floating point numbers.

Test 2: (40 points)

```
def getNTPTimeValue(server, port):
    # make an NTP packet
    # take a timestamp, T1 = current_time
    # send packet to the server,port address
    # receive the response packet
    # take a timestamp, T4 = current_time
    # return a 3-tuple:
    # return (pkt, T1, T4)
```

2.3 Setting the clock

This function combines the previous 2; communicating with the remote server, parsing a packet, and computing the current time. The function must return the current time, computed with an average of offset values.

Test 3: (20 points)

```
# Computing the current time in Unix time format (seconds with
microsecond fractions since 00:00:00 UTC on 1 January 1970)
def getCurrentTime(server,port,iters=20):
    # offsets = empty list
    for _ in range(iters):
        # call (pkt,T1,T4) = getNTPTimeValue(server, port)
        # call (RTT,offset) = ntpPktToRTTandOffsett(pkt,T1,T4)
        # append offset to offsets
    # currentTime = average of offsets + current time with
     # microsecond granularity
    # return currentTime in Unix time as a Python float
```

3. Stub code for the client:

```
#!/usr/bin/env python

'''
CS352 Assignment 1: Network Time Protocol
You can work with 1 other CS352 student

DO NOT CHANGE ANY OF THE FUNCTION SIGNATURES BELOW
'''

from socket import socket, AF_INET, SOCK_DGRAM
import struct
from datetime import datetime


def getNTPTimeValue(server="time.apple.com", port=123) -> (bytes, float,
float):
      # fill in your code here
      return (pkt, T1, T4)
```

```
def ntpPktToRTTandOffset(pkt: bytes, T1: float, T4: float) -> (float, float):
       # fill in your code here
       return (rtt, offset)

def getCurrentTime(server="time.apple.com", port=123, iters=20) -> float):
       # fill in your code here
       return currentTime

if __name__ == "__main__":
       print(getCurrentTime())
```

4. How to handin your client

Your client must be a single file named ntpclient.py

You need to upload your client in Gradescope. Log into canvas and use the Gradescope tool on the left. The assignment is called “NTP Client”

Upload a Python file called “ntpclient.py”

How to add group members in gradescope:
https://help.gradescope.com/article/m5qz2xsnjy-student-add-group-members