**CSCI/ECEN 5673: Distributed Systems**
**Spring 2022**

# Programming Assignment Three

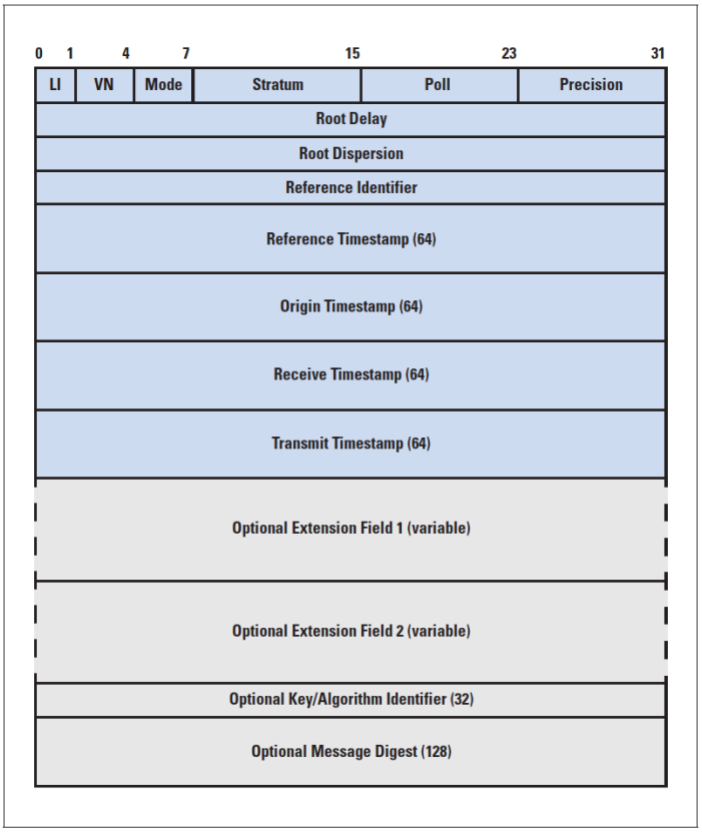
*Due Date and Time: 11:59 PM, Friday, March 11, 2022*

The goal of this assignment is to implement a scaled-down version of the NTP protocol and experiment with time synchronization accuracy.

You may work in teams of size two students

**NTP: A Brief Overview** (*https://labs.apnic.net/?p=462*)

As discussed in class, the NTP protocol is basically a UDP based clock request transaction, where a client requests the current time from a server, passing its own time with the request. The server adds its time to the data packet and passes the packet back to the client. When the client receives the packet, the client can derive two essential pieces of information: an *offset* between its local clock and the server time and *roundtrip delay* between the client and the server. Repeated iterations of this procedure allow the local client to remove the effects of network jitter and thereby gain a stable value for the offset, which can then be used to adjust the local clock so that it is synchronized with the server. Further iterations of this protocol exchange can allow the local client to continuously correct the local clock to address local clock skew.




**Packet Format**

NTP packet format is shown below:

*LI*: Leap Indicator (2 bits): This field indicates whether the last minute of the current day is to have a leap second applied. The field values are: 0: No leap second adjustment; 1: Last minute of the day has 61 seconds; 2: Last minute of the day has 59 seconds 3: Clock is unsynchronized

*VN*: Version Number (3 bits) (current version is 4)

*Mode*: NTP packet mode (3 bits): The values of the Mode field are as follows: 0: Reserved; 1: Symmetric active; 2: Symmetric passive; 3: Client; 4: Server; 5: Broadcast; 6: NTP control message; 7: Reserved for private use

*Stratum*: Stratum level of the time source (8 bits): The values of the Stratum field are as follows: 0: Unspecified or invalid; 1: Primary server; 2-15: Secondary server; 16: Unsynchronized; 17-255: Reserved

*Poll*: Poll interval (8-bit signed integer): value of the maximum interval between successive NTP messages, in seconds.

*Precision*: Clock precision (8-bit signed integer): The precision of the system clock, in $\log_2$ seconds.

*Root delay*: The total round-trip delay from the server to the primary reference sourced. The value is a 32-bit signed fixed-point number in units of seconds, with the fraction point between bits 15 and 16. This field is significant only in server messages.

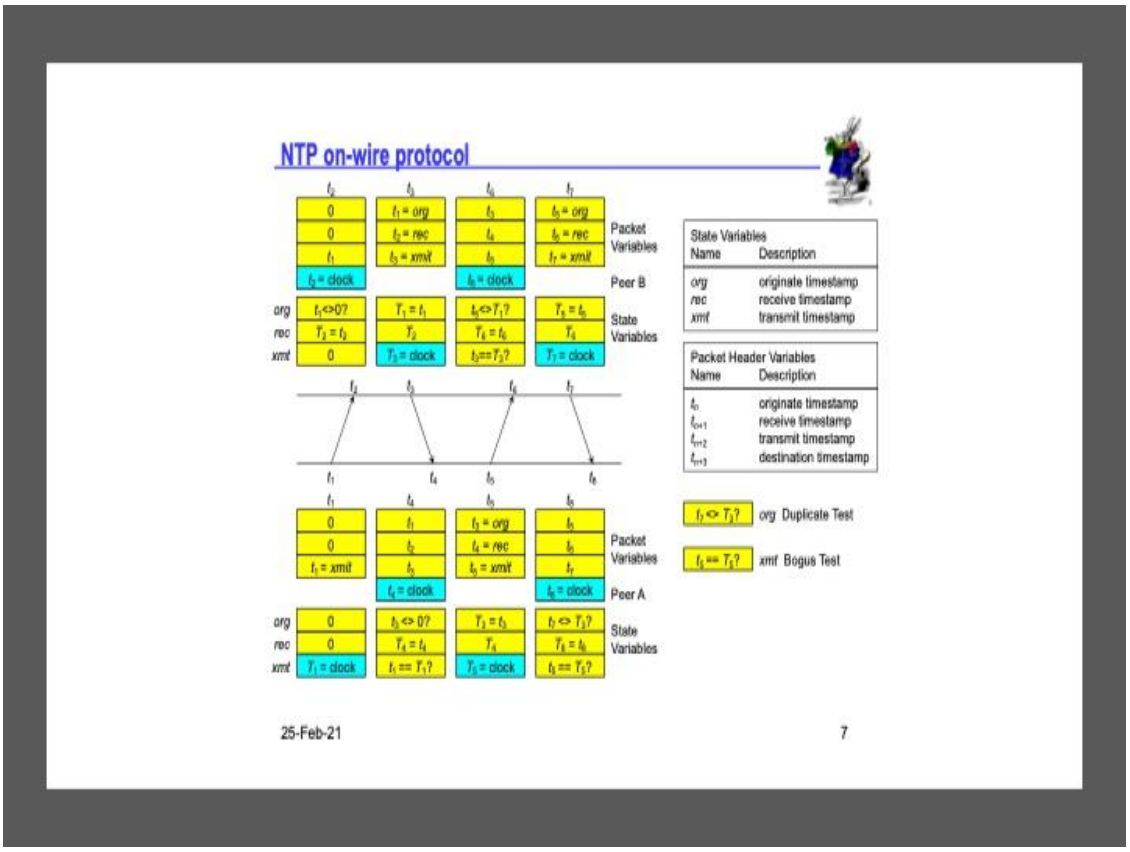
*Root dispersion*: The maximum error due to clock frequency tolerance. The value is a 32-bit signed fixed-point number in units of seconds, with the fraction point between bits 15 and 16. This field is significant only in server messages.

*Reference identifier*: For stratum 1 servers this value is a four-character ASCII code that describes the external reference source. For secondary servers this value is the 32-bit IPv4 address of the synchronization source, or the first 32 bits of the *Message Digest Algorithm 5* (MD5) hash of the IPv6 address of the synchronization source. This field is significant only in server messages.

The next four fields use a 64-bit time-stamp value. The unit of time is in seconds, and the epoch is 1 January 1900, meaning that the NTP time will cycle in the year 2036 (two years before the 32-bit Unix time cycle event in 2038). This value is an unsigned 32-bit seconds value and a 32-bit fractional part. The smallest time fraction that can be represented in this format is 232 picoseconds. For example, in this notation the value 2.5 would be represented by the 64-bit string:

*0000|0000|0000|0000|0000|0000|0000|0010 . |1000|0000|0000|0000|0000|0000|0000|0000*
*Integer Part | Decimal Fractional Part*

*Reference timestamp*: This field is the time the system clock was last set or corrected



# CSCI/ECEN 5673: Distributed Systems
# Spring 2022

The definitions of *Originate timestamp (org)*, *Receive timestamp (rcv)* and *Transmit timestamp (xmt)* are as follows (See the figure above (*https://www.eecis.udel.edu/~mills/ntp.html*)):

- When a client sends a request message: *org*: the (server) time at which the last response message received from the server was sent (0 if this is the first request message); *rcv*: the (local) time at which the last response message from the server was received (0 if this is the first request message); *xmt*: the (local) time at which this current request message is sent.

- When a server sends a response message: *org*: the (client) time at which the last request message received from the client was sent; *rcv*: the (local) time at which the last request message from the client was received; *xmt*: the (local) time at which this current response message is sent.

The optional *Key* and *Message Digest* fields allow a client and a server to share a secret 128-bit key and use this shared secret to generate a 128-bit MD5 hash of the key and the NTP message fields. This construct allows a client to detect attempts to inject false responses from a man-in the-middle attack.

**Protocol Operation**

The basic operation of the protocol is as follows: When a client receives a response from the server, it records four times: T1 is *Origin Timestamp* field of the response message; T2 is the *Receive Timestamp* field of the response message; T3 is the *Transmit Timestamp* field of the response message; and T4 is the time this response message arrived. The client then estimates the delay ($d_i$) and offset ($o_i$) between the client and the server. The minimum of the last eight delay measurements is $\delta_0$, and the offset corresponding to this minimum delay is $\theta_0$. The values ($\theta_0$, $\delta_0$) become the NTP update value.

$d_i = (T4 - T1) - (T3 - T2)$
$o_i = ½ [(T2 - T1) + (T3 - T4)]$

The final part of this overview of the protocol operation is the polling frequency algorithm. An NTP client will send a message at regular intervals to an NTP server. This regular interval is commonly set to be 16 seconds. If the server is unreachable, NTP will back off from this polling rate, doubling the back-off time at each unsuccessful poll attempt to a minimum poll rate of 1 poll attempt every 36 hours. When NTP is attempting to resynchronize with a server, it will increase its polling frequency and send a burst of eight packets spaced at 2-second intervals. When the client clock is operating within a sufficient small offset from the server clock, NTP lengthens the polling interval and sends the eight-packet burst every 4 to 8 minutes (or 256 to 512 seconds).

NTP operates over the *User Datagram Protocol* (UDP). An NTP server listens for client NTP packets on port 123. Upon receipt of a client NTP packet, the receiver time-stamps receipt of the packet as soon as possible within the packet assembly logic of the server. The packet is then passed to the NTP server process. This process interchanges the IP Header Address and Port fields in the packet, overwrites various fields in the NTP packet with local clock values, timestamps the egress of the packet, recalculates the checksum, and sends the packet back to the client.

**Requirements of programming assignment three**

Implement scaled-down versions of NTP client and NTP server. For your NTP client implementation, you do not need to implement polling frequency algorithm or Key/Message Digest. For your NTP server, you do not need to implement Root delay, Root dispersion, Reference identifier or Key/Message Digest. **Do not alter the packet format even though you are not using some of the fields** – Your NTP client must be able to interoperate with a public NTP timeserver.

# CSCI/ECEN 5673: Distributed Systems
# Spring 2022

Note that when you run your own NTP server, you won't be able to use port 123 as it is a reserved port number. For this assignment, choose a random port number (above 1023) for your own NTP server and configure your client with this port number. However, you will still use port number 123 when connecting your NTP client to a public NTP timeserver.

Further, note that NTP uses UDP as the underlying transport protocol. In UDP, a message may be lost, indefinitely delayed or delivered more than once. Your NTP client must handle these error conditions. To handle message loss in your client, use a positive-acknowledgement technique where the ack message is the response message from the server; make sure to update the *Transmit timestamp* field when retransmitting a request message. You may use system calls such as *epoll*( ), *poll*( ) or *select*( ) to implement timeouts. To check for duplicates/delayed messages, inspect various timestamp fields in the received NTP packet.

Evaluation

Measure delays, offsets and update values ($\theta_0$, $\delta_0$) over a one-hour period (eight-packet burst every 4 minutes) for the following scenarios:

1. Client and server are on different machines on the same LAN.
2. Client runs on your laptop and the server on cloud.
3. Client runs on your laptop and the server is an NTP public timeserver.

Plot all your measurements in a graph (x-axis: <burst #, message pair #>; y-axis: $o_i$, $d_i$, $\theta_0$, $\delta_0$) for each scenario. Provide an analysis of your results in terms of variations between different message pairs within a burst, between different bursts, and between different scenarios. Based on your data, can you justify the statement: *The shorter and more symmetric the round-trip time is, the more accurate the estimate of the current time will be.*

What to submit

Submit a single zip file via the submission link provided on Canvas. Your file must contain the following:

- All source code files including a makefile
- A README file that includes a description of how to compile and run your program. In addition, include any limitations of your program – what works, what doesn't, sources of potential errors, etc.
- A PDF file that contains all your graphs and their explanation.
- Raw measurement data (T1, T2, T3, T4) for each scenario in separate files in some reasonable format, e.g. Excel (clearly identifying burst # and message pair #).

Resources

- http://www.ntp.org
- Sample implementations of NTP client are available on the Internet, e.g. https://lettier.github.io/posts/2016-04-26-lets-make-a-ntp-client-in-c.html and https://medium.com/learning-the-go-programming-language/lets-make-an-ntp-client-in-go-287c4b9a969f. Feel free to look at them but please write your own code for this assignment. Also, note that I haven't verified whether these sample implementations are correct or not.