CMSC 417 Computer Networks **Spring 2017**

Programming Assignment 1

Assigned: February 3 *Due: February 10, 11:59:59 PM.*

1 Description

In this assignment, you will write a UDP client and server to run a simplified version of NTP (Network Time Protocol). In contrast to TCP, UDP provides fewer properties and guarantees on top of IP. As in TCP, UDP supports 2^{16} ports that serve as communication endpoints on a given host (which is identified by the IP address). Unlike TCP, UDP is a connection-less protocol, meaning that a source can send data to a destination without first participating in a "handshaking" protocol. Additionally, UDP does not handle streams of data, but rather individual messages which are termed "datagrams". Finally, UDP does *not* provide the following guarantees: 1) that datagrams will be delivered, 2) that the datagrams will be delivered in order, 3) that the datagrams will be delivered without duplicates.

You can find information on the UDP socket API calls in the Donahoo/Calvert book. For more information on UDP, you can refer to Section 5.1 and the introduction to Section 5 in the Peterson/Davie book.

2 Protocol

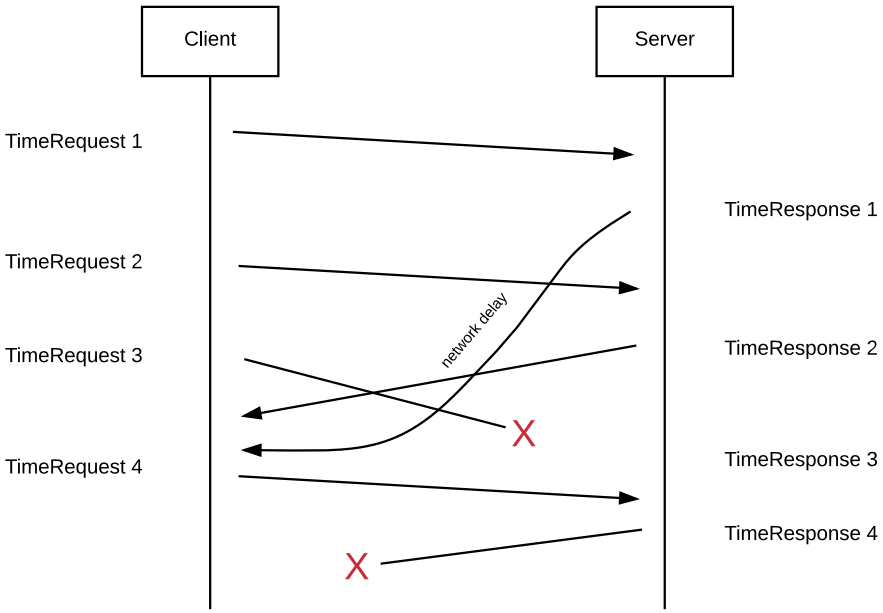

Figure 1: Example message sequence diagram for the assignment protocol. There are two dropped packets, and an out of order response.

The protocol you will implement resembles that of NTP and will use two types of messages: TimeRequests and TimeResponses. In this project the contents of the packets will include the time on the machine when the packet was sent. You will use `clock_gettime()` to get the current time; that function writes the time into a `timespec` structure containing two 64-bit unsigned integers (the number of seconds and nanoseconds since the start of the epoch). These values will be referred to hereon as the time in seconds and the time in nanoseconds. Unlike the protocol in Assignment 0, there is no initialization. The client sends a TimeRequest that contains a sequence number for the request and a timestamp of when it sent the payload. Upon receiving the TimeRequest, the server will reply with a TimeResponse that contains the sequence number it received, the timestamp that it received with client's payload, and the timestamp of when it sends the TimeResponse. The formats of these messages are shown below.

TimeRequest (Client → Server)

1. ID: A two-byte integer in network byte order that is set to the value 0x417.
2. Sequence Number: A four-byte integer in network byte order that identifies the ordering of requests sent from the client.
3. Client Seconds: An eight-byte integer in network byte order representing the time in seconds when the client sent the TimeRequest.
4. Client Nanoseconds: An eight-byte integer in network byte order representing the time in nanoseconds when the client sent the TimeRequest.

TimeResponse (Server → Client)

1. ID: A two-byte integer in network byte order that is set to the value 0x417.
2. Sequence Number: A four-byte integer in network byte order that should be identical to the sequence number sent from the client's TimeRequest.
3. Client Seconds: An eight-byte integer in network byte order representing the time in seconds when the client sent the TimeRequest.
4. Client Nanoseconds: An eight-byte integer in network byte order representing the time in nanoseconds when the client sent the TimeRequest.
5. Server Seconds: An eight-byte integer in network byte order representing the time in seconds when the server sent the TimeResponse.
6. Server Nanoseconds: An eight-byte integer in network byte order representing the time in nanoseconds when the server sent the TimeResponse.

3 Server Implementation

The server will be a command line utility, which takes the following arguments:

1. **-p <Number>** = Port that the server will bind to and listen on. Represented as a base-10 integer. Must be specified, with a value > 1024.

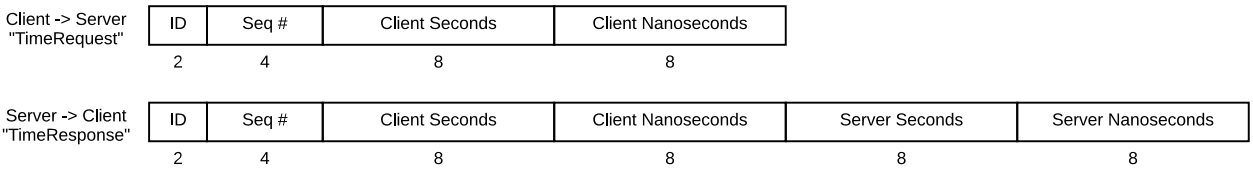

Figure 2: Example message sequence diagram for the assignment protocol. There are two dropped packets, and an out of order response.

2. **-d <Number>** = Percentage chance that the server will drop any given UDP payload that it receives. Represented as a base-10 integer. This parameter is optional, but if included, must have a value in [0, 100]. (A 0 value means that the server will not purposefully ignore any packets, and a value of 100 means that the server will ignore all packets.) Omitting this flag means that the server will not drop any packets.

An example usage is as follows:

./server -p 41717 -d 15

The server will bind to the UDP port specified by the command line arguments and receive incoming TimeRequests. Upon receiving a payload, the server will do the following:

1. Randomly keep or ignore the payload. The probability of dropping the payload is specified in the command line arguments.

2. If the payload is not ignored, take a timestamp using `clock_gettime`.

3. If the current sequence number of the payload is lower than the highest observed sequence number from the current client, print out highest observed sequence number and the current sequence number. The format of the print statement should be the address and port of the client, separated by a colon, followed by the current sequence number and the highest observed sequence number for the client, all separated by spaces.

   <ADDR>:<Port> <SEQ> <MAX>

   If the maximum sequence number for a given client has not changed for 10 minutes, the server should clear its cache for that client, removing the recorded sequence number.

4. Craft and send the TimeResponse payload back to the sender of the TimeRequest. The sequence number and client timestamp should be the same as the corresponding TimeRequest. The server time should be using the time stamp that the server just took.

4 Client Implementation

The client will be a command line utility, which takes the following arguments:

1. **-a <String>** = The IP address of the machine that the server is running on. Represented as a ASCII string (e.g., 128.8.126.63). Must be specified.

2. **-p <Number>** = The port that the server is bound listening on. Represented as a base-10 integer. Must be specified.

3. **-n <Number>** = The number of TimeRequests (N) that the client will send to the server. Represented as a base-10 integer. Must be specified, and have a value ≥ 0.

4. **-t <Number>** = The time in seconds (T) that the client will wait after sending a TimeRequest or receiving a TimeResponse before terminating. Must be specified. A value of 0 will make the client not have a timeout, meaning it will wait indefinitely for dropped TimeResponses.

An example usage is as follows:

./client -a 128.8.126.63 -p 41717 -n 100 -t 5

After parsing in the arguments, the client will have two tasks: sending all TimeRequests and reporting on the received time data.

Sending TimeRequests As specified by the command line arguments, the client will send N TimeRequest UDP payloads. For each payload the client will send, it specifically does the following:

1. Determine the current sequence number. The sequence number is simply K, where K is the Kth TimeRequest you are sending out. For instance, the first TimeRequest you send out will have sequence number 1, and so on.

2. Take the current time, using `clock_gettime`.

3. Craft and send the TimeRequest payload by including the sequence number and the timestamp.

Receiving TimeResponses The client will need to to calculate the differences of timestamps upon receiving a TimeResponse. For each TimeResponse the client receives, it should do as follows:

1. Take the current time using `clock_gettime`. We will refer to this timestamp as T_2.

2. Retrieve the original client timestamp and the timestamp generated by the server from the TimeResponse payload. We will refer to the original client timestamp as T_0 and the server timestamp as T_1.

3. Compute the time offset and round trip delay for the TimeRequest and TimeResponse change for the given sequence number. The time offset θ for the sequence is defined as

$$\theta = \frac{(T_1 - T_0) + (T_1 - T_2)}{2}$$

and the round trip delay δ is defined below.

$$\delta = T_2 - T_0$$

After computing the data, print the data for the given sequence as specified in the section below.

Reporting NTP Data The client will need to print out the time offset and round trip delay for each TimeResponse it receives.

The client should terminate immediately after receiving all TimeResponses and printing them. In the case that the client does not receive all TimeResponses within T seconds of sending all of its TimeRequests or receiving its most recent TimeResponse, the client should print out the sequence numbers of all the TimeRequests that did not get a TimeResponse. For each sequence, the print statement should be the sequence number, the time offset, and the round trip delay. The sequence number and the time offset are separated by a colon and a space, and the time offset and round trip delay are separated by a space. Each sequence's print statement should be on its own line. Below is the general format of the print statement.

<SEQ>: <THETA> <DELTA>

All of these values (except sequence number) should be printed to four decimal places. If the client did not receive any responses within T seconds of its last response (or its last TimeRequest), it should print out all of the sequences in order, using the following format:

<SEQ>: Dropped

5 Grading

Your project grade will depend on the parts of the project that you implement. Each letter grade also depends on successful completion of the parts mentioned for all lower letter grades. Assuming each part has a "good" implementation, the grades are as follows:

Grade	Parts Completed
C	Protocol completes using UDP when there are no errors
B	Handle dropped and out of order packets with 1 Client
A	Handle dropped and out of order packets with >1 Clients

6 Additional Requirements

1. Your code must be submitted as a series of commits that are pushed to the origin/master branch of your Git repository. We consider your latest commit prior to the due date/time to represent your submission.

2. The directory for your project must be called 'assignment1' and be located at the root of your Git repository.

3. You must provide a Makefile that is included along with the code that you commit. We will run 'make' inside the 'assignment1' directory, which must produce two binaries 'server' and 'client' also located in the 'assignment1' directory.

4. You must submit code that compiles in the provided VM, otherwise your assignment will not be graded.

5. Your code must be -Wall clean on gcc/g++ in the provided VM, otherwise your assignment will not be graded. Do not ask the TA for help on (or post to the forum) code that is not -Wall clean, unless getting rid of the warning is the actual problem.

6. You are not allowed to work in teams or to copy code from any source.