MPCS 54001, Winter 2021

Project 3[1]

Due: March 7 at 11:59 pm

Submit your assignment to your GitLab repo, in a folder named "project3". This work must be entirely your own. If you need help, post questions to Ed Discussion and/or visit the staff during office hours. As a reminder, if you make a public post on Ed Discussion, please don't give away the answer!

Task

In this project, you will implement a UDP-based ping client that communicates with the provided UDP-based ping server. The functionality provided by your client will emulate some of the standard ping utility functionality available in modern operating systems, except that it will use UDP rather than raw IP sockets. (There's currently not a good way to use raw IP sockets directly on departmental Linux machines.) The real ping utility is implemented with ICMP echo requests and replies over raw IP sockets.

Ping Protocol

This project's ping protocol enables a client machine to send a UDP packet of data to a remote server machine, and have the remote server machine return the data back to the client unchanged (an action referred to as *echoing*, which should sound familiar from Project 1). Among other uses, this project's ping protocol also enables hosts to determine round-trip times and packet loss to other machines.

Although UDP is being used for the transport in Project 3, the ICMP echo request and reply message formats will be used for the application message payloads (i.e., essentially ICMP-over-UDP). The ICMP request and reply message formats are defined in RFC 792. The specific message formats for Project 3 are defined in Figure 1 (Project 3's message formats have been adapted from the ICMP message formats defined in RFC 792 to include 48-bit timestamps). *The*

[1] Adopted from a Networks project written by William Conner

ping request and reply messages should be read and written in network byte order (i.e., big endian)

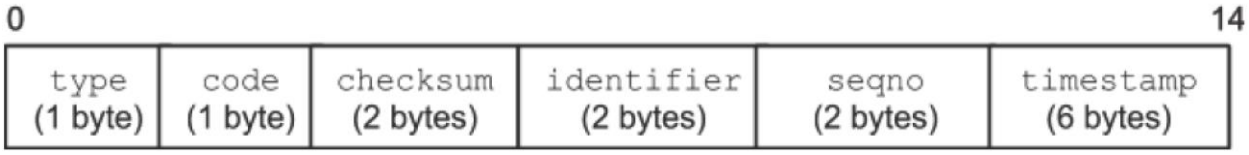

Figure 1

`type` (8 bits): 0x8 for echo request (ping), 0x0 for echo reply (pong)

`code` (8 bits): always 0x0

`checksum` (16 bits): Internet checksum computed over message

`identifier` (16 bits): arbitrary unsigned integer (must be consistent for entire duration of process, so you might want to use process ID here)

`seqno` (16 bits): sequence number starting at 1 and incremented for each successive ping message sent by the client

`timestamp` (48 bits): unsigned integer representing the number of milliseconds since the UNIX epoch

Ping Server

You can download the ping server from the assignment in Canvas. To run the ping server:

```
java -jar pingserver.jar --port=<port>
                         [--loss_rate=<rate>]
                         [--bit_error_rate=<rate>]
                         [--avg_delay=<delay>]
```

The ping server sits in an infinite loop listening for incoming UDP packets. When a packet comes in, the server validates the request and creates the corresponding reply based on field values in the request.

Packet Loss and Bit Errors

UDP provides applications with an unreliable transport service, because messages may get lost in the network due to router queue overflows, dropped due to bit errors, or other reasons. In

contrast, TCP provides applications with a reliable transport service and takes care of any lost or dropped packets by retransmitting them until they are successfully received.

Because packet loss and bit errors are rare (or even non-existent in typical campus networks), the provided ping server in this project injects artificial packet loss and bit errors to simulate the effects of an unreliable network. The ping server has a command-line parameter `loss_rate` that specifies the target percentage of packets that should be dropped and another parameter `bit_error_rate` that specifies the target percentage of ping replies that should have invalid checksums.

Delays

The server also has another command-line parameter `avg_delay` that is used to simulate average transmission delay (in milliseconds) from sending a packet across the Internet. You should set `avg_delay` to a positive value when testing your client and server on the same machine, or when machines are close by on the network. You can set `avg_delay` to 0 to find out the true round trip times of your packets.

Ping Client

Your ping client should support the following command-line options as specified below.

```
python ping_client.py <server_ip> <server_port> <count>
  <period> <timeout>
```

OR

```
python ping_client.py --server_ip=<server_ip>
                      --server_port=<server_port>
                      --count=<count>
                      --period=<period>
                      --timeout=<timeout>
```

The ping client should send `count` total ping requests to the ping server listening at IP address `server_ip` and port `server_port`. Each ping request should be separated by approximately `period` milliseconds[2]. Each ping message contains a payload of data that includes the fields defined in Figure 1 (see Ping Protocol section for message format details).

Because UDP is an unreliable protocol, some of the packets sent from the client to the server may be lost, or some of the packets sent from server to client may be lost. For this reason, the client cannot wait indefinitely for a reply to a ping request. For any given request, the client should wait up to `timeout` milliseconds to receive a reply. If a request times out without a reply from the server, then the client assumes that either its request packet or the server's reply packet has been lost in the network (i.e., packet loss event occurred). You will need to research datagram sockets in your language of choice to generally learn how to send/receive UDP datagrams and specifically to find out how to set the timeout value on a datagram socket.[3]

Whenever the client receives a reply from the server with an invalid checksum, the client should drop the reply and treat it as a packet loss.

Statistics

Your ping client should print both individual and aggregate statistics. As each individual reply is received at the client, the client should print the following individual statistics:

1. source IP address of reply UDP packet
2. ping message sequence number
3. individual elapsed time (in milliseconds)

[2] You will probably want to use a periodic task to implement this behavior. See "Periodic tasks" under the Project Resources section for references.
[3] The *Project Resources* section has some references on how to use datagram sockets.

After all replies have either been received or timed out, the client should print the following aggregate statistics:

1. total number of packets transmitted
2. total number of packets received
3. percentage of packet loss
4. cumulative elapsed time (in milliseconds)
5. minimum, average, and maximum round trip times (in milliseconds)

Here is some sample output obtained by modifying real output from the ping utility. Although not strictly required, I recommend that you make your output as close as possible to this sample output.

```
PING 128.135.164.171
PONG 128.135.164.171: seq=1 time=139 ms
PONG 128.135.164.171: seq=2 time=131 ms
PONG 128.135.164.171: seq=3 time=138 ms
PONG 128.135.164.171: seq=4 time=134 ms
PONG 128.135.164.171: seq=5 time=134 ms

--- 128.135.164.171 ping statistics ---
5 transmitted, 5 received, 0% loss, time 3999ms
rtt min/avg/max = 131/135/139 ms
```

Project Resources

- Internet checksum test vectors have been posted on Canvas
- Example test case output has also been posted on Canvas
- Datagram Sockets: documentation, `settimeout()`
- Timestamps: time.time()
- Periodic tasks: `threading.Timer`

Deliverables

Create a directory named `project3` in your individual GitLab repository. Upload your source code (client source files, README file) to that directory. You should verify that your files were

submitted correctly using the web interface at https://mit.cs.uchicago.edu. Your README should contain instructions for using your code. You should also include useful code comments for the graders.

Grading

This project will be worth 20% of your overall grade (out of the 40% for projects) for the course.

We will run your code submissions and grade them using the ping server provided for your testing.

General grading rubric for Project 3 is as follows (total 100 points):

- Starts/Runs: **5 points**
- Sends ping requests over UDP to server: **10 points**
- Receives ping replies over UDP from server: **10 points**
- Uses correct ping message format: **10 points**
- Sets correct checksum for outgoing ping requests: **5 points**
- Verifies checksum for incoming ping replies: **5 points**
    - Should print error message when checksum verification fails
    - Should discard any echo replies with invalid checksums as a packet loss
- Waits correct time interval between sending ping requests: **10 points**
- Times out correctly if reply not received within specified timeout: **10 points**
- Print individual statistics as replies are received: **10 points**
- Print aggregate packet loss statistics: **5 points**
- Print aggregate round trip time statistics: **10 points**
- Style points, grader discretion (efficiency, code correctness, documentation in README file, or other considerations e.g., you didn't just hardcode source IP address in statistics): **10 points**