CS380 — Project 5

February 13, 2017

Due: Wednesday, February 22, 2017 by midnight (60 points)

Description

In this project, you'll be implementing UDP on top of IPv4. To do this, you will create IPv4 packets as in project 3, then create UDP packets as the data for the IPv4 packets. These will then be transmitted to the server. The server will first check the IPv4 headers to ensure the IPv4 packet is constructed correctly, then check the IPv4 data to get the UDP packet and check the header and data.

Your IPv4 headers will have the same restrictions as project 3 except the protocol should be set for UDP instead of TCP. Keep in mind that the UDP checksum is calculated using the same method but different input. It includes the UDP header, data, and a "pseudoheader" that includes part of the IPv4 header.

Our programs will perform the following procedure:

(1) First, we'll do a "handshaking" step where you send a single IPv4 packet (no UDP here) with 4 bytes of data hard-coded to `0xDEADBEEF`. Then, my server will respond first with the response code for that packet (4 bytes, see below table) then with 2 bytes of raw data (not an IP packet, so you don't need to be able to parse incoming IP packets) which you should treat as an unsigned 16-bit integer corresponding to a port number.

Note that if your response code here doesn't come back with `0xCAFEBABE` don't expect the next two bytes to be sent.

(2) You will then send UDP packets (inside of IPv4 packets) with the destination port header field assigned to the number you received in the previous step. You should send packets as follows: send 12 total packets with data size starting at 2 bytes and doubling each time. The UDP source port can be set to anything you like but the data must be randomized[1].

(3) After sending each UDP packet, wait for the server response which will contain exactly 4 bytes hard-coded to `0xCAFEBABE` if your packet was constructed correctly. Print out the server response and the elapsed time in milliseconds since you sent the packet (an estimate for the RTT) for each packet transmitted. After sending all 12 packets print the mean RTT for all 12 packets.

Here's a list of potential response codes:

Code	Reason
`0xCAFEBABE`	Everything looks fine
`0xBAADF00D`	Problem with IPv4 portion of packet (see project 3 to fix!)
`0xCAFED00D`	Incorrect destination port in UDP packet
`0xDEADC0DE`	Invalid UDP checksum
`0xBBADBEEF`	Incorrect UDP data length

The host to connect to is `codebank.xyz` and the port is 38005.

[1] You can use Java's `java.util.Random` class and the `nextBytes(byte[] b)` method.

Sample Output

You must indicate the handshake response, identify each UDP packet, indicate the UDP packet responses, list the RTT for each packet, and show the final average RTT.

This is an example of how your output might look:

```
$ java UdpClient
Handshake response: 0xCAFEBABE
Port number received: 50058

Sending packet with 2 bytes of data
Response: 0xCAFEBABE
RTT: 51ms

Sending packet with 4 bytes of data
Response: 0xCAFEBABE
RTT: 43ms

Sending packet with 8 bytes of data
Response: 0xCAFEBABE
RTT: 50ms

Sending packet with 16 bytes of data
Response: 0xCAFEBABE
RTT: 41ms

Sending packet with 32 bytes of data
Response: 0xCAFEBABE
RTT: 41ms

Sending packet with 64 bytes of data
Response: 0xCAFEBABE
RTT: 40ms

Sending packet with 128 bytes of data
Response: 0xCAFEBABE
RTT: 41ms

Sending packet with 256 bytes of data
Response: 0xCAFEBABE
RTT: 51ms

Sending packet with 512 bytes of data
Response: 0xCAFEBABE
RTT: 41ms

Sending packet with 1024 bytes of data
Response: 0xCAFEBABE
RTT: 46ms

Sending packet with 2048 bytes of data
```

```
Response: 0xCAFEBABE
RTT: 44ms

Sending packet with 4096 bytes of data
Response: 0xCAFEBABE
RTT: 47ms

Average RTT: 44.67ms
```

Submission

1. On codebank.xyz, create a project as your user named `CS380-P5`. Follow this naming convention precisely including case.
2. Make sure you add the reference to the remote repository in your local project with:

   ```
   $ git remote add origin https://codebank.xyz/username/CS380-P5.git
   ```

3. Your project should have a main class named `UdpClient` in a file named `UdpClient.java`. You can have other files or classes, but it should successfully compile and run by simplying using:

   ```
   $ javac UdpClient.java
   $ java UdpClient
   ```