CS380 — Project 4

January 26, 2015

Due: Wednesday, February 4, 2015 (60 points)

Description

In this project, you'll be applying the same techniques you used in project 3 to implement IPv6. IPv6 packet headers may be larger than IPv4, but are actually quite a bit simpler! Like project 3, here's a table of things that must be done:

Version	Implement
Traffic Class	Don't implement
Flow Label	Don't implement
Payload length	Implement
Next Header	Implement and set to UDP protocol value
Hop Limit	Implement and set to 20
SourceAddr	Implement assuming it is a valid IPv4 address that has been extended to IPv6 for a device that does not use IPv6
DestinationAddr	Implement assuming it is a valid IPv4 address that has been extended to IPv6 for a device that does not use IPv6 using the IP address provided in this project specification.

Send the packets as raw bytes through the socket's output stream. The host will be the same as the previous project: 76.91.123.97 on port 38004. The server should be running by Wednesday, January 29.

For more information about the implemenation of IPv6, see your textbook, the RFC specification at `https://www.ietf.org/rfc/rfc2460.txt`, or the Wikipedia article about IPv6: `https://en.wikipedia.org/wiki/IPv6`.

Like project 3, you should generate 12 packets with data size starting at 2 bytes and doubling for each packet.

Unlike project 3, you will not be getting back text messages indicating what went wrong or if the packet was correct. Instead, I'll be sending back a specific response code for each packet you send consisting of a four byte magic number. Here's a table indicating which codes will be sent under various circumstances:

Code	Reason
`0xCAFEBABE`	Packet was correct
`0xCAFED00D`	Incorrect version
`0xDEADF00D`	One of the "do not implement" fields was not all 0s
`0xBBADBEEF`	Payload length is wrong
`0xFEEDFACE`	Next header is wrong
`0xFEE1DEAD`	Hop limit is wrong
`0xDEADC0DE`	Bad source address
`0xABADCAFE`	Bad destination address

Therefore, you should be aiming for `0xCAFEBABE` after each packet. Your program should print each response code in hexadecimal after sending the packets. I recommend *not* using a separate thread for receiving from the server for this project, it's easier to just read the four bytes after each packet that you send.

Submission

Submit a single Java file, `Ipv6Client.java` to Blackboard. You should hardcode the above host and port information, ask for the user to input it, or accept it through command line arguments. Don't package your classes. If you use more than one class, include all of them i nthe single `Ipv6Client.java` file.