CS 3251 - Computer Networks I

Spring 2017

Programming Assignment 1: Basics of Socket Programming

DUE DATE: FRIDAY FEBRUARY 10, 5PM
Please work individually – no group submissions

Introduction

In this assignment you will write your own client-server application using network sockets. The first goal of the assignment is to provide you experience with basic sockets programming. The second goal is to prepare you for the second and third programming assignments, which will be significantly more challenging.

Assignment

You will implement two versions of the same client-server application. One version will be based on TCP. The other will use UDP. You will implement both the client and server sides of the application. You can use Python, Java or C/C++. If you want, you can even use one programming language for the client, and another language for ther server.

Client/Server interaction:
The server will be given a plain-text file that contains a set of words (”suspicious words”) that often appear in SMS spam messages. We will provide you with such a file (one word per line – case insensitive).
The client reads an SMS text message from a local file and it sends that message to the server. The latter receives the string, breaks it into individual words, and checks if each of these words appear in the set of suspicious words. It then returns back to the client a “spam score”, which is a floating-point number between 0 and 1.
The server calculates the spam score as follows:
Spam-score = (number of occurrences of suspicious words in the received message) divided by (total number of words in the received message). For example the spam score of the message: `[Please send me your credit card]` is 2/6 = 0.333, assuming that the words “credit” and “card” appear in the set of suspicious words.

Starting the server: To server’s executable file should be called “smsengine” and it will be given two command-line arguments: a port number and a text file with the set of suspicious words.
Example:

```
smsengine 8591 suspicious-words.txt
```

The client: The client command should be called "smsclient". Your client's command line should include the server IP address, port number, and the name of a text file that contains a single SMS message (between 1 and 1000 characters). In the following exampl, the client's query is sent to the server running at the same local host as the client (represented by the special IP address 127.0.0.1) listening at port 8591.

Example:
```
smsclient 127.0.0.1 8591 msg.txt
```

Server's functionality and reply format: When the server receives a message from the client, it should calculate the spam score of the received SMS text message. Then the server replies back with three pieces of information:
1) The number of words in the server's set of suspicious words (unsigned integer),
2) The spam-score of the received SMS message (floating-point number),
3) The suspicious words found in the received SMS message (successive words separated by a blank space).
If the client provided any type of bad input, the server sends back the following special message:
```
0 -1 ERROR
```

TCP and UDP versions: You will implement two versions of the assignment: one using TCP and another using UDP. The TCP application should close the connection after the response is received. The UDP client should set a timeout for receiving a response. If the client does not receive a response within 2 seconds, it should retry the same query 3 times. After 3 unsuccessful requests, the client should print an error message and exit. For example, the following command would start the client, and have the following output:

```
smsclientUDP 127.0.0.1 8591 msg.txt
The server has not answered in the last two seconds.
retrying...

Response from server:
0.333 1000 credit card
```

To test this functionality try executing the UDP client before starting the server.

Note: The command line must work exactly as specified above so the TAs can easily test your program.

Submission

You will implement two separate versions of this assignment: a client and a server using TCP, and a client and a server using UDP. Your final submission should include all the source code and

instructions to produce 4 separate executables from the provided source code. Please call these executables: smsclientTCP, smsengineTCP, smsclientUDP and smsengineUDP.
Your programs are to be written in Python, Java or C.

We will test your code at the 8 network-lab machines (named "networklab1.cc.gatech.edu" through "networklab8.cc.gatech.edu"). We strongly suggest that you test your code in those machines before you submit it.

Please turn in well-documented source code, a README file, and a sample output file called Sample.txt. The README file must contain:

- Your Name and email address
- Class name, date and assignment title
- Names and descriptions of all files submitted
- Detailed instructions for compiling and running your client and server programs
- Any known bugs or limitations of your program

You must submit your program files online.

- Create a ZIP archive of your entire submission.
- Use T-Square to submit your complete submission package as an attachment.

For example, a submission may look as follows-

```
pa1.zip
  | -- pa1/
        | -- smsclientTCP.py
        | -- smsengineTCP.py
        | -- smsclientUDP.py
        | -- smsengineUDP.py
        | -- smsengineUDP.java
        | -- smsengineTCP.java
        | -- README.txt/.pdf
        | -- sample.txt
```

Grading

The grading will be divided as follows:

- 20% Documentation
- 40% TCP application
- 40% UDP application

You will get 100% of the points if:

- You can successfully exchange messages using both TCP and UDP.
- You cover correctly all error cases (including unexpected termination of the client or server, host-order/network-order transformations, etc).

- Your TCP and UDP applications behave as expected.
- Your code is well documented.
- The TAs can successfully run your code.

Important Notes:

- A timeout is necessary only in the case of UDP, not in the case of TCP
- Submit only source code -- no executables
- **Using special network libraries:** Certain network libraries can make this assignment trivial – to the point that you will not learn much if you use them. For this reason, please ask (ideally on Piazza) before you use a specific library.