CS 421 – Computer Networks
Programming Assignment II
Flood the Network
Due: May 10, 2021 at 11:59PM

1) Introduction

In this programming assignment, you are asked to implement a program in **Java or Python 3**. You should code a peer program which learns the Peer-to-Peer (P2P) overlay network topology from a text file and according to that acquired topology makes necessary connections with other peers and finally floods the network with specific messages. Peer program must communicate with the other peers (same peer that you will implement) by using the protocol defined in this document.

The goal of the assignment is to make you familiar with the directory service in P2P architecture. You must implement your program using the **Java Socket API of the JDK or the Socket package in your default Python distribution.** If you have any doubt about what to use or not to use, please contact your teaching assistant.

When preparing your project please keep in mind that your project will be auto-graded by a computer program. Problems in formatting may also create problems in the grading; although you will not get a zero from your project, you may need to have an appointment with your teaching assistant for manual grading. Errors caused by incorrect naming of the project files and folder structure will cause you to lose points.

2) Specifications

In this assignment, you will use TCP for communication between the peers. Your first task is to construct the necessary overlay network after acquiring connectivity information from the text file and to get ready for flooding the network. In this process, your peers should be run from different ports to differentiate from each other. The topology can be subject to change (keep in mind that while grading different topologies can be utilized.). Your peers should first read the text file that demands a specific topology. Topology text file provides the necessary connectivity information of each peer on a separate line, an example is given below. The first line of the file corresponds to the number of peers and each remaining line corresponds to a link in the overlay topology.

```
4\r\n
1->2\r\n
2->3\r\n
3->4\r\n
4->1\r\n
```

It is the peer's duty to dissect the information needed for itself. For example, let us say the overlay network topology is a simple loop such that 1->2->3->4->1 where each number denotes the ID of a peer. The same topology is available to all peers. For link i->j, peer i must initiate the TCP connection with peer j. In the above sample topology, peer 1 must only initiate connection to peer 2 and peer 1 waits for a connection request from peer 4. Each peer is waiting for TCP connections at port (60000+peer ID). Note that depending on the network topology, a peer might be expected to accept TCP connections from multiple peers. After establishing TCP connections, peers must authenticate themselves. After the authentication step, all peers must **wait till the system time hits the minute mark** to start flooding periodic messages. So, let us say all 4 peers constructed the necessary topology at **20:39:45.** Then, each peer is required to send messages bearing its timestamp and peer ID to all its neighbors in the overlay network once **every 5 seconds starting at 20:40:00**. This will last for **30 seconds** (i.e., 7 rounds of messaging in total) and after that all peers are supposed to terminate their connections and shut themselves. In the implementation of the flooding mechanism, you need to make sure that

- all nodes receive each message flooded into the network and
- nodes forward each message with the same source peer ID and timestamp only once. To that end each peer is required to check whether it has already forwarded an incoming message or not.

A client-side peer, which initiates the connection, must print a message for each server-side peer it is connected to in the following format after establishing the TCP connection and passing the authentication step:

```
TCP connection established with peer X.

Authenticated to peer X.
```

Before shutdown each peer must also report its own message summary which states the number of times it has received messages for each timestamp and source peer ID during the entire run. This will be shown in the example structure below. This table should have number of peersx7 rows in total (without the headlines etc.).

```
Source Node ID   |  Timestamp  |  # of messages received
---------------------------------------------------------
                 |             |
                 |             |
                 |             |
                 |             |
                 |             |
                 |             |
```

The flowchart below can help you understand the overall process.

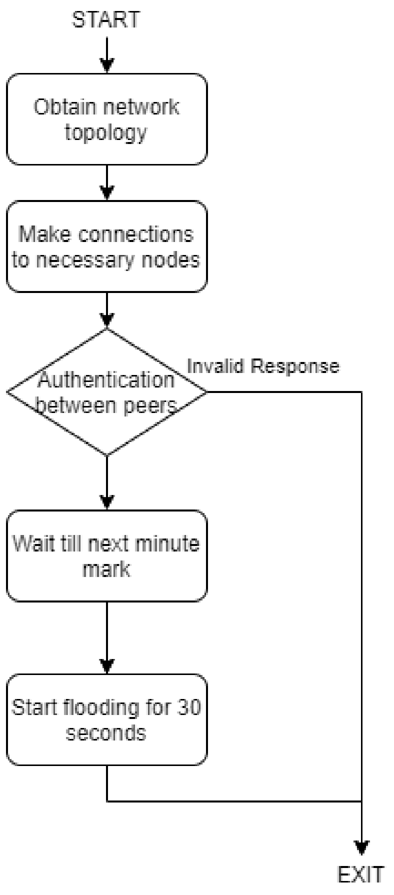

START
Obtain network topology
Make connections to necessary nodes
Authentication between peers
Invalid Response
Wait till next minute mark
Start flooding for 30 seconds
EXIT

Figure 1 Peer process flowchart

3) Connection Formats

i) Commands

You will need to use specific commands to communicate with other peer. Commands should be strings encoded in **US-ASCII**. The format is as shown below:

`<MessageType><Space><Arguments><CR><LF>`

- `<MessageType>` is the name of the command. For all possible commands and their explanations, see Figure 2.
- `<Space>` is a single space character. Can be omitted if `<Arguments>` is empty.

- `<Arguments>` is the argument specific to the command. Can be empty if no argument is needed for the given command. See Figure 2 for more information.
- `<CR><LF>` is a carriage return character followed by a line feed character, i.e., “\r\n”.

MessageText	Arguments	Message Definition	Message Example
USER	`<username>`	Sends the username for authentication	`USER bilkentstu\r\n`
PASS	`<password>`	Sends the password for authentication	`PASS cs421s2021\r\n`
FLOD	`<source peer’s ID> <Space> <timestamp>`	Floods the network with original sender and timestamp message.	`FLOD 1 12:08:55`
EXIT	-	Terminates the program	`EXIT/r/n`

Figure 2 List of commands with examples

ii) Responses

The response messages sent by the peers are also encoded in **US-ASCII**. **Responses should only be given for authentication messages.** They have the following format:

`<StatusCode><Space><StatusText><CR><LF>`

- `<StatusCode>` is either OK (success) or INVALID (failure), for the sake of simplicity.
- `<StatusText>` is the response message given by the server-side peer.
    - You should check the `<StatusText>` for the reason of the failure if `<StatusCode>` is INVALID.
- `<CR><LF>` is a carriage return character followed by a line feed character, i.e., “\r\n”.

4) Running the Peers

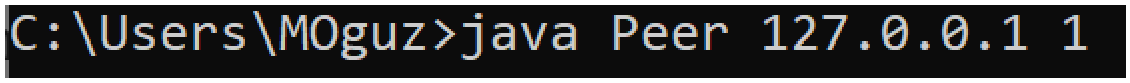
Your program must be **a console application** (no graphical user interface, GUI, is allowed) and should be named as Peer.java or Peer.py (i.e., the name of the class that includes the main method should be Peer). Your program should run with the command (or python equivalent)

```
java Peer <Addr> <peer ID>

python Peer.py <Addr> <peer ID>
```

where "< >" denotes command-line arguments.

Example:



```
C:\Users\MOguz>java Peer 127.0.0.1 1
```

In this example, the peer program (run for execution of peer 1) waits for connections to be made from IP 127.0.0.1, i.e., localhost, on port 60001(60000+peer id). **Please note that you must run the same program for each peer with the corresponding port numbers, i.e., 60000 + peer ID**. You may prefer to use a batch file to launch all peer programs at once for convenience.

5) Final Remarks

- **Please contact your teaching assistant if you have any doubt about the assignment.**
- **Do not forget to check the response message after sending each command to see if your code is working** and to debug otherwise.
- You might receive some socket exceptions if your program fails to close sockets from its previous instance. In that case, you can manually shut down those ports by waiting for them to timeout, restarting the machine, etc.
- Remember that all the commands must be constructed as strings and encoded with US-ASCII encoding.
- Use big-endian format whenever necessary.

6) Submission rules

You are required to comply with the following rules in your submission. You will lose points otherwise or if your program does not run as described in the assignment above.

- "All the files must be submitted in a single zip file whose name must start with your name and student ID. For example, the file name must be "AliVelioglu20141222" if your name and ID are "Ali Velioglu" and "20141222", respectively. If you are submitting an assignment done by two students, the file name must include the names and IDs of both group members. The file name in this case must be "AliVelioglu20141222AyseFatmaoglu20255666" if group members are "Ali Velioglu" and "Ayse Fatmaoglu" with IDs "20141222" and "20255666", respectively. The file must be a .zip file, not a .rar file, or any other compressed file."
- All the files must be in the root of the zip file; directory structures are not allowed. Please note that this also disallows organizing your code into Java packages. The archive should not contain any file other than the source code(s) with .java extension. The archive should not contain these:
    - Any class files or other executables,
    - Any third-party library archives (i.e., jar files),
    - Project files used by IDEs (e.g., JCreator, JBuilder, SunOne, Eclipse, Idea or NetBeans, etc.). You may, and are encouraged to, use these programs while developing, but the end result must be a clean, IDE-independent program.
- The standard rules for plagiarism and academic honesty apply; if in doubt refer to https://w3.bilkent.edu.tr/web/provost/SAIC_Students.pdf and http://ascu.bilkent.edu.tr/Academic_Honesty.pdf..