CS765 Project Part-I

Team Size : At Most 3 students per team

Due date: September **9**, 2020

Objective: Build a Gossip protocol over a peer-to-peer network to broadcast messages and check the liveness of connected peers. Here "liveness" means that the peer is up and running, and is able to respond to messages sent to it.
Overview: Each peer must connect to at least $\lfloor (n/2) \rfloor + 1$ seeds out of n available seeds. Note that n will vary for each instance of the peer-to-peer network and the config file should be changed accordingly. Config file contains the details (IP Address:Port) of seeds. The definitions of "seed","peer" etc. are given below. Each "peer" in the network must be connected to a randomly chosen subset of other peers. The network must be connected, that is the graph of peers should be a connected graph. In order to bootstrap the whole process, the network should have more than one *Seed* node which has information (such as IP address and Port Numbers) about other peers in the network. Any new node first connects to seed nodes to get information about other peers, and then connects to a subset of these.

Once the network is formed, the peers broadcast messages in the network and keep checking the liveness of connected peers in regular intervals. If a node is found to be dead, the details of that node needs to be sent to the seeds. This setup will be used in future assignments to form the base to build blockchain systems.

Network setup

Your network will consist of the following type of nodes:

1. *Seed Nodes*: A *Seed* node is a node that is used by a peer to get into the P2P network. There should be n (n>1) seed nodes active, this number n will be chosen arbitrarily for different instances of P2P network. A *Seed* node maintains a list of IP address and port number pairs of peers that have connected to it. This list is called *Peer List (PL)*. On startup, any new peer registers itself with at least $\lfloor (n/2) \rfloor + 1$ randomly chosen seed nodes. Registration request triggers an event at the seed node to add an entry containing the peer's IP and port number to the PL. For simplicity, in this project, seed nodes are not *peers*, that is, they do not perform actions of peers as described below. Seed Node helps in building a Peer to Peer Network by giving details about already registered Peer Nodes to new Peer Nodes.
   When a seed receives a 'dead node' message (details given below) from any peer, it should remove the dead node's details from the PL if present.

   Please note that for this to happen, the peers will not close connection formed initially with the seed nodes.

2. *Peer Nodes*: On launching the *peer* node, it first reads the IP Addresses and Port number of seeds from a config file. The IP Addresses and Port Numbers of the seeds will be hardcoded in this config file. You will be allowed to change only this config file at the time of evaluation (demo of project).

The peer then randomly chooses $\lfloor (n/2) \rfloor + 1$ seed nodes and registers its identity (IP address: port) with them. After registration with the chosen seed nodes, a peer requests the list of peers available at the seed node. In response, the seed nodes send their peer list. The peer node then takes a union of all the peer lists that it received from different seeds and randomly selects a maximum of **4 distinct** peer nodes and establishes a TCP connection with each of them.

Along with this, every node maintains a data structure called *Message List(ML)* to efficiently broadcast a gossip message so that every message travels through a link (pair of connected peers) at most once. ML consists of the hash of the message, and information about which of its connected peers it has sent the message to or received the message from. ML can also have other fields as per your needs.

Also, the peer nodes have to check for the liveness of its connected peers periodically. So a node will send a liveness request message to all the peers it is connected to every 13 seconds. If a node gets this message, they have to reply only to the sender with the liveness reply message. If 3 consecutive liveness messages are not replied to, the node will notify the seed nodes that it is connected to, that this particular IP Address is not responding. The peer will then close the TCP connection with the node that isn't alive. Upon getting the message that a particular node is not alive, the seed will remove the details of the 'dead' node from it's PL.

Removing dead nodes from a PL ensures that peers do not try to connect to nodes which are offline. However, the scheme described above to remove dead nodes is open to security attacks by malicious nodes. Try to think of how a malicious peer can launch some type of attack, and also think of a solution to the attack which you propose. You need not implement the attack or the solution, but be prepared with answers at the time of the evaluation (demo).

Gossip Message format

A peer should generate a message of the form

<self.timestamp>:<self.IP>:<self.Msg#>

Such messages will be generated by each peer node every **5** seconds. The first message should be generated by a peer after it connects to selected neighbors after registration. A peer will stop message generation after it generates 10 messages.

Gossip protocol

After a node (peer) generates a message M, it transmits M to all its adjacent nodes. On receipt of a message for the first time, a node makes an entry in the ML(<hash(M),*true*>) and forwards it to all peers except the peer from which it got the message. On receiving the same message subsequently, the node just ignores it.

Maintaining ML will prevent the nodes from forwarding the same Message more than once. This is to avoid unnecessary forwarding, and to prevent loops.

Liveness Message Format

The liveness request message format should be:

Liveness Request:<self.timestamp>:<self.IP >

The liveness reply message format should be:

Liveness Reply:<sender.timestamp >:<sender.IP >:<self.IP >

The 'self.timestamp' in liveness request must match 'sender.timestamp' in liveness reply. Similarly, 'self.IP' should match 'sender.IP' in liveness requests and replies respectively.

Liveness Testing
The liveness messages should be sent every 13 seconds, even after the gossip broadcast has stopped. The liveness messages won't stop until the node goes offline.

Reporting the node as 'Dead'
When 3 consecutive liveness requests do not receive a reply, the peer sends a message of the following format to all the seeds it is connected to:

Dead Node:<DeadNode.IP>:<self.timestamp>:<self.IP>

Program Output
Each seed writes to the screen and also to an output file the connection request from new peers. It also outputs if message is received about a 'dead' node.

Each peer writes to the screen and also to an output file the list of peers it got from seed nodes. Every gossip message received the first time should be displayed on the console as well as written to the output file along with a local timestamp and the IP address of the node from which it received the message.
The node does not need to output the liveness messages to the output file (You have the option to output these messages if you need). If the node doesn't get three consecutive liveness replies, then it must output the dead node message that it sends to the seeds in the screen as well as in the output file.

Useful Links You can follow the links below to brush up your knowledge about socket programming.

1. Beej's Guide to Network Programming.
2. Python Network Programming.

Tips for testing:

1. Before starting the actual assignment, to test socket implementation, test the transmission of messages between two nodes.
2. You may use different instances of Terminal(Linux)/Command Prompt(Windows) with same IP Address and different Port numbers to simulate different nodes. However, your code should also be able to run on different machines.

SUBMISSION INSTRUCTIONS:

1. The assignment should be done in a group consisting of a maximum of three students.
2. The code should follow programming etiquette with appropriate comments.
3. Add a **README** file which includes a description of the code and gives detailed steps to compile and run the code.

4. Name your files as **peer**.extension, **seed**.extension, **config**.csv/txt and **outputfile.txt**. Where extension is the extension of the file according to the programming language used. e.g - seed.py and peer.py if python is used.

5. Zip all your code as a single file with the name **rollno1-rollno2-rollno3.tar.gz** and upload it to Moodle. Only one group member should upload the file.

6. Please note that you won't be allowed to make any changes to the code once submitted. You can only make changes to the config file that contains the IP Addresses and Port Numbers of the seed nodes. Any changes to the code will result in 20% penalty of total marks. The penalty will increase with the amount of changes done.