Project Goal:

The goal is to design and implement a simple **application layer protocol** over UDP to facilitate High Availability Cluster (HAC). HAC has a set of mechanism to detect failovers, network/node failure etc. in order to re-route the traffic to the available systems. In this project you will not be working on the questions like how to perform consistent failover, or high availability distributed hypervisor. However, your task is to design and implement a protocol to maintain the up-node information throughout the cluster.

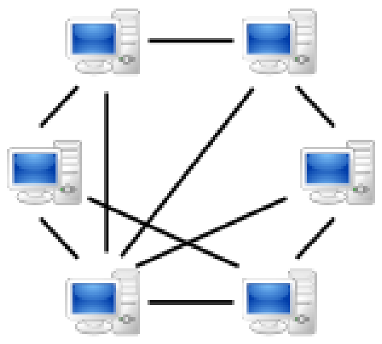
Your designed protocol should perform the following functions:

a) To detect node failure periodically (**also the Server failure in case of Client-Server mode)
b) To inform the other nodes in the network about the failure (peering option)
c) To be able to detect when the failed node comes back to life
d) To inform other nodes about the availability of new node

(Please read details about a HAC at: https://en.wikipedia.org/wiki/High-availability_cluster)

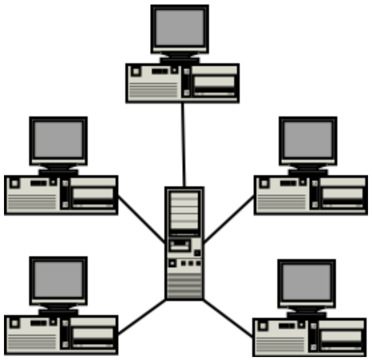
Design Requirements: Your protocol design structure must be flexible and deep enough so that the same packet format could be used in P2P (peer to peer) or Master-Slave (Client-Server) mode of HAC.

P2P:



- In this mode, each node sends its availability to all other nodes (assume that the IP addresses of all the nodes is available in the config file) in the network at random interval of time (0-30 seconds). In the output panel, the availability of all the nodes in the network is displayed in the form of a list. Assume that, each node is aware that there are ‘n-1’ other nodes in the network, whose ip addresses are available in the config file.

Client-Server:



- This design ensures that all the client nodes send updates to the server node, and the server node informs everyone else (master-slave option). Client will generate its availability packet after a random interval of time, which will be randomly selected between 0 second to 30 seconds. Server will listen to the availability of all the clients, and will generate the packet with all clients' availability and will forward it to all clients. Server considers a client dead, if it does not hear anything from any client till 30 seconds. However, a client can anytime come back and send its availability to Server. It is possible that a Server goes down, in such an event one of the Clients will assume the role of the Server, and it is known as automatic failover. Your design should consider an approach about how the automatic failover would happen. When the Server that went down comes back online, it assumes the role of a Client to start with.

HINT:

A typical protocol packet contains version, length, flags, reserved (for future) and other major important fields/sections that contain protocol specific information.

Implementation:

You would be designing the HAC protocol and implement it over UDP in P2P and Client-Server mode. It is recommended to use Java for this project. Remember that your implementation would be tested in one mode at a time i.e. either P2P or Client-Server. Thus, you should create two Java projects using Eclipse (or any other IDE), one for P2P implementation and other for Client-Server. If you choose Java, you are required to use DatagramPacket and DatagramSocket class present in Java library for UDP communication. How your project would perform in either mode is already described earlier in this document. Irrespective of your mode, the protocol structure (format) will remain the same. For testing purposes, it is recommended that you use atleast 4 computers, however the development could be done using "localhost" or loopback ipaddress (127.0.0.1). If you do not have access to these many computers you may use free AWS cloud for additional nodes.

If in any case, you need more clarification or information, do not hesitate to contact me through email or office hours.

Grading: There is no partial grading in this project. I will only grade completely finished project submissions.

Your project will be graded on:

- The depth of design of your protocol. I will use the protocols discussed in chapter 2 for comparing your protocol
- The correctness of your P2P and Client-Server implementations, and the expectations are discussed earlier in this document
- Testing your implementations based on the use-cases I have designed