DESIGN DOCUMENT

CS550 - Advanced OS

Fall 2020

Amit Nikam

A20470263

anikam@hawk.iit.edu

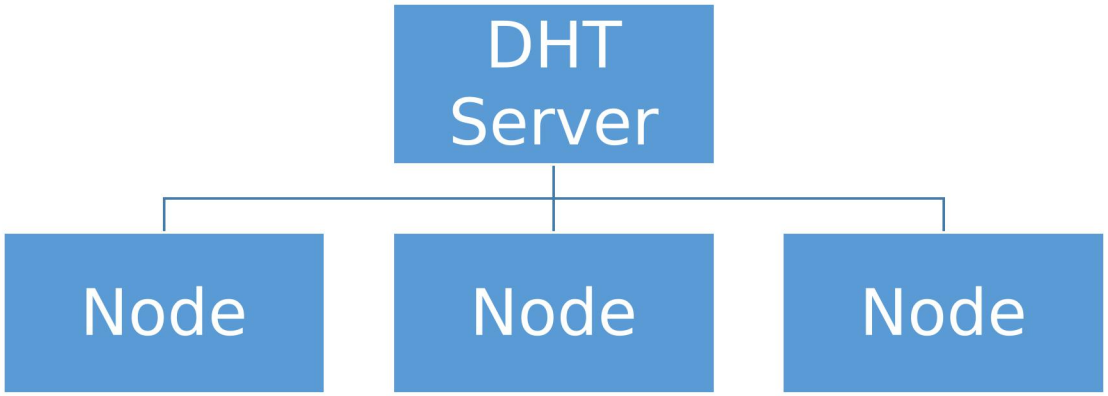
Implementation

Overview: In this assignment I have programmed a very naive P2P architecture where the server is a centralized Distributed Hash Table server which keeps a record of all the active nodes in the network and the files they have. These nodes are synchronized with the DHT Server while they are active and keep a replica of the record for this period. The DHT server and nodes are both multithreaded and can connect to multiple other nodes at the same time. Nodes are capable of uploading and downloading files in parallel. DHT Server and node make use of sockets along with a set of predefined protocol to establish connection and communicate. Sockets are software structures within a network node of a computer network that serve as an endpoint for sending and receiving data across the network. This program is implemented using python 3.8.3 and it's default modules.

Server: Being a DHT socket server, upon initialization the server binds itself to an endpoint (by default self host and port 9000). An endpoint is a port on the host machine and can be set manually by a user by passing it as an argument to the server while initializing. Upon initializing the DHT server an empty record is created which is later updated with active nodes in the network and their corresponding files.

Nodes can connect to the DHT server using the endpoint address, which is server's IP address and port number. Node needs to be aware of the server's endpoint to connect to it. Every time a node connects to the server, the server creates a new thread that specifically handles that particular connection. Through this connection, the node activates itself, updates the file record on DHT, Synchronizes with the DHT and deactivates itself. Once the connection is closed, the corresponding thread also terminates. No limit has been set to the creation of threads to experiment with the performance.

Following image shows the DHT Server to node relation:



The server actively keeps listening for any new connections and messages. The Server in particular doesn't initiate any communication but waits for the nodes to communicate. Upon receiving an activation message from the node, the server adds the node to the record. It follows a request-reply form of communication. If a client requests for a record of nodes and files, the server returns the record as a reply. If the client requests a deactivation, the server drops the node from the record and network.

Node: The node implementation happens to be a server and client at the same time. Depending upon the case, the node can become a server or be a client to other nodes. While initializing the node, the user can pass arguments to declare where to host the files for the node, set node IP address or set node port to listen for connections. If the file directory is not created, the node creates one. Each node needs to be aware of the DHT server's location.

Upon initializing, the node establishes a threaded connection to the DHT Server i.e. a threaded connection at node is connected with a threaded connection at the server end. This connection acts like **a node handler**. After successfully establishing the connection, the node sends the server a message to activate itself, update its record of files, fetch the DHT record and deactivate before shutdown. The server sends the latest record on demand. All the communications come under node handler.

The node then requests the user to provide some input for selecting the node they want to download files from. Then the node establishes connection with the other nodes as a client since it is trying to download files from those nodes. The node-client fetches the node-server's file list first and then takes user input as to which files need to be downloaded.

After getting the list of files to download from a specific node, the node-client proceeds to establish a threaded connection to download these files from the respective node. This connection works as **a download handler**. The files are downloaded serially, one after another. If any of the downloads fail, the download handler tries to download them again. After all the downloads a completed, the node updates and synchronizes with the DHT through the node handler.

The listener thread on node acts as a server and is listening on the specified port. It serves the other nodes which are looking to download files. The node listener is a request-response type thread and so it serves the clients on demand. It is **a client handler.**

Each and every node in the network has a node handler and a client handler, but a node may have a download handler only if it is downloading files from other nodes. All these 3 handlers together make up the Node.

Protocol: The DHT server and nodes follow certain communication protocol. For every message, a header of 16 Bytes is padded at the beginning of the message. This header contains the length of the payload. The header is first decoded and validated before accepting the rest of the message. Messages have a packet size of 2048 Bytes, so any messages larger then that are sent across in multiple packets. A default timeout of has been set for messages so that any packet loss shall terminate the message. All the messages are encoded and decoded using python's pickle module, which happens to be a

serializer. Serializing the messages enabled to send python lists and objects through the message payload.

Possible Improvements

One of the major area of improvement in my opinion happens to be in the way a node downloads. Currently each node is only capable of downloading from one other node, but it can be uploading to multiple nodes. It would be much more faster to download from multiple sources or parallely download different files.

Looking at the DHT Server, the server keeps a record of the active nodes explicitly by receiving an activate or deactivate message from the node. It would be better to implement a ticker that keeps the record of active connections implicitly. Furthermore a DHT Server that can broadcast the updated records to all the nodes in the network can also be a major improvement.

Lastly, a node that can take the position of a DHT server upon the DHT server’s termination can be a major improvement.