CS425A: Computer Networks

Course Project

P2P Chat loosely based on Gnutella

Group Name: SSLP
Members: Prateek (14492), Shivanshu (14659), Lukesh (14451), Shubham (160816)

Introduction

In this project we aim to design and implement a Peer to Peer (P2P) chat system loosely based on Gnutella, the first decentralised peer-to-peer network of its kind for file sharing. Although Gnutella is not actually associated with GNU, the name Gnutella is a blend of GNU and Nutella (a famous brand of Italian hazelnut flavored spread). For this project we have used *Go* Language, a language becoming popular for multiprocessing and computer network programming due to its built-in lightweight concurrency primitives.

Background

When we think of a chat system we most likely think of one of the most common chat systems like Facebook Messenger, Hike Messenger, Google Hangouts etc. All of these chat services require the user to first create an account and sign-in to their centralized server (which may be single or distributed depending on the chat service provider) to be able to use their service. A lot of people use many different chat systems to communicate with other people and this require them to keep track of many different accounts to be able to talk to everyone they want to talk to. What if there existed a system that require no previously created account, no login and uses no central server to communicate with their friends? This system will vaguely resemble the popular WhatsApp messaging service, in terms of resource usage by master server. The design described in this report is to implement such a system with the limited knowledge we have, just to have an exposure to network programming.

Gnutella

As mentioned earlier, it is one of the very first P2P file sharing network. In this communication mechanism, each Gnutella node needs to know at least one other Gnutella node address to work. Typically, a Gnutella node is initialized with a set of active node addresses . The node requests all of its connected nodes for any data and this request is then forwarded by its connected nodes to their connected nodes until the node having that data is reached. This process is called flooding. To avoid any request to forever flood the network a *Time To Live* (TTL) parameter (basically an integer count) is added to the request and each intermediate node decrement it by one before flooding the request received; this way the request will die out after certain number of hopes.

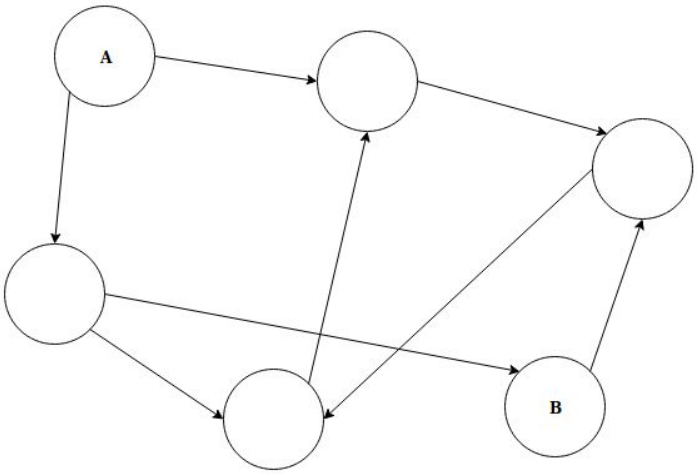

Figure 01: A typical Gnutella Mesh network

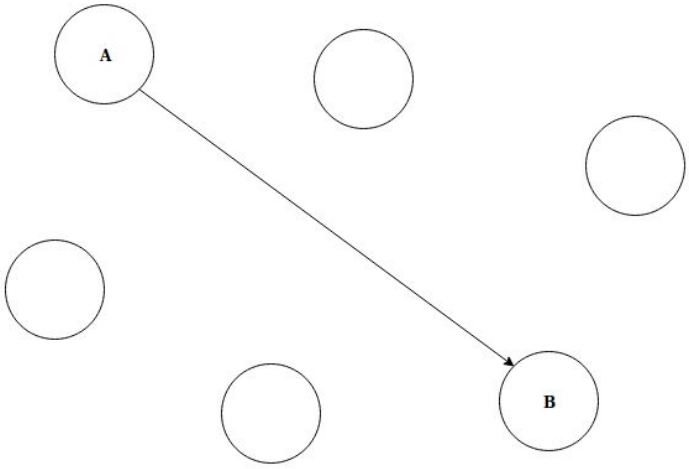

Figure 02: Direct connection setup in Gnutella after finding the data node

Chat System Architecture

Our chat system employs the Gnutella's flooding mechanism for delivering the chat messages. In our implementation, we have a master server and independent peer nodes. A peer's username (i.e. alias) is unique (subject to the active status) and is binded to the set <IP address, Port>. Each peer have their own dynamic set of connected nodes. This way we get a mesh type of network.
We have employed a master server which will do the following tasks:

- Maintain list of each active peer's info (IP address, port, username, last active timestamp)
- Refresh the last active timestamp of each peer on receiving refresh request from them
- Refresh the active peer list periodically
- Initialise and refresh each peer's set of connected nodes

The master server maintains the active peer list using a timeout mechanism, it deletes the peers who have been inactive more than 60 seconds. This is done so that any peer don't get disconnected from the network (i.e., to not get a disjoint mesh network) in the unwanted scenario of all its connected set of peers becoming inactive. For being in the active list of the master server, every peer needs to send a refresh request to the master server. In response to a refresh request, master server updates the last active timestamp of that peer and send a fresh set (having supplied buffer size no. of elements) of connected peers (these are all the active peers) to the requesting peer.
Having done with establishing the master server with the minimalist functionality, which is the main advantage over a typical heavily loaded server of a server-client based chat system; we move towards defining the functionality of peer node. We have borrowed the concept of TTL again from Gnutella to avoid the infinite flooding of messages, which will increase the overload unnecessarily.
A typical peer node will have following functions to perform:

- Refresh it's active status by periodically sending refresh request to master server
- Flood any message received which is not destined to it after decrementing TTL
- A way to avoid showing duplicate message received from multiple peers

The term *Flood* above means sending message to all the peers in its connected set of peers. A peer needs to send a refresh request to the master server every 30 seconds to be present in master's active peer list. Since it is a mesh connected network, the possibility of getting duplicate messages is quite high, so we implemented a MD5 checksum on the message. When the message reaches its destination, destination check whether that MD5 hash is already present (i.e. if that message has already been received), if not then we show that message to the user otherwise ignore that message. This way we will be able to avoid case of getting message duplicates.
For the client-side User Interface (UI), we have used a very simple terminal based UI powered by curses library. So this is all about the implementation part of the proposed chat system.

Salient features of this implementation

Implementation of chat system using a P2P approach have many advantages. Some of the very main advantages over the server-client model of chat system are:

- No single point of failure. Even if the master server goes down for a short interval of time, it won't affect the ongoing communication between the users. Since master server is just for updating the set of connected peers. This feature was successfully tested.
- Minimalistic resource usage at the master server end, no big data centers are required.

Some known drawbacks of P2P chat model

Even though we have many advantages over server-client model, but we have a few drawbacks here too. Some of those are:

- Chat backup on the peer side may not be possible for all users.
- Peer side need to have a moderate networking and computation resource, while in case of server-client model, a low-end device would suffice as a client.

Future implementable features

1) Adding Public and Private key (PKI) architecture for forwarding message. This will not only ensure confidentiality but also ensure the authenticity of the message. In current scenario, the message is sent in plain text.
2) Unique username feature can be implemented (might be based on something unique like Aadhaar number, though we are not a big fan of using Aadhaar number everywhere). In the current implementation scenario, a username is unique (i.e., some attacker can't establish connection using your username, the most they can do is refresh the timestamp of corresponding peer info) as long as that peer is continuously active. Someone else can claim that username if the previous user using that username becomes inactive (after timeout).
3) A interactive Web User Interface with chat saving functionality.

References

1) [Tommy Mattsson] 2012. A peer-to-peer based chat system. [ONLINE] Available at: http://www.diva-portal.org/smash/get/diva2:524732/FULLTEXT01.pdfIT [Accessed: 15 November 2018]
2) [Wikipedia] 2018. Gnutella. [ONLINE] Available at: https://en.wikipedia.org/wiki/Gnutella [Accessed: 15 November 2018]