Overview

This project is a P2MP-FTP protocol implementation using socket programming to share files among multiple hosts. One of the hosts acts as a client (that sends the file) and there can be multiple servers that receive the file.

The code is written in Python 2.7 and uses standard python libraries. The implementation divides the code into three python files :

1. p2mpclient
2. p2mpserver
3. utils

The instructions to run the programs are given in the README.md file of the repository.

Implementation

Client

The client implementation sends file to listening server. Following are the key points about the client:

1. The file F (to be sent) is given as a parameter to the client program
2. It sends a file F to N servers whose addresses are given as parameters to it. To each of these servers, each chunk is sent parallely (using multithreading to create a unique UDP socket on the client side for each server instance) and the client waits for all the servers to send an acknowledgement for the same chunk. If the client does not receive acknowledgement for a chunk, it then retransmits the packet to only that server until it receives an acknowledgement form it. Once acknowledgement from all the servers is received, the client transmit the next chunk of data. The reliability part of the communication is implemented via the rdt_send() method.
3. The client uses a socket timeout value of 0.1. This is the time which the client uses to determine packet loss and perform retransmission.
4. The client sends the file F in chunks of size MSS. The value of MSS is provided as the third parameter.
5. The sequence numbers for the MSS start with 1 and continue until the file is completely sent.
6. For each packet that does not receive acknowledgement, the client prints a message indicating the timeout and the sequence number of the corresponding packet. The packets are buffered locally until they are correctly received by all the servers.

Server

Following are the key points about the server:

1. The server implementation listen on a port given as parameter (7735 for instance).
2. The file F (to be received) is stored at a path given as a parameter to the program
3. The probabilistic loss factor (p) is also given as a parameter to the server. This value provided should be between 0 and 1. This value is used to model the effect of randomly losing a packet while it was in the network with a probability of p.
4. Based on #2, the server sends an acknowledgement to the client as a receipt of the packet or it drops the packet as if it did not receive it.
5. The server buffers the contents it receives from client in the destination file.
6. For each packet that was dropped, the server prints a message indicating packet loss and the sequence number of the packet that was lost.

Message Format

The packet consists of following fields in the order they are listed:

1. Sequence number of the packet. This binary value starts from 1 and is of size 32 bits.
2. Checksum of the packet. This is a 16 bit binary value calculated as the over sequence numbers, packet category and the payload. It is the one's complement of the one's complement sum of all 16-bit words in the above three fields.
3. Packet category is the field that defines the type of packet. The value used for a data packet is ‘0101010101010101’ and an acknowledgement is '1010101010101010'
4. A chunk of file is the payload of size MSS. The data is read and converted to binary format before transmission. On the receiver end, it is decoded into ascii encoding and stored.

The message structure can be visualised as shown in the diagram below.

Acknowledgement Number	
Checksum	Packet Category
Payload	