Department of Electrical and Computer Engineering
Concordia University

Communication Networks and Protocols - COEN 366

Winter 2024

Peer to Peer File System

Designed by: Ferhat Khendek

1. Introduction

The project consists of implementing in, Java, C++, or Python, a Peer-to-Peer File System (P2FS), over UDP and TCP. The description of the protocol(s) to implement is given Section 2 while the requirements are stated in Section 3.

2. Peer to Peer File System (P2FS)

The Peer-to-Peer File System (P2FS) consists of several clients and one server. The main goal of the P2FS is to allow for users (clients/peers) to share files. For the project we are dealing only with text files.

The role of the server is to keep track of the registered clients (peers), how they can be reached and the list of files available from each one of them. It will use this information to inform the clients about each whenever needed.

The communications between the clients (peers) and the server is through UDP, while the communication between the clients (peers) is through UDP at the beginning, but file transfer must be through TCP.

In the following we assume one user per client and therefore we use the terms user, client and peer interchangeably. The protocol(s) are described below.

2.1. Registration and De-registration

The server is always available at a fixed UDP socket and listening for incoming messages and communicates with the clients through UDP. You can fix the server UDP socket as you wish, for instance 3000.

A new user must register with the server before publishing or discovering what is available for share. A message “REGISTER” is sent to the server through UDP. For registering a user must send his/her name (every user has a unique name), IP Address and a UDP socket# it can be reached at by the server or a client.

REGISTER	RQ#	Name	IP Address	UDP socket#

Upon reception of this message the server can accept or refuse the registration. For instance, the registration can be denied if the provided Name is already in use. If the registration is accepted the following message is sent to the user.

REGISTERED	RQ#

If the registration is denied, the server will send the following message and provide the reason.

REGISTER-DENIED	RQ#	Reason

The RQ# is used to refer to which "REGISTER" message this confirmation or denial corresponds to. It is the same case of all the messages where RQ# is used.

A user can de-register by sending the following message to the server.

DE-REGISTER	RQ#	Name

If the name is already registered, the current server will remove the name and all the information related to this user.

In case Name is not registered, for instance, the message is just ignored by the server. No further action is taken by the server.

2.2. Publishing file related information

A registered client can publish and retrieve information about available files and where to download them from. When files become available for share at a given client, it informs the server with the following message.

PUBLISH	RQ#	Name	List of files

The server will add the list of files to the current list of files available from this client and acknowledge this by sending a confirmation message to the client.

PUBLISHED	RQ#

If publication is denied, because of errors like "Name" does not exists, the server sends the following message to the client.

PUBLISH-DENIED	RQ#	Reason

The client can try again by sending this "PUBLISH" message for a few times before giving up.

If a client decides to remove a file (or a list of files) from its offering, it sends the following message to the server.

REMOVE	RQ#	Name	List of files to remove

The server will remove the list of files from the current list of files available from this client and acknowledge this by sending a confirmation message to the client.

REMOVED	RQ#

If removal is denied, because of errors like "Name" does not exists, the server sends the following message to the client.

REMOVE-DENIED	RQ#	Reason

The client can try again by sending the "REMOVE" message a few times before giving up.

2.3. Server sharing information with the registered clients

Every time a change occurs at the server (a client deregisters, a client publishes files for sharing, a client removes files from sharing, etc., see also Section 2.5 for mobility case), the server must update all the registered clients with the UPDATE message containing the list of currently registered clients and the files currently proposed to share.

UPDATE	List of (Name, IP address, UDP socket#, list of available files)

An UPDATE message must be sent every 5 minutes at the latest even in case of no changes.

2.4. File transfer between clients (peers)

Once a client decides to download a file from a peer, it must set a TCP connection with that peer. For that purpose, it sends through UDP a FILE-REQ to the peer.

FILE-REQ	RQ#	File-name

If the file exists at destination and the peer is willing to share it, it must respond and provide the TCP socket# to which the client must connect for the purpose of file transfer.

FILE-CONF	RQ#	TCP socket#

Once the TCP connection is set, the peer will start transferring the file, through TCP, in small chunks not exceeding 200 characters using the following message (where Chunk# indicates the order/place of the Text in the original file).

FILE	RQ#	File-name	Chunk#	Text

The last chunk of the file is carried in a special message to indicate the last portion of the file.

FILE-END	RQ#	File-name	Chunk#	Text

While receiving these messages the client who requested the file puts the chunks together to compose again the original file. Upon complete reception of the file the client closes the TCP connection.

If the requested file does not exist at destination or for some other reasons the contacted peer cannot engage in a file transfer it sends the following message.

FILE-ERROR	RQ#	Reason

A peer must accept an arbitrary number of legitimate TCP connections for files sharing. The TCP socket# are chosen randomly.

2.5. Clients updating their contact information (mobility)

A registered user can always modify his/her IP address and/or UDP socket# using the following message.

UPDATE-CONTACT	RQ#	Name	IP Address	UDP socket#

The message is sent to the server. Upon reception of this message the server can accept the update and reply to the user using the message.

UPDATE-CONFIRMED	RQ#	Name	IP Address	UDP socket#

In case of denial of update, because of errors such as "Name" does not exist, the following message is sent to the user.

UPDATE-DENIED	RQ#	Name	Reason

As for the registration and publication, every time a change occurs the server must update all the registered clients with the UPDATE message containing the list of currently registered clients and the files currently proposed to share.

3. Requirements

Project should be done in groups of 3 students. By Feb 15, 2024, you should send your team list including student names, ID numbers and **ECE** email addresses to ferhat.khendek@concordia.ca.

Design and implement the client and server that follow the protocol(s) aforementioned. The coding of the protocol messages is part of your design, i.e. you must come out with the appropriate coding of the messages. You can decide to use simple text message, etc.

The information stored in the server should be persistent, i.e. if the server crashes and is restored it will recover all the information as before crashing.

The client and the server must be multi-threaded.

Reporting: Server and clients should be reporting their communications to the users of the system using a log file or printing directly into the screen. In other words, during the demonstration, I would like to see the messages sent and received, progress and failures.

Assumptions/Error/Exception Handling
You should be aware that the description as it is does not state everything. For instance, what happens if a client receives a response with a RQ# that does not correspond to any of its (pending) requests?

And more to be discussed in class …

State and document clearly any assumption you make beyond the assumptions made by the instructors.

Extra: Notice that in this project we do not require authentication of users. Extra marks will be given to students who add an authentication scheme to the proposed P2FS. Also, a GUI is not required as long as we can run the clients and the server and see what is going on with the messages. Extra marks will be given if you decide to build a GUI.

You should hand in a report, by Week 13, where you clearly document your assumptions, design decisions, code and experiments. You should also clearly state the contributions of every member of the group. Every student must contribute **technically** (designing and implementing the protocol) to the project.

A demo will be held during Week 13 of this Winter term. During the demo the members of the group should all be ready to answer questions.

During the demo we may also go through the code itself as well as the report.

The project will be discussed further in class.