Department of Electrical and Computer Engineering
Concordia University

Communication Networks and Protocols - COEN 366

Fall 2024

Peer-to-Peer Shopping System

Designed by: Ferhat Khendek

1. Introduction

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

2. Peer to Peer Shopping System (P^2S^2)

The Peer-to-Peer Shopping System (P^2S^2) consists of several peers (a peer can have two roles: buyer and seller) and one server. The main goal of the P^2S^2 is to allow for users (peers) to search for items/goods and buy them from other peers at a reasonable price. The peers do not interact directly with each other, they communicate only with the server.

To have access to the service the clients (peers) need to register with the server. The first role of the server is to keep track of the registered clients (peers) and how they can be reached. It is also the intermediary between the peers for searching for items/goods and purchasing them from each other as described in the section below.

Some of the communications between the clients (peers) and the server are through UDP, other communications for finalizing the purchase are through TCP. The peers never communicate with each other, except for shipping the goods through surface mail.

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

The server is always reachable/available at a known IP address, fixed UDP socket where it is listening for incoming messages. For registration and item search purposes the clients communicate with the server through UDP, but for finalizing a purchase they communicate through TCP (see Section 2.3). You can fix the server UDP socket# as you wish, for instance 5000.

2.1. Registration and De-registration (Communications through UDP)

A new user must register with the server before using the service for items/goods purchasing/selling. For registering a user must send his/her name (every user has a unique

name), IP Address, a UDP Socket# and a TCP Socket# where it can be reached by the server. A message "REGISTER" is sent to the server through UDP.

REGISTER	RQ#	Name	IP Address	UDP Socket#	TCP 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 or when the server cannot handle additional clients. 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. Searching for items (Communications through UDP)

A registered client can search for items to buy when needed. A registered client looking for an item, sends a message to the server describing the item (name of the item, and a small description) and the maximum price it is willing to pay for it.

LOOKING_FOR	RQ#	Name	Item_Name	Item_Description	Max_Price

Upon reception of this message, the server asks all the registered peers, except the buyer who sent the LOOKING_FOR message, if they have the item in question.

SEARCH	RQ#	Item_Name	Item_Description

The RQ# of the SEARCH message may be different from RQ# of the LOOKING_FOR message as the first one is used to number a requested generated by the buyer and the second one to number a request generated by the server.

A registered peer who received the message and has the item in store can answer the server, to offer the item and indicate the price (which can be higher than the maximum set by the buyer).

OFFER	RQ#	Name	Item_Name	Price

The name in the message OFFER refers to the name of the peer who is offering the item. The RQ# in the OFFER message is the same as the RQ# in the SEARCH message, this is how the SEARCH and the OFFER match.

If the server receives no offer for this item after 5 minutes, it answers the buyer about the non-availability of the desired item by sending the following message.

NOT_AVAILABLE	RQ#	Item_Name	Price

The RQ# in this message is equal to the RQ# of the initial LOOKING_FOR message sent by the buyer. This is how the buyer knows exactly what this NOT_AVAILABLE message refers to.

If the server receives only offers at prices higher than the maximum set by the buyer, it informs the seller offering the lowest price about the maximum price the buyer is willing to pay using the following message.

NEGOTIATE	RQ#	Item_Name	Max_Price

The RQ# is the same as for the message OFFER sent by the seller. This is how we keep track of the stream of communication about an item.

In the case the seller decides to accept to sale at the maximum price set by the buyer, it sends the following message to the server.

ACCEPT	RQ#	Item_Name	Max_Price

Here again, the RQ# is the same as for the message OFFER sent by the seller.

After sending the ACCEPT message, the item is reserved automatically by the seller, and the server informs the buyer with the following message.

FOUND	RQ#	Item_Name	Price

The Price here is equal to the Max_Price set by the buyer. The RQ# in this message is equal to the RQ# of the initial LOOKING_FOR message sent by the buyer.

In the case the seller decides not to accept to sale at the maximum price set by the buyer, it sends the following message to the server.

REFUSE	RQ#	Item_Name	Max_Price

Here again, the RQ# is the same as for the message OFFER sent by the seller.

The server informs the buyer with the following message.

NOT_FOUND	RQ#	Item_Name	Max_Price

The RQ# in this message is equal to the RQ# of the initial LOOKING_FOR message sent by the buyer. This message is interpreted by the buyer as the item is not available at the fixed maximum price, and this is different from the case where the item is not available at all.

If the server receives offers from peers offering the item at a price lower than the maximum set:

- It reserves the item with the seller offering it at the lowest price using the RESERVE message.

RESERVE	RQ#	Item_Name	Price

- It informs the buyer about the availability of the item and the requested price using the FOUND message.

FOUND	RQ#	Item_Name	Price

After receiving the information about the availability of the item at a price less or equal to the maximum price, the buyer decides to move forward or not, it informs the server accordingly. If the buyer decides not to move forward it informs the server using the message CANCEL. The server informs the seller to cancel the reservation using a similar message CANCEL.

CANCEL	RQ#	Item_Name	Price

If the buyer decides to purchase the item at the requested price, it informs the server with the BUY message.

BUY	RQ#	Item_Name	Price

Buyer, server and seller move to finalize the purchase.

2.3. Finalizing the purchase (Communications through TCP)

The server opens two TCP connections, one to the buyer and one to the seller. The server will require more information from both sides, like credit card information and addresses. The server sends the following message to both sides through the TCP connections.

INFORM_Req	RQ#	Item_Name	Price

The RQ# is a new one generated by the server. Item_Name is the name of the item being purchased and Price is the price agreed on.

Both sides (buyer and seller) answer with the following message and provide their respective information.

INFORM_Res	RQ#	Name	CC#	CC Exp_Date	Address

CC# stands for the credit card number while CC Exp_Date stands for the credit card expiration date.

The server proceeds with charging the buyer's credit card and crediting the seller's credit card with 90% of the price and keeps 10% as transaction fees. If the operation does not go through, the transaction is cancelled, and buyer and seller are informed with the following CANCEL message.

CANCEL	RQ#	Reason

At this point everything is cancelled.

If the operation goes through, the server informs the seller and provides the address of the buyer with the following message where it provides the name of the buyer and his/her address for shipping.

Shipping_Info	RQ#	Name	Address

Both TCP connections are closed after this exchange. The connection to the buyer is closed by the server, while the other TCP connection is closed by the seller.

The seller will then ship the item to the buyer through surface mail.

3. Requirements

Project should be done in groups of three (3) students. By September 20, 2024, you should send your team list including student names, ID numbers and Concordia email addresses to ferhat.khendek@concordia.ca.

Design and implement the client (peer) 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 client (peer) and the server must be multi-threaded.

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.

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? Or if a client not yet registered is looking for an item to buy and sends a message to the server ….

Offering of items

There is no need to keep a database of available items and their prices for every peer. During the execution of the protocol, we will decide for each peer if it has the item or not and at which price it offers it. If you want to have a list of items offered by each peer and stored locally with the peer, you can also do so.

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 P^2S^2. 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.

We strongly recommend the following schedule to avoid the rush of the end of the term:

- **Phase 1:** Registration and deregistration of peers**. (by mid October)**
- **Phase 2:** Request handling and offer management through UDP. **(by first week of November)**
- **Phase 3:** Finalization of transactions using TCP. **(by third week of November)**

Following this schedule, you will be able to show your work to the Lab TAs and fix potential problems before final submission and demonstrations.

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. During the demo the members of the group should all be present and 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 during lectures.