Design Doc:

P2P “Market” Implementation

This project is meant to simulate a peer-to-peer “bazaar”, wherein there are multiple “sellers”, each carrying a certain number of items. “Buyers” can connect to the P2P network and search for items to purchase; if a seller has an item, the buyer will send a purchase request, and the amount of that item the seller has in stock will decrease by one.

Classes:

**P2PNode**: Arbitrary node in the network. *Should* be treated as abstract, i.e. only inherited by BuyerNode and SellerNode and not implemented on its own, but a bare p2pNode can function as part of the network.

Members:  
idNum: Integer ID number uniquely assigned by a tracker.  
trackerSocket: Socket for connection to a Tracker object.  
listenSocket: Socket for incoming connections. Port is uniquely assigned by tracker, since this is running on a single computer.  
connectedNodeList: List of active socket objects, each representing another p2pNode this p2pNode is connected to.  
searchRequestsSentList: List of search request ID numbers that this node has already passed on.  
searchRequestsReceivedDict: Dictionary of search\_item() requests. Each key is an ID number representing a search request, with the corresponding value a list of UUIDs representing the p2pNodes that this node has received that request from.  
connectionLock: Semaphore that ensures multiple connections do not interfere with each other.  
maxConnections: Maximum number of P2PNodes this node can be connected to. Further connections will not be immediately refused, but will be opened long enough to send a message stating “no slots available” and then closed.  
pathDict: Dictionary matching other node IDs to path lists. Directly connected nodes are *not* included in the dictionary.

Methods:  
joinNetwork(tracker): Connects to the Tracker object and selects a random p2pNode in the network to connect to.   
requestOtherNode(): Requests an ID and port number for another P2PNode from the Tracker.  
connectNode(otherID): Establishes a connection to another P2PNode.  
disconnectNode(otherID): Breaks a connection with a P2PNode. Not recommended!  
shutdown(): Alerts the Tracker that this node will no longer be available for connections, then closes all sockets and enters a ready-for-deletion state.  
handleReceived(packetString): Handles the data received from another node; bare P2PNode version calls passOnSearchRequest if it’s a search request, or passOnPathedRequest if it’s a request with a path built in. Otherwise it derps.  
passOnSearchRequest(searchRequest): Checks a SearchRequest’s ID first against searchRequestsSentList to see if this node has already passed on this request. If not, waits for the connectionLock to unlock, then checks the request against SearchRequestsReceivedDict, adds its own ID to the search request’s path list, and fires it to any connected nodes that it did not receive the request from.   
passOnPathedRequest(request): Passes a request with a path down the chain..

**Tracker**: Centralized “server” that keeps track of which nodes are in the network and what ports they are using. Does not mediate sales or search requests.

Members:

listenSocket: Socket for incoming connections.

activeNodeDict: Dictionary of active nodes in network. Each key is an integer representing a node’s ID number, each value is the port number used by the node with that ID number.

Methods:

handleReceived(packetString): Handles the data received from a P2PNode.  
sendNodeInfo(): Sends the ID and port number of a P2PNode to a different P2PNode that requested it.

**BuyerNode:** P2PNode child class that searches for and purchases items.

Members:

desiredItem: Current item the BuyerNode is searching for.

shoppingBag: List of items the BuyerNode has bought.

waitingForSearchReply: Flag set when searchItem is fired and unset when a buyer replies.

Methods:

searchItem(targetItem): Generates a search request and fires it to all connected nodes.

buyItem(sellerID, targetItem): Fires a buy request directly to the seller if connected; otherwise fires it along the path. Awaits a BuyOK from the seller; after a timeout, resends the buy request.

handleReceived(packetString): Handles incoming data from a connection. BuyerNode version can pass on search requests like the P2PNode version, and can also handle replies to search requests and BuyOK messages.

handleSearchReply(searchReply): Adds buyer’s path to the pathDict if not already there (or if it’s shorter), then calls buyItem.

**SellerNode:** P2PNode child class that carries an item inventory and sells to buyers.

Members:

itemInventory: List of items (probably just strings?) the SellerNode has in stock.

buyRequestsReceived: List of buy request IDs the seller node has received.

Methods:

handleReceived(packetString): Handles incoming data from a connection. SellerNode version will reply to a search request if itemInventory contains the item requested, or else pass on the request like the P2Pnode version. It can also handle buy requests.

reply(buyerID): When a search request is received, attempt to make a connection to the buyer, and fire a search reply if successful. Otherwise fire back a reply along the path. Either way, mark the relevant item as locked.

handleBuyRequest(): When a buy request is received, remove the bought item from inventory if possible and store the request ID in buyRequestsReceived. (Otherwise error.) Send back a BuyOK message along the path. If the ID is already in buyRequestsReceived, resend the BuyOK but do not subtract from inventory.

MESSAGE STRUCTURE: Packed into json using Python’s built-in functionality for that purpose.

type: String designating the “type” of the message.

path: Series of IDs showing the path the message should take along the P2P network. Nodes should only pass along the message if they’re on the path, and only attempt to “handle” it if they’re the last on the path.

returnPath: Series of IDs showing the path a search request has travelled along, used to provide a connection path when a direct connection is not available.

item: An item name. Searches, buy requests, and buyOKs should carry one of these.

MESSAGE TYPES:

Search: Sent from a buyer. Flooded throughout the network.   
type: “search”  
path: n/a (not in object)  
returnPath: initially contains originating node, each node that receives it appends its own id  
item: desired item  
id: Randomly generated integer.

SearchReply: Sent from a seller, directly or along a path.  
type: “reply”  
path: Reverse of returnPath in corresponding search  
item: Item desired by corresponding search.  
id: ID number of corresponding search.  
  
Buy: Sent from a buyer, directly or along a path.  
type: “buy”  
path: Direct or copied from pathDict  
item: Item desired for purchase.  
id: Randomly generated integer.

BuyOK: Acknowledgement from seller that a buy request has been received.  
type: “buyOK”  
path: Direct or copied from pathDict  
id: ID contained in buy request.

ThisIsMe: Information sent along with a newly established connection.   
type: “thisisme”  
port: Node’s listen port.  
id: Node’s ID number. Should be -1 if being sent to the tracker (the tracker will send back a proper ID number).

ThisIsYou: ID information sent from the tracker back to the node in response to a ThisIsMe.  
type: “thisisyou”  
port: Randomly assigned listen port  
id: Randomly assigned ID number

Error: Message indicating something’s gone wrong.  
type: “error”  
code: Short, parseable code that tells the program what’s gone wrong.  
Code list:   
“notim” = Node failed to receive a timely ThisIsMe message after a connection.  
“notiy” = Node failed to receive a timely ThisIsYou message from the tracker after a connection.  
“wtf” = Node or tracker received a message it didn’t know how to handle. Note that *receiving* an error message should never trigger a wtf.  
“wrongway” = Node or tracker received a message with a path that did not include it.  
info: Optional human-readable message that can tell a user what’s gone wrong.

Disconnect: Message indicating that the remote socket is about to close and this one should do likewise.  
type:”dc”

Ping: Message asking the recipient to send a reply. Be careful to avoid infinite ping loops!  
type:”ping”

NodeRequest: Message from a node asking the tracker to send it the ID and port of another node.  
type: “nodereq”

NodeReply: Reply to a NodeRequest with a node’s ID and port.  
type: “nodereply”  
id: Another node’s ID number  
port: Corresponding node’s listen port