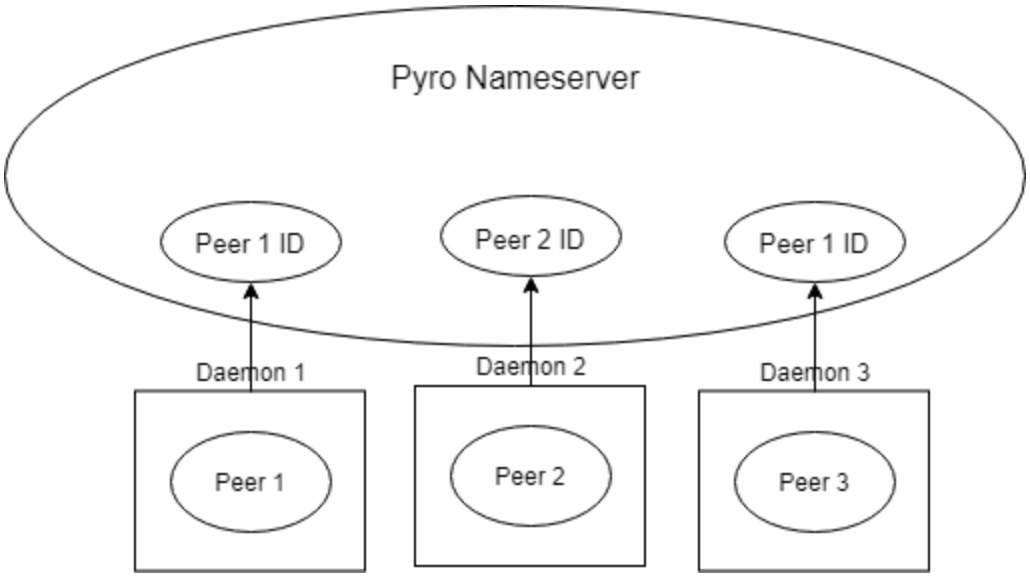
Lab 1: Asterix and the Bazaar Program Design
Chung Yang and Russell Lee

Overall Design

Our implementation makes use of the Pyro library to setup the P2P network and allow remote procedure calls. Peers are defined to be Pyro objects, which are just Python objects which are registered with Pyro to allow remote access. All peers are registered on a pre-specified Pyro nameserver. This nameserver gives us control of what peers are on the network, and it provides newly joined peers a list of neighbors to start with. Furthermore, each peer is registered with a Pyro daemon so it can accept incoming requests. Name server lookup only happens during initialization of peers and when a direct transaction happens between a buyer and seller. The need for name server lookup during direct transaction can be eliminated by passing a tuple of (peer_id, peer_location). We ran out of time to implement this but the design allows no name server loop beside the initial peer joining. All communication would be done between peers.



We follow the overall system specification outlined in the writeup - peers are randomly initialized to be buyers or sellers, and the desired good/inventory good of each peer is also randomly initialized. Buyers continually send lookup requests to their neighbors for the their desired good, while sellers continually listen for buyers.

How it Works

We follow the component structure and component interaction laid out in the lab handout. In this section, we describe how each component accomplishes its intended function, as well as any notable features.

Peers:

We define a peer as a subclass of a Python threading object, so multiple peers can join the network concurrently with one execution of the python script. Every peer uses a thread pool with 10 threads to handle incoming requests.

Neighbor assignment in our system is dynamic - once a peer registers to the nameserver, it asks the name server for a list of all the peers that are registered and have the specified ID format. (So we don't fetch other people's registered objects). From the list it obtains, it randomly picks a neighbor that has the same hostname and also randomly picks a neighbor that has a different hostname, if the aforementioned peers exist. When a peer adds another peer to its neighbor list, it issues a remote procedure call to let that neighbor peer knows about its existence. By doing this, we ensure the connection between each peer is birectional, so a situation where a peer becomes a network sink node won't happen and every single peer can be reached. Dynamic leaving is not implemented, so running reset_ns.sh is necessary to clean up inactive peers.

run():

This function has a seller loop and buyer loop. The seller loop is just a simple while loop that timeout one second every iteration. The buyer loop initiates the lookup request and it waits until all requests come back before initiate buy request. Upon every purchase, successful or not, a random item is picked up for the buyer.

lookup(product_name, hopcount, id_list):

Lookups are initiated by buyers to ask its neigbhors about its desired product. To allow backtracking the seller response through the network once a seller is found, lookup has an additional argument to grow a list of peers that receive this lookup request. Anyone who is not a matching seller propagates the lookup to its neighbors (avoiding the neighbor which just send the request), decrements the propagated hopcount, if hopcount is 0, the message is discarded, and adds its own ID to the peer list. If a matching seller receives the lookup, it will append its ID to the beginning of id_list and pops an item from the list to use as the recipient for a reply message.

reply(id_list):

Replies are initiated only when a matching seller has received a lookup from an interested buyer. Calls to reply also have an argument of a shrinking list of peers to backtrack through. Once the list has shrunk to just one peer, then the seller ID has reached the original buyer. The original buyer adds the sellerID to its current list of sellers that can provide the product, and reply propagation stops.

buy(peer_id):

Buys are initiated only when a buyer has a list of sellers that can provide the product. If it has multiple sellers, it just picks one at random to contact, and clears its list of available sellers after sending a buy request to a seller. Note that a seller can receive multiple buy request concurrently. To deal with this, we put a lock on the item inventory of a seller. Its item inventory

can only be decremented one at a time from buy requests. It changes the item a seller sells and resets the item counter when the current item is sold out.

Design Tradeoffs

The general design is unstructured because a peer randomly picks its neighbors. One disadvantage of that is certain buyers might get longer response time because they are located far from the sellers. Another disadvantage is that a message might not be able to propagate through the whole network if hop count is to low. Advantages of unstructured design is easier implementation, no addition logic needs to be included in the code to maintain the network structure.

Further Work

We can implement a way to allow the peers to dynamically leaves the network. One way to do this is by keeping a count for each neighbor. If a neighbor fails to be connected, we decrement its point and take it out of neighbor list when the point falls below 0.