

COMPSCI 677: Distributed and Operating Systems

Lab 1

Jenish Bajracharya, Aditya Chaloo

October 22, 2024

1 Introduction

This report documents Asterix and the Bazaar, a peer-to-peer trading system that simulates the exchange of three goods **salt**, **boar**, **fish** between peers, which can act as buyers and sellers. Each buyer initiates a search for an item by communicating with its neighboring peers, and sellers respond if they have requested item in stock. Each buyer or seller interacts through message exchanges to search for products, make purchases, and confirm transactions. The design supports dynamic interactions, allowing peers to adapt to changes in stock, handle timeouts, and buy multiple products.

2 System Architecture

2.1 Peer

The **Peer** class serves as the core component for each participant in the network. Each peer operates either as a buyer or a seller, with distinct functionalities tailored to their roles. We summarize the functionality for each peer via the finite state machine.

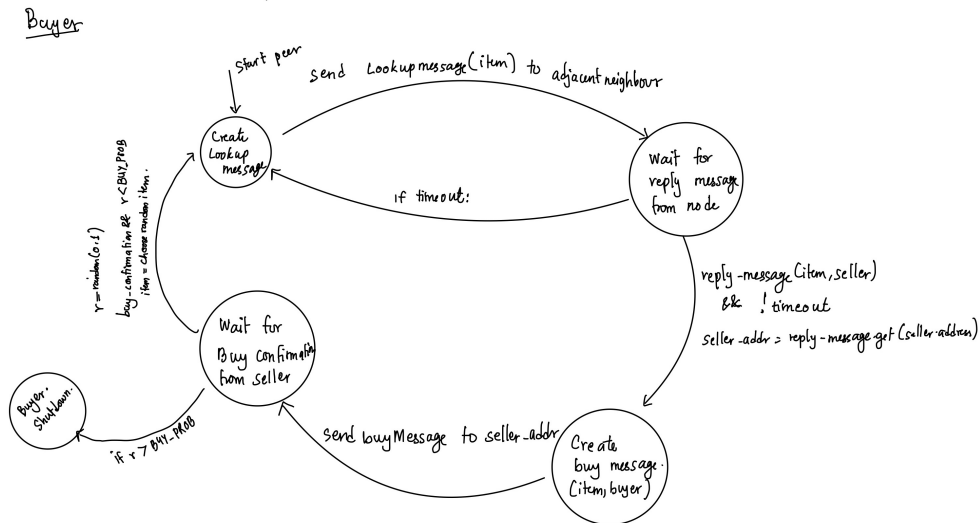


Figure 1: FSM for Buyer

2.1.1 Methods

send_message(addr, message) Serializes and sends messages to the specified address. Implements error handling to manage failed transmissions gracefully.

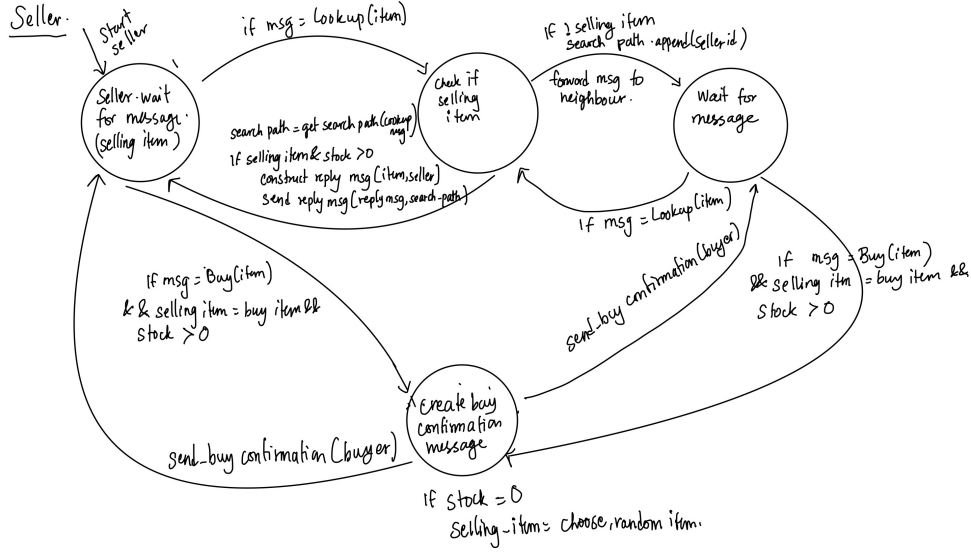


Figure 2: FSM for Seller

`lookup_item(product_name, hopcount)` Initiates a search for a specified product. If `product_name` is `None`, selects a random item from the available list. Records the start time for Round trip time (RTT) calculation and sends lookup messages to all neighboring peers.

`handle_buy_confirmation(message)` Processes buy confirmation messages from sellers. Updates transaction counts, calculates RTT, and decides whether to initiate further lookups or shut down based on the buy probability defined in `config.py`. If the probability is 1, the buyer always initiates lookup for another item that is chosen randomly.

`handle_lookup(message, addr)` Handles a lookup request from a buyer or another peer. Manages the internal cache by adding new requests and evicting the oldest entry if the cache exceeds its size limit. If the peer is a seller with the requested product and available stock, it sends a reply to the buyer. Otherwise, if the hop count is greater than zero, it propagates the lookup request to neighboring peers. If the hop count reaches zero, the message is discarded.

`handle_reply(message)` Handles a reply message recursively. If there is a reply path remaining, it forwards the reply to the next peer in the path. If the reply path is empty and the peer is a buyer, it decides to purchase the item by sending a buy message to the seller. If the peer is not a buyer, it acknowledges the receipt of the reply without further action.

`handle_buy(message, addr)` Processes a buy request from a buyer. If the peer is a seller with sufficient stock of the requested product, it decrements the stock and sends a buy confirmation reply to the buyer. If the stock depletes to zero, the seller selects a new item to sell and resets the stock. If the stock is insufficient, it sends a buy confirmation reply indicating the failure to complete the purchase.

`shutdown_peer()` Gracefully shuts down the peer by closing the socket and setting the running flag to `False`.

2.2 Network Design

Peers are first connected in a linear topology, where each peer is connected to its immediate successor and predecessor. This ensures that the network is initially fully connected, preventing isolated peers.

To introduce randomness and prevent a rigid network structure, peers are further connected to random peers, ensuring that no peer has more than three neighbors.

2.2.1 Network Diameter

The network diameter is the longest shortest path between any two peers in the network. We utilize the `graph_diameter` function (located in `utils/network_utils.py`) to compute the diameter of the constructed network. The hop count for search requests is set to be one less than the network diameter to optimize search efficiency while ensuring broad coverage.

2.3 Message Handling

The system employs a message-passing mechanism using sockets to facilitate interactions between buyers and sellers. Message handling is managed by `utils/messages.py`.

2.3.1 Message Classes

LookupMessage Initiated by buyers to search for a specific item. The lookup message keeps bouncing on the network until hopcount reaches 0.

ReplyMessage Initiated by the seller, for any **LookupMessage** request, if the seller has the requested item, it creates a reply message that traverses recursively back to the buyer based on `reply_path`.

BuyMessage Sent by buyers to sellers to initiate a purchase.

BuyConfirmationMessage Sent by sellers to buyers to confirm the outcome of a purchase. If the purchase is successful, the `status` tag is `True`.

3 Testing

3.1 Running the Scripts

Run the `main.py` script with the desired number of peers as an argument. For example, to start a network with 6 peers:

```
python main.py 6
```

3.2 Test Case Categories

- **Functional Tests:** Verify that the system's functionalities work as intended.
- **Edge Case Tests:** Evaluate the system's behavior under unusual or extreme conditions.
- **Error Handling Tests:** Ensure that the system gracefully handles errors and exceptions.

NOTE: Terminal outputs are formatted as `[{peer_id} <output>]` which shows, the output is generated by peer with `peer_id`.

3.3 Functional Test Cases

1. TC_01: Peer and Network Initialization

- **Description:** Verify that peers initialize correctly with the specified roles (buyer or seller) and the network is created where buyers and sellers have random neighbors with not peer having more than three neighbors.

```

Network structure initialized:
Peer 0 (seller) connected to peers [1, 5, 2]
Peer 1 (buyer) connected to peers [0, 2, 5]
Peer 2 (buyer) connected to peers [1, 3, 0]
Peer 3 (seller) connected to peers [2, 4]
Peer 4 (buyer) connected to peers [3, 5]
Peer 5 (buyer) connected to peers [4, 0, 1]
Peer 0 (seller) with item salt listening on port 5000...
Peer 1 (buyer) with item None listening on port 5001...
Peer 2 (buyer) with item None listening on port 5002...
Peer 3 (seller) with item fish listening on port 5003...
Peer 4 (buyer) with item None listening on port 5004...
Peer 5 (buyer) with item None listening on port 5005...
Network diameter is 2

```

Figure 3: Steps for TC_01: Peer Initialization

- **Expected Outcome:** All peers are randomly initialized with their roles and Network is configured correctly with each peer connected to at most 3 random peers .
- **Result:** Upon initializing the network with 6 peers, we observe that peer with `peer_ids` 0, 3 are assigned as sellers selling `salt` and `fish` respectively. The rest of the peers are buyers. The provided connection details confirm that each peer has at most 3 neighbors. For example Peer 0 (seller) is connected to Peers 1, 5, and 2.

2. TC_02: Message Sending and Receiving

- **Description:** Ensure that peers can successfully send and receive messages using the socket-based message-passing mechanism.

```

Network structure initialized:
Peer 0 (buyer) connected to peers [1]
Peer 1 (seller) connected to peers [0]
Peer 0 (buyer) with item None listening on port 5000...
Peer 1 (seller) with item fish listening on port 5001...
Network diameter is 1
Buyer 0 is initiating a lookup for salt with hopcount 1
21.10.2024 20:23:53.593 [0] Initiating lookup for salt
[0] Lookup Message: {'request_id': 'c5678fd614af7075ada3922b162c495c11d04551e4d3423ec912cedd02f14d16', 'type': 'lookup', 'buyer_id': 0, 'product_name': 'salt', 'hop_count': 1, 'search_path': [(0, 'localhost', 5000)], 'last_peer_id': 0}]
[0] Looking for salt with neighbor 1
[1] Received Message: {'request_id': 'c5678fd614af7075ada3922b162c495c11d04551e4d3423ec912cedd02f14d16', 'type': 'lookup', 'buyer_id': 0, 'product_name': 'salt', 'hop_count': 1, 'search_path': [(0, 'localhost', 5000)], 'last_peer_id': 0}

```

Figure 4: Steps for TC_02: Message Sending and Receiving

- **Expected Outcome:** Peer 1 receives the exact message sent by Peer 0.
- **Result:** Buyer 0 initiates the lookup for salt, seller 1 receives the same lookup message as the message sent by the buyer.

3. TC_03: Item Lookup Functionality

- **Description:** Validate that the `lookup_item` method correctly searches for items and handles cache management.
- **Expected Outcome:** The `LookupMessage` traverses through the network based on the forwarding algorithm. If the seller is found, the `ReplyMessage` from the seller traverses backward recursively to the buyer. If the seller is not found and timeout is triggered, then the buyer looks for another item.
- **Result:** In figures 14, 15, the network configured with 4 peers with neighbors hard-coded such that 0-1-2-3-0. In 14, peer 0 is looking for item salt which seller 3 has in stock. The request first gets forwarded from peer 0 to 1. Peer 1 forwards the same message to peer 2 and so on until it reaches peer 3. Since peer 3 has the item, the item traverses back from peer 3 to 2 to 1 and then 0. In 15, the buyer cannot find the item (`fish`) because of which a timeout is triggered, after which the buyer look for another item (`salt`)

```

Network structure initialized with hardcoded neighbors:
Peer 0 (buyer) connected to peers [1]
Peer 1 (seller) connected to peers [0, 2]
Peer 2 (seller) connected to peers [1, 3]
Peer 3 (seller) connected to peers [2]
Peer 0 (buyer) with item None listening on port 5000...
Peer 1 (seller) with item boar listening on port 5001...
Peer 2 (seller) with item boar listening on port 5002...
Peer 3 (seller) with item salt listening on port 5003...
Network diameter is 3
Buyer 0 is initiating a lookup for salt with hopcount 2
21.10.2024 20:42:28.526 [0] Initiating lookup for salt
[0] Looking for salt with neighbor 1
[1] Forwarding lookup for salt to Peer 2
[2] Forwarding lookup for salt to Peer 3
[3] Sent reply for 390cc829c8ac5541a91de38f11ffe42a05df64620e1e4176728fb3d18f4c7026 to peer 2 for item salt
[2] Sent reply to peer 1 for item salt with id 390cc829c8ac5541a91de38f11ffe42a05df64620e1e4176728fb3d18f4c7026
[1] Sent reply to peer 0 for item salt with id 390cc829c8ac5541a91de38f11ffe42a05df64620e1e4176728fb3d18f4c7026
[0] Deciding to buy item salt from seller 3

```

Figure 5: Case: Seller Found

```

Network structure initialized with hardcoded neighbors:
Peer 0 (buyer) connected to peers [1]
Peer 1 (seller) connected to peers [0, 2]
Peer 2 (seller) connected to peers [1, 3]
Peer 3 (seller) connected to peers [2]
Peer 0 (buyer) with item None listening on port 5000...
Peer 1 (seller) with item boar listening on port 5001...
Peer 2 (seller) with item salt listening on port 5002...
Peer 3 (seller) with item salt listening on port 5003...
Network diameter is 3
Buyer 0 is initiating a lookup for fish with hopcount 2
21.10.2024 20:55:42.135 [0] Initiating lookup for fish
[0] Looking for fish with neighbor 1
[1] Forwarding lookup for fish to Peer 2
[2] Forwarding lookup for fish to Peer 3
[3] Hopcount 0 reached for request 341ae37bc3f4c486a8ac1b48c68784538d498f9c9f63c9773a0001b21d40b. Discarding message.
[0] No response received for fish. Timing out and selecting another item.
[0] Searching for a new products salt
21.10.2024 20:55:43.136 [0] Initiating lookup for salt
[0] Looking for salt with neighbor 1
[1] Forwarding lookup for salt to Peer 2
[2] Sent reply for 85bc02f0bea0101a8b05726a071e00a798c2917735a25a7c779ee5fb42e1d57 to peer 1 for item salt
[1] Sent reply to peer 0 for item salt with id 85bc02f0bea0101a8b05726a071e00a798c2917735a25a7c779ee5fb42e1d57
[0] Deciding to buy item salt from seller 2

```

Figure 6: Case: Seller not found

4. TC_04: Purchase Process

- **Description:** Test the end-to-end purchase process from item lookup to buy confirmation.

```

network diameter is 3
Buyer 0 is initiating a lookup for fish with hopcount 2
21.10.2024 21:23:48.697 [0] Initiating lookup for fish
[0] Looking for fish with neighbor 1
[1] Forwarding lookup for fish to Peer 2
[2] Sent reply for 6d641358d8eb695c6a809c5cb7f1e1de1900df1a9cb6871422677d67f735e756 to peer 1 for item fish
[1] Sent reply to peer 0 for item fish with id 6d641358d8eb695c6a809c5cb7f1e1de1900df1a9cb6871422677d67f735e756
[0] Deciding to buy item fish from seller 2
[2] Sold item to buyer 0. Remaining stock: 0
[2] Sold out of fish. Now selling boar
21.10.2024 21:23:48.698 [0] bought product fish from seller 2

```

Figure 7: Steps for TC_04: Purchase Process with restocking

- **Expected Outcome:** The purchase is completed successfully, stock is decremented on the seller's side, and Buyer Peer receives a confirmation. If the stock goes down to zero, the seller restocks with another item.
- **Result:** The network setup is same as TC_03, however the `SELLER_STOCK` in `config.py` is set to 1 which means upon initialization, each seller starts with 1 item. Buyer 0 is looking for `fish` which seller 2 has it in stock. Upon receiving the reply message from seller 2, buyer 1 initiates to buy the item directly. Once the buyer receives the buy confirmation, it prints out the timestamp at which the execution was completed. For the seller, the stock for `fish` is decremented to 0, so seller prints out `Sold out of fish. Now selling boar`. New buyers can then look for `boar` with seller 2.

5. TC_05: Peer Shutdown

- **Description:** Verify that peers can shut down gracefully without disrupting the network.
- **Expected Outcome:** Peer shuts down gracefully, closes its socket, and the network remains functional with remaining peers unaffected. Peer shutting down is defined based on the the `BUY_PROBABILITY` value.
- **Result:** Figure 8, 9, 10 shows the buyer shutting down process for varying `BUY_PROBABILITY`. With $p = 0$ we can expect once a transaction is completed, the buyer shutdown (figure: 8). For case where $p = 1$, the buyer keeps searching and buying for products (figure: 10).

3.4 Edge Case Test Cases

1. TC_06: No Available Sellers

- **Description:** Test the system's behavior when no sellers have the requested item.

```

C:\Users\j\OneDrive\Desktop> python3 test_main.py
Network structure initialized with hardcoded neighbors:
Peer 0 (buyer) connected to peers [1]
Peer 1 (seller) connected to peers [0, 2]
Peer 2 (seller) connected to peers [1, 3]
Peer 3 (seller) connected to peers [2]
Peer 0 (buyer) with item None listening on port 5000...
Peer 1 (seller) with item salt listening on port 5001...
Peer 2 (seller) with item fish listening on port 5002...
Peer 3 (seller) with item boar listening on port 5003...
Network diameter is 3
Buyer 0 is initiating a lookup for boar with hopcount 2
21.10.2024 21:33:14.284 [0] Initiating lookup for boar
[0] Looking for boar with neighbor 1
[1] Forwarding lookup for boar to Peer 2
[2] Forwarding lookup for boar to Peer 3
[3] Sent reply for 977a093fa9eae64f76ee3896629398534928a226b532d...
[1] Sent reply to peer 0 for item boar with id 977a093fa9eae64f76ee...
[0] Deciding to buy item boar from seller 3
[3] Sold item to buyer 0. Remaining stock: 0
[0] Buyer is satisfied and stops buying.
[0] Shutting down peer.
All buyers have shut down. Shutting down sellers and exiting program.

```

Figure 8: BUY_PROBABILITY = 0

```

Network diameter is 3
Buyer 0 is initiating a lookup for salt with hopcount 2
21.10.2024 21:33:18.491 [0] Initiating lookup for salt
[0] Looking for salt with neighbor 1
[1] Forwarding lookup for salt to Peer 2
[2] Forwarding lookup for salt to Peer 3
[3] Hopcount 0 reached for request adefb091f07e01c6ealbe4cc...
[0] No response received for salt. Timing out and selecting...
[0] Searching for a new product: fish
21.10.2024 21:33:19.402 [0] Initiating lookup for fish
[0] Looking for fish with neighbor 1
[1] Forwarding lookup for fish to Peer 2
[2] Sent reply for 1dda9484775cedc56996f81d0dff28f78b32b1e6...
[1] Sent reply to peer 0 for item fish with id 1dda9484775ce...
[0] Deciding to buy item fish from seller 2
[2] Sold item to buyer 0. Remaining stock: 0
[2] Sold out of fish. Now selling fish
21.10.2024 21:33:19.493 [0] bought product fish from seller...
[0] Buyer is satisfied and stops buying.
[0] Shutting down peer.
All buyers have shut down. Shutting down sellers and exiting program.

```

Figure 9: BUY_PROBABILITY = 0.75

```

[2] Forwarding lookup for salt to Peer 3
[3] Hopcount 0 reached for request c9bcc32014ec5...
[0] No response received for salt. Timing out and selecting...
[0] Searching for a new product: boar
21.10.2024 21:33:32.479 [0] Initiating lookup for boar
[0] Looking for boar with neighbor 1
[1] Sent reply for bdb366a9a5b4e0009b0283fd30fa3...
[0] Deciding to buy item boar from seller 1
[1] Sold item to buyer 0. Remaining stock: 0
[1] Sold out of boar. Now selling boar
21.10.2024 21:33:32.481 [0] bought product boar
[0] Buyer decided to continue looking for another item
21.10.2024 21:33:32.481 [0] Initiating lookup for fish
[0] Looking for fish with neighbor 1
[1] Forwarding lookup for fish to Peer 2
[2] Forwarding lookup for fish to Peer 3
[3] Hopcount 0 reached for request e008484d6fb84...
[0] No response received for fish. Timing out and selecting...
[0] Searching for a new product: salt
21.10.2024 21:33:33.483 [0] Initiating lookup for salt
[0] Looking for salt with neighbor 1
[1] Forwarding lookup for salt to Peer 2
[2] Forwarding lookup for salt to Peer 3
[3] Hopcount 0 reached for request 5e1bd2715fdb1...
^CTraceback (most recent call last):
  File "/Users/jenish/Desktop/UMASS/courses/cs67...
    main(num_peers, num_buyers, num_sellers, nei...
  File "/Users/jenish/Desktop/UMASS/courses/cs67...
    time.sleep(1) # Sleep before checking again
KeyboardInterrupt
^CException ignored in: <module 'threading' from...
  python3.12/threading.py>
Traceback (most recent call last):
  File "/opt/homebrew/Cellar/python@3.12/3.12.2...
    utdown
      lock.acquire()
KeyboardInterrupt:

```

Figure 10: BUY_PROBABILITY = 1

Peer Shutdown

```

[1] No response received for boar. Timing out and selecting another item.
[1] Searching for a new product: fish
21.10.2024 20:57:37.067 [1] Initiating lookup for fish
[1] Looking for fish with neighbor 0
[1] Looking for fish with neighbor 2
[5] Sent reply to peer 4 for item boar with id c0eda1bcf0c06d3d4055e5eaafeba92523955567795f838a401ca93c76253966
[0] Sent reply for 10a1a84d75cd622d2d880675b8499cc6b029b5c31ef03598a6a08e18963d1799 to peer 1 for item fish
[1] Deciding to buy item fish from seller 0

```

Figure 11: Steps for TC.06: No Available Sellers

- **Expected Outcome:** The system handles the situation gracefully, notifying the buyer that the item is unavailable.
- **Result:** We can see that Peer 1 being a buyer and looking for boar cannot find a seller for it. In this case Peer 1 starts looking for a different item [in this case a fish].

2. TC.07: Maximum Hop Count Reached

- **Description:** Verify that the system correctly discards lookup messages when the maximum hop count is reached.

```

Network structure initialized with hardcoded neighbors:
Peer 0 (buyer) connected to peers [1]
Peer 1 (buyer) connected to peers [0, 2]
Peer 2 (seller) connected to peers [1]
Peer 0 (buyer) with item None listening on port 5000...
Peer 1 (buyer) with item None listening on port 5001...
Peer 2 (seller) with item fish listening on port 5002...
Network diameter is 2
Buyer 0 is initiating a lookup for salt with hopcount 1
21.10.2024 22:22:24.231 [0] Initiating lookup for salt
Buyer 1 is initiating a lookup for boar with hopcount 1
[0] Looking for salt with neighbor 1
21.10.2024 22:22:24.231 [1] Initiating lookup for boar
[1] Looking for boar with neighbor 0
[1] Looking for boar with neighbor 2
[2] Forwarding lookup for salt to Peer 2
[2] Hopcount 0 reached for request 197fae87f91f911b6830803045b5df43e2e80d57c5d12b35334fa0b2ecdc4f22. Discarding message.

```

Figure 12: Steps for TC.07: Maximum Hop Count Reached

- **Expected Outcome:** The lookup message is discarded once the hop count reaches zero, and no further propagation occurs.
- **Result:** Here we can see Peer 0 (Buyer) is initiated a look up for salt. Since there is only one seller (who is selling fish) the message reaches the max hop count and the message for buying salt by Peer 0 is discarded.

3. TC_08: Stock Depletion Handling

- **Description:** Ensure that the system correctly handles scenarios where a seller's stock depletes to zero.

```
[2] Deciding to buy item fish from seller 3
[2] Deciding to buy item boar from seller 0
[3] Sold item to buyer 2. Remaining stock: 0
[3] Sold out of fish. Now selling boar
[2] Deciding to buy item fish from seller 5
[0] Sold item to buyer 2. Remaining stock: 0
[0] Sold out of boar. Now selling fish
21.10.2024 20:57:37.219 [2] bought product fish from seller 3
[2] Buyer decided to continue looking for another item.
21.10.2024 20:57:37.219 [2] Initiating lookup for salt
[2] Looking for salt with neighbor 1
[5] Sold item to buyer 2. Remaining stock: 4
21.10.2024 20:57:37.219 [2] bought product boar from seller 0
[2] Buyer decided to continue looking for another item.
```

Figure 13: Steps for TC_08: Stock Depletion Handling

- **Expected Outcome:** After the purchase, Seller Peer updates its stock to zero and selects a new item to sell.
- **Result:** We can see that Peer 3 has no stock of fish left. It successfully completed transaction with Peer 2 Buyer. Since its stock is 0 it switches to selling another random item [in this case boar]

3.5 Error Handling Test Cases

1. TC_09: Failed Message Transmission

- **Description:** Test the system's ability to handle failed message transmissions gracefully.

```
[2] Sent reply to peer 1 for item fish with id b165f0298bdc7622c5802f5f853261ea89f639c27b380ca653edeead66d76dd
[2] Forwarding lookup for fish to Peer 3
[1] Error sending message to ('localhost', 5005): [Errno 9] Bad file descriptor
[5] Forwarding lookup for fish to Peer 0
```

Figure 14: Case: Failed message transition

```
[1] No response received for boar. Timing out and selecting another item.
[1] Searching for a new product: fish
21.10.2024 20:57:37.067 [1] Initiating lookup for fish
[1] Looking for fish with neighbor 0
[1] Looking for fish with neighbor 2
[5] Sent reply to peer 4 for item boar with id c8eda1bcf0c06d3d4055e5eaafeba92523955567799f838e401ca93c76253966
[0] Sent reply for 10a1a84d75cd42d2d2d880675b8499ccab027b5c31ef03598a6a08e18963d1799 to peer 1 for item fish
[1] Deciding to buy item fish from seller 0
```

Figure 15: Case: Seller not found

- **Expected Outcome:** The system detects the failure, logs an appropriate error message, and continues functioning without crashing.

2. TC_10: Concurrent Requests Handling

- **Description:** Ensure that the system can handle multiple concurrent lookup and purchase requests without performance degradation or data inconsistencies.

```
[1] Sold item to buyer 2 at 2024-10-21 20:39:42.658. Remaining stock: 4
[1] Sold item to buyer 3 at 2024-10-21 20:39:42.658. Remaining stock: 3
[1] Sold item to buyer 4 at 2024-10-21 20:39:42.658. Remaining stock: 2
[1] Sold item to buyer 5 at 2024-10-21 20:39:42.658. Remaining stock: 1
[1] Sold item to buyer 6 at 2024-10-21 20:39:42.659. Remaining stock: 0
.
-----
Ran 1 test in 0.003s

OK
```

Figure 16: Steps for TC_10: Concurrent Requests Handling

- **Expected Outcome:** All requests are processed correctly and consistent updates to transaction counts and stock levels.
- **Results:** Here we can see that a seller Peer 1 is handling concurrent buy requests from 5 buyers and successfully utilizing the allotted stock.

4 Evaluation

4.1 Test Configuration Constants

The following constants are used to configure the behavior of buyers and sellers within the P2P network:

```
1 BUY_PROBABILITY = 1      # Probability that a buyer will continue buying after a successful
   purchase
2 SELLER_STOCK = 5         # Each seller starts with 5 items
3 TIMEOUT = 0.1            # Timeout duration in seconds
```

Listing 1: Test Configuration Constants

4.2 Test Outputs

The following sections present the results of the concurrency tests conducted with different values of `MAX_TRANSACTIONS` for a buyer. Each test simulates single buyer purchasing stock and records the average RTT for the buyer along with the network connections.

4.3 Test with `MAX_TRANSACTIONS = 20`

Configuration:

- `MAX_TRANSACTIONS = 20`
- Each seller starts with 5 items.

Output:

```
RTT for each buyer:
Buyer 4 average RTT: 0.0010 seconds
Peer 0 (seller) connected to peers [1, 5, 3]
Peer 1 (seller) connected to peers [0, 2, 5]
```


Peer 2 (seller) connected to peers [1, 3, 4]
Peer 3 (seller) connected to peers [2, 4, 0]
Peer 4 (buyer) connected to peers [3, 5, 2]
Peer 5 (seller) connected to peers [4, 0, 1]

4.4 Test with MAX_TRANSACTIONS = 40

Configuration:

- MAX_TRANSACTIONS = 40
- Each seller starts with 5 items.

Output:

RTT for each buyer:
Buyer 4 average RTT: 0.0261 seconds
Peer 0 (seller) connected to peers [1, 5, 2]
Peer 1 (seller) connected to peers [0, 2, 3]
Peer 2 (seller) connected to peers [1, 3, 0]
Peer 3 (seller) connected to peers [2, 4, 1]
Peer 4 (buyer) connected to peers [3, 5]
Peer 5 (seller) connected to peers [4, 0]

4.5 Test with MAX_TRANSACTIONS = 100

Configuration:

- MAX_TRANSACTIONS = 100
- Each seller starts with 5 items.

Output:

RTT for each buyer:
Buyer 4 average RTT: 0.0018 seconds
Peer 0 (seller) connected to peers [1, 5, 4]
Peer 1 (seller) connected to peers [0, 2, 5]
Peer 2 (seller) connected to peers [1, 3]
Peer 3 (seller) connected to peers [2, 4]
Peer 4 (buyer) connected to peers [3, 5, 0]
Peer 5 (seller) connected to peers [4, 0, 1]

4.6 Analysis of Test Results

The concurrency tests were conducted to evaluate how the system's RTT performance varies with different numbers of transactions a buyer can perform before shutdown. The tests were run with three different configurations of MAX_TRANSACTIONS: 40, 20, and 100.

4.7 Observations

- **RTT Performance:**
 - With MAX_TRANSACTIONS = 40, the average RTT per buyer was measured at **0.0261 seconds**.
 - Reducing MAX_TRANSACTIONS to **20** significantly decreased the average RTT to **0.0010 seconds**.
 - Increasing MAX_TRANSACTIONS to **100** resulted in a slight increase in average RTT to **0.0018 seconds**.

4.8 Average RTT vs Number of Buyers

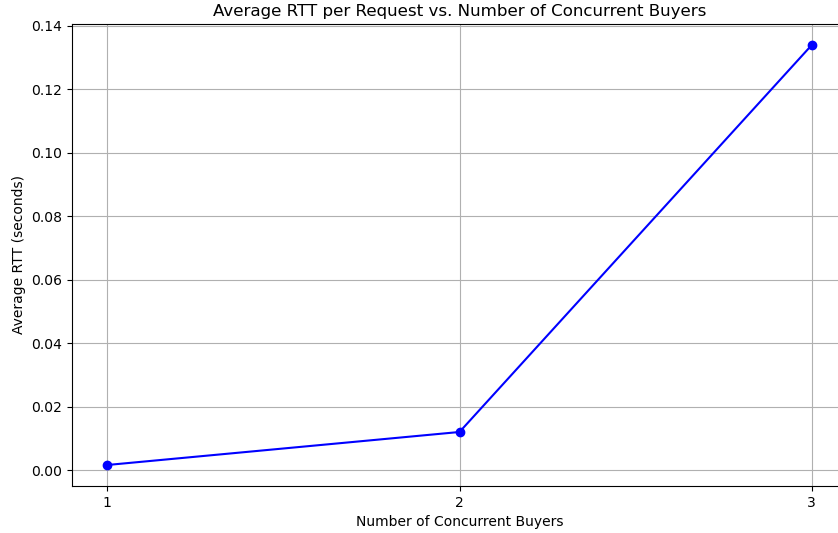


Figure 17: Average RTT vs number of Buyers when number of peers is 4

As the number of buyers increases, we observe a corresponding increase in the average RTT. This trend is expected because higher concurrency levels, managed through threads, lead to increased contention for network resources, resulting in greater latency.

5 Limitations and Design Trade-off

In this section, we describe the limitation(L) and design tradeoffs (T) we considered for our system.

1. (T1) UDP for Communication: We use UDP as the transport layer protocol for efficiency. UDP is suitable for running peers locally, however for large systems limitations of UDP over TCP might bottleneck the entire system.
2. (T2) Simple Cache Eviction Policy: We use a simple (FIFO) cache eviction process for a node to store the received requests. This does not account for access patterns, potentially evicting frequently.
3. (L1) Static Role Assignment: The buyers and sellers in our system are statically assigned. This does not necessarily depict a real world scenario where a peer who was a buyer could also be a seller. Furthermore, our system does not support new peers joining the network once initialized.
4. (L2) Fault Tolerance: If a peer fails or goes offline unexpectedly our system cannot handle advanced fault tolerance mechanisms. Pending requests are lost, and the system relies on timeouts and retries for recovery.