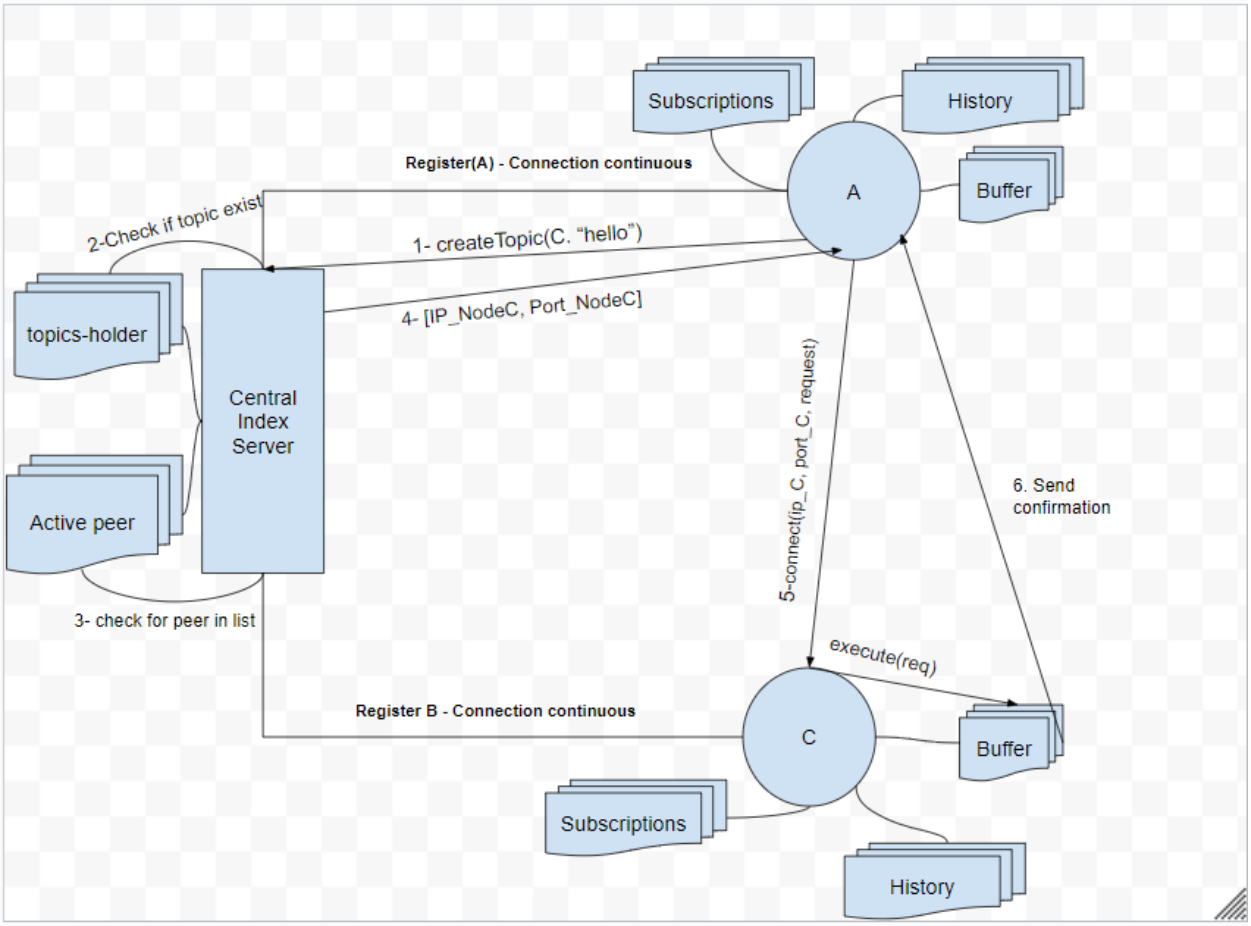
# 1. Design of the System and Discussion

The attached picture is the overview of my system design which I used. There are 2 peer nodes **A, B,** and a central index server. The central index server hosts a dictionary for topic-holder nodes and a dictionary of active peers. Peer- [ip, port]. It uses it to manage the changes that occur and based on the peer node operation keeps the up-to-date information about the system. Each peer node has its Buffer, where it stores the messages, History to keep track of the reading history for each peer node that subscribed to a specific topic, and a list of subscriptions of nodes.

The connection between any peer node and the Index server is **continuous**, but the connection between nodes is 1 session only. Nodes after executing the request close the connection between each other.




Let's see scenario described in the attached picture. Once server start running, node A first register itself with server. Server assigns port address to the node A. After it stores information of A in Active peer dictionary. It also initializes **topics** dictionary to empty as there's no topic yet.

Peer node C also does the same.

Peer A requests to create the topic **"hello"** in Peer C. It sends the request to the index server. The index server checks if the topic **"hello"** already exists or not. If not, it checks if there is **peer C** in the active peer dictionary. If there is no such topic and peer C in the list of active peer dictionaries, then it fetches the credentials of the peer node C and sends it back to the requestor which is peer A. After peer A

**connects to** peer C. Peer C deserializes the packet and checks the request. After it executes the request, it sends back the reply as confirmation and closes the connection.

**Tradeoffs of design**

1. One of the benefits of this design system is it uses a very simple structure that does not cause overhead to the CPU.

a. One of the benefits of using Buffer dictionary for index servers to store the topic and their holder is that access time is $O$ (1). Even if we have 1 million topics to hold in the buffer, it still takes constant time to find the elements.

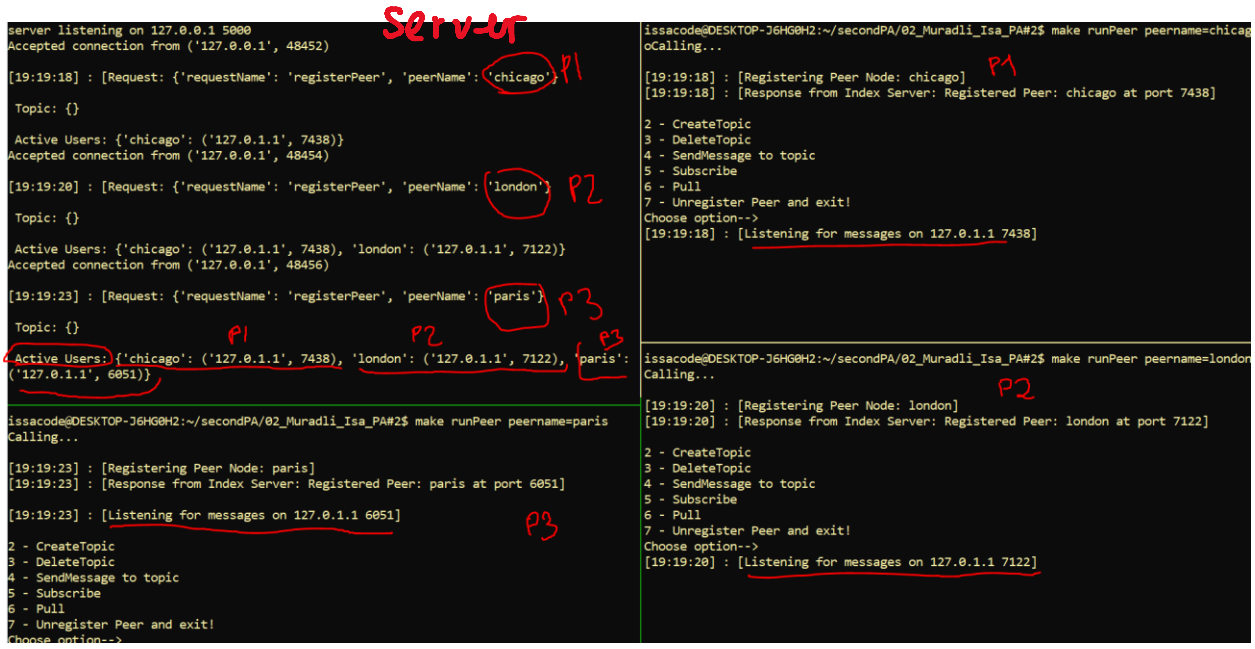
b. In rare situations (e.g., many collisions), access time can degrade to $O$ ($n$), where $n$ n is the number of items in the dictionary

Since I'm building this system locally, there's not such issues regarding the security and the privileges to the data. But what if this system is not local, in that case, other peers directly accessing my topics is not very secure. Even if there are protocols, other peer nodes that are not reliable can corrupt my buffer. Therefore, we need to also make sure that authorization to the files is highly considered.

For this system, APIs are very sufficient as amount of the data being transferred are negligible, but if there's many peer nodes and each of them require a quite bit of bandwidth of network for file transfer, it will be painful.

## 2. Deploying 3 peers and index server.

As can be seen in the screenshot, 3 peers named **Chicago, London, and Paris** have connected to the index server. As soon as we run the peer file and specify peername=" name", it calls the **redigsterPeer** and sends the request to the index server. The index server going to store it in the list of active peers. Meanwhile, peers also start listening for the incoming connection while being connected to the index server.

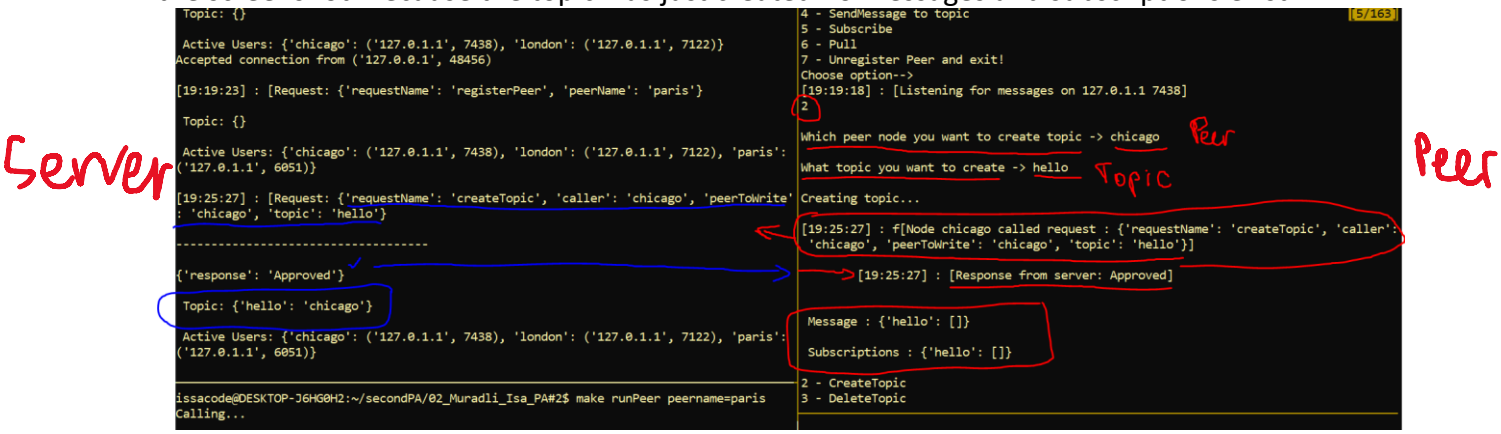


## a. Create topic API.

Peer nodes can create a topic on their own side or choose to host it in the peer they want. For that we **Option 2**: Create Topic.

**Let's first create a topic on the local side**. As you can see once we choose option 2, the terminal prompts to ask for PeerName where to create, and topic name.

After it builds the request and sends it to the index server. The Index server receives the request and deserializes it. After it check for the Peer node and if the topic exists already in its buffer. It fulfills the request and creates the response **Approved.** The response was sent back to the peer **Chicago** and the topic was created locally in peer Chicago, it's showing on the right side of the screenshot. Because the topic was just created no messages and subscriptions exist.




**Let's create a topic now in the other node.**

This time **Peer London** will try to create a topic **welcome** in **Peer Chicago.**

4 - First Peer London sends the request to the index server. The index server checks if a specified peer node exists in the active directory.

5 - If verifies for the forward and sends back to peer London the connection credentials of the peer Chicago.

6. London receives it, sees the forwarding message, and takes the credentials of peer Chicago to connect.

7. It establishes the connection with peer Chicago

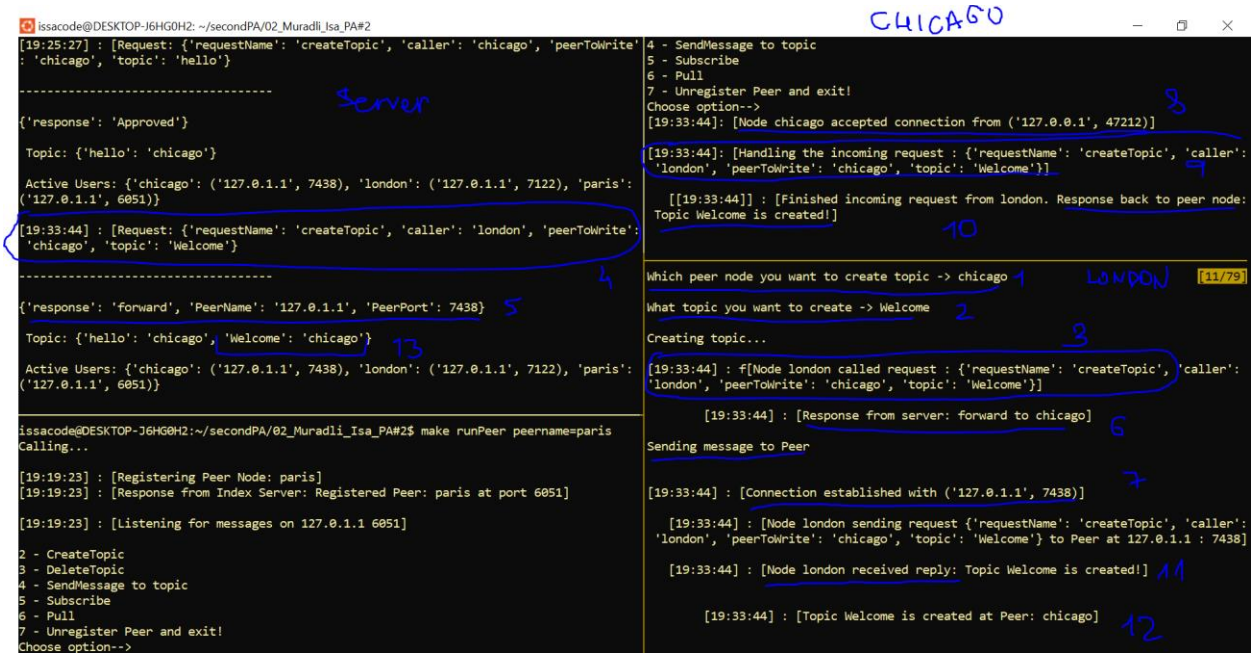
8. Peer Chicago accepts the connection from London.

9. Deserializes the request and fulfills it locally.

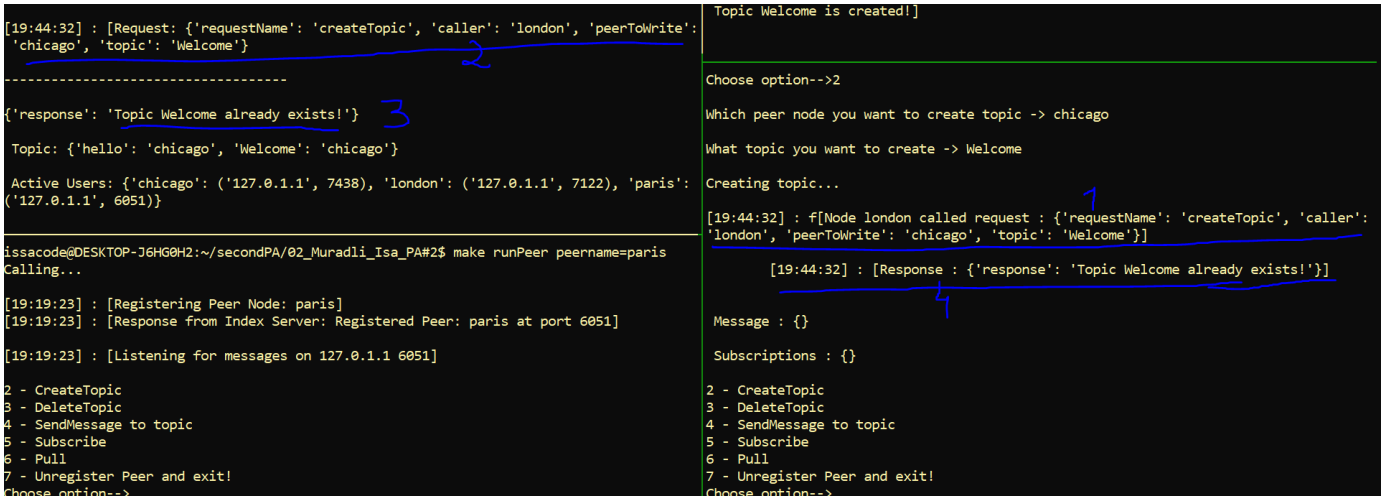
10. It sends back an acknowledgement message "Topic Welcome is created!"

11/12 – Peer London receives back acknowledgment about the creation of a topic in peer Chicago

13 – The server also updates the list of topics and assigns the topic Welcome to Chicago in its Topic buffer.

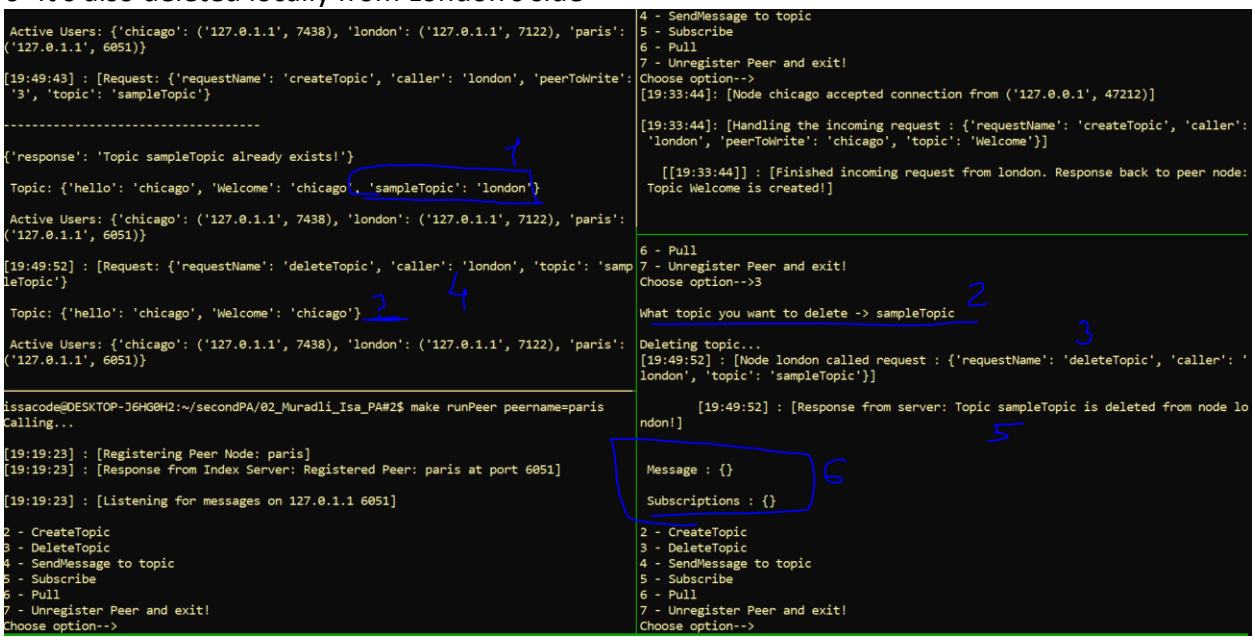



If peer London tries to create the same topic in Chicago or even locally, the Index server will check in its topics buffer if there is such a topic. Therefore, the node gets back a response as “Topic Welcome already created”



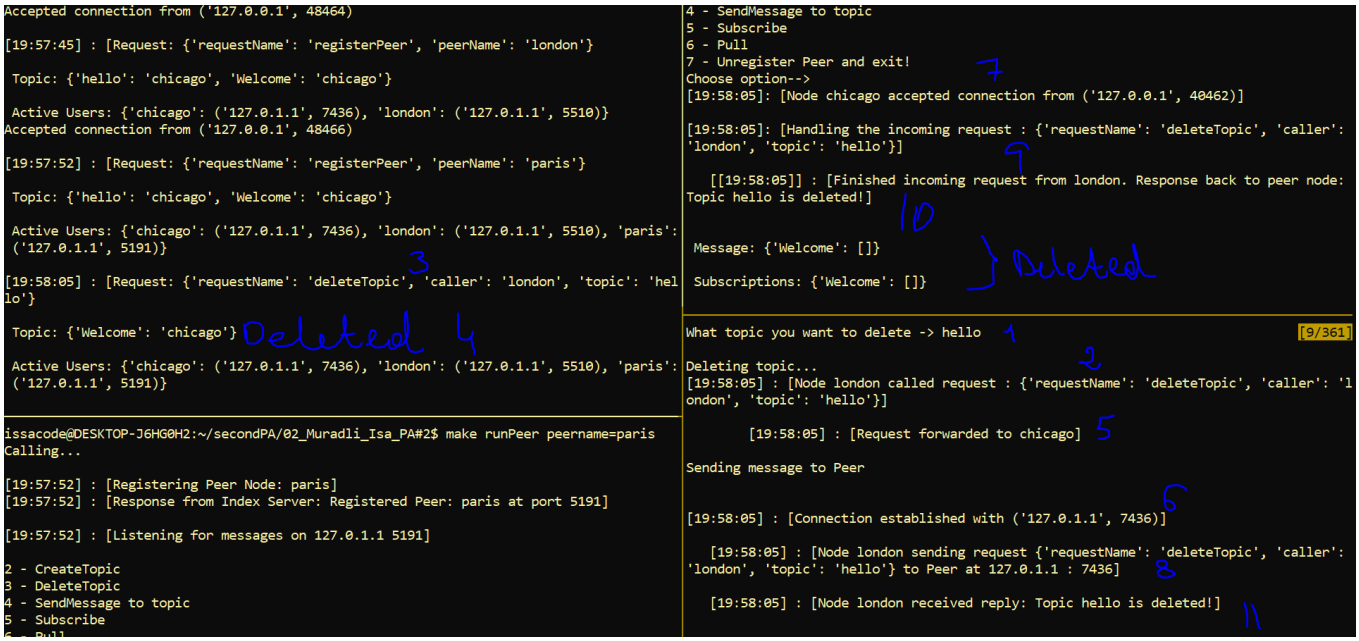
## b. Delete topic API.

1- Peer London tries to delete the topic "**sampleTopic"** that it hosts. It exists on the server side as can be seen in point
2/3 - London creates a request to send to the index server to delete it.
4 – The server deserializes the request and deletes it from the topics buffer. As it's shown in the**?** Mark put in the screenshot
5- London receives acknowledgement of deletion
6- It's also deleted locally from London's side




**If London tries to delete a topic from Chicago, it can also be done.**

1. London specifies a topic name to delete.
2. Creates the request and send it to the index server.
3. The server receives, handles the request and send forwarding response for London
4. Server updates buffer by deleting topic "**hello".**
5. London receives **forwarding** and gets the credentials of Chicago
6. Establishes the connection with Chicago
7. Chicago accepts the connections
8. London sends the request for execution
9. Chicago accomplishes the request and deletes the topic locally
10. Sends the **Topic hello is deleted** back to peer London
11. London receives the message and closes the connection with Chicago


Accepted connection from ('127.0.0.1', 48464)
[19:57:45] : [Request: {'requestName': 'registerPeer', 'peerName': 'london'}
Topic: {'hello': 'chicago', 'Welcome': 'chicago'}
Active Users: {'chicago': ('127.0.1.1', 7436), 'london': ('127.0.1.1', 5510)}
Accepted connection from ('127.0.0.1', 48466)
[19:57:52] : [Request: {'requestName': 'registerPeer', 'peerName': 'paris'}
Topic: {'hello': 'chicago', 'Welcome': 'chicago'}
Active Users: {'chicago': ('127.0.1.1', 7436), 'london': ('127.0.1.1', 5510), 'paris': ('127.0.1.1', 5191)}
[19:58:05] : [Request: {'requestName': 'deleteTopic', 'caller': 'london', 'topic': 'hello'}
3
Topic: {'Welcome': 'chicago'}
Deleted
4
Active Users: {'chicago': ('127.0.1.1', 7436), 'london': ('127.0.1.1', 5510), 'paris': ('127.0.1.1', 5191)}
issacode@DESKTOP-J6HG0H2:~/secondPA/02_Muradli_Isa_PA#2$ make runPeer peername=paris
Calling...
[19:57:52] : [Registering Peer Node: paris]
[19:57:52] : [Response from Index Server: Registered Peer: paris at port 5191]
[19:57:52] : [Listening for messages on 127.0.1.1 5191]
2 - CreateTopic
3 - DeleteTopic
4 - SendMessage to topic
5 - Subscribe
6 - Pull
4 - SendMessage to topic
5 - Subscribe
6 - Pull
7 - Unregister Peer and exit!
7
Choose option-->
[19:58:05]: [Node chicago accepted connection from ('127.0.0.1', 40462)]
[19:58:05]: [Handling the incoming request : {'requestName': 'deleteTopic', 'caller': 'london', 'topic': 'hello'}]
9
[[19:58:05]] : [Finished incoming request from london. Response back to peer node: Topic hello is deleted!]
10
Message: {'Welcome': []}
Subscriptions: {'Welcome': []}
} Deleted
What topic you want to delete -> hello
1
[9/361]
Deleting topic...
2
[19:58:05] : [Node london called request : {'requestName': 'deleteTopic', 'caller': 'london', 'topic': 'hello'}]
[19:58:05] : [Request forwarded to chicago]
5
Sending message to Peer
[19:58:05] : [Connection established with ('127.0.1.1', 7436)]
6
[19:58:05] : [Node london sending request {'requestName': 'deleteTopic', 'caller': 'london', 'topic': 'hello'} to Peer at 127.0.1.1 : 7436]
8
[19:58:05] : [Node london received reply: Topic hello is deleted!]
11

## c. Send API.

**First peer Paris is sending a message to its own topic personalTopic**

1/2 It gives the topic name and the message to send

3 – A message is being sent to the topic

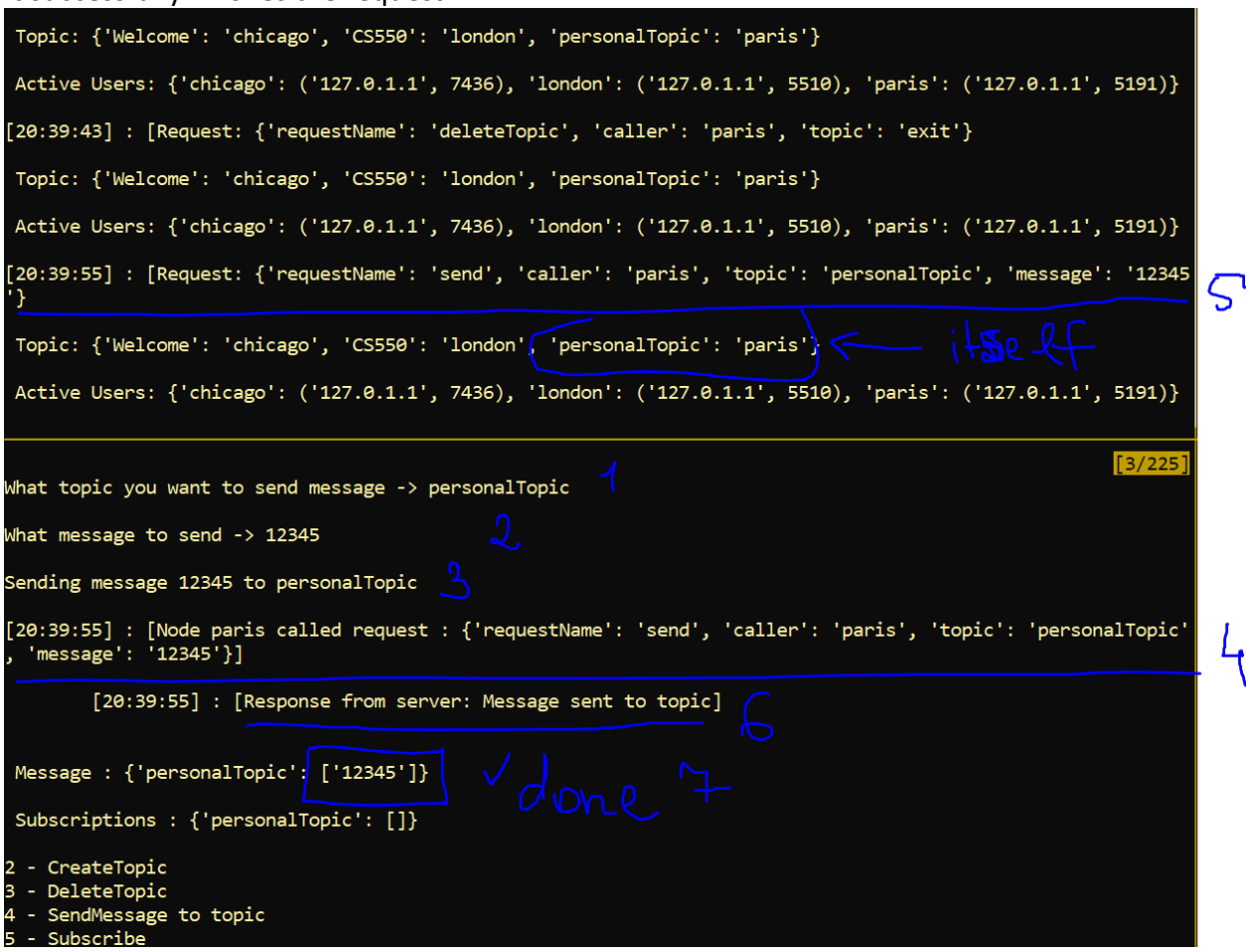
4 – It creates the request and sends it to the index server.

5 – The index server checks if there's such a topic and who is the holder of that topic. Since Paris itself is the holder of a topic, it does not forward it.

6 – Paris gets from the server that the message is sent to the topic.

7- It gets added to the message Buffer

It successfully finishes the request



**In this scenario, peer Paris is sending the message HelloFromParis to peer node London.**

1 – Create the request to send the message **HelloFromParis** to topic **CS550.**

2/3– The Index server deserializes the request and checks for who is the holder of the topic **CS550,** it is London

4 – Peer Paris gets response from the index server **Forward to Node London** and also provides the address of the London peer node

5. Paris establishes the connection with the peer node London

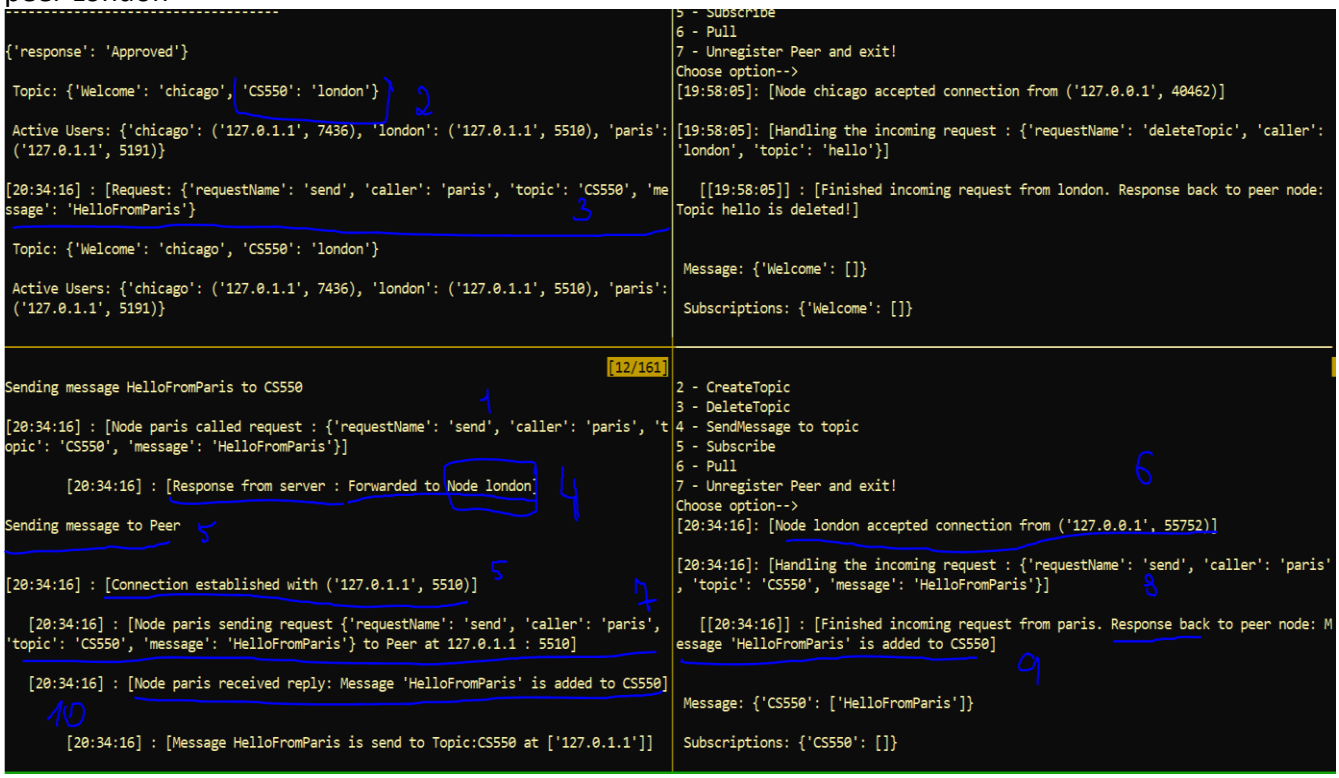
6. London accepts the incoming connection from Paris

7. Paris send the same request with the message and topic name to the London

8. London deserializes the request and handles the request.

9. It adds the message to the topic CS550 and sends back the acknowledgment

10 – Paris receives a response from London that **HelloFromParis** has been added to the **CS550** in peer London

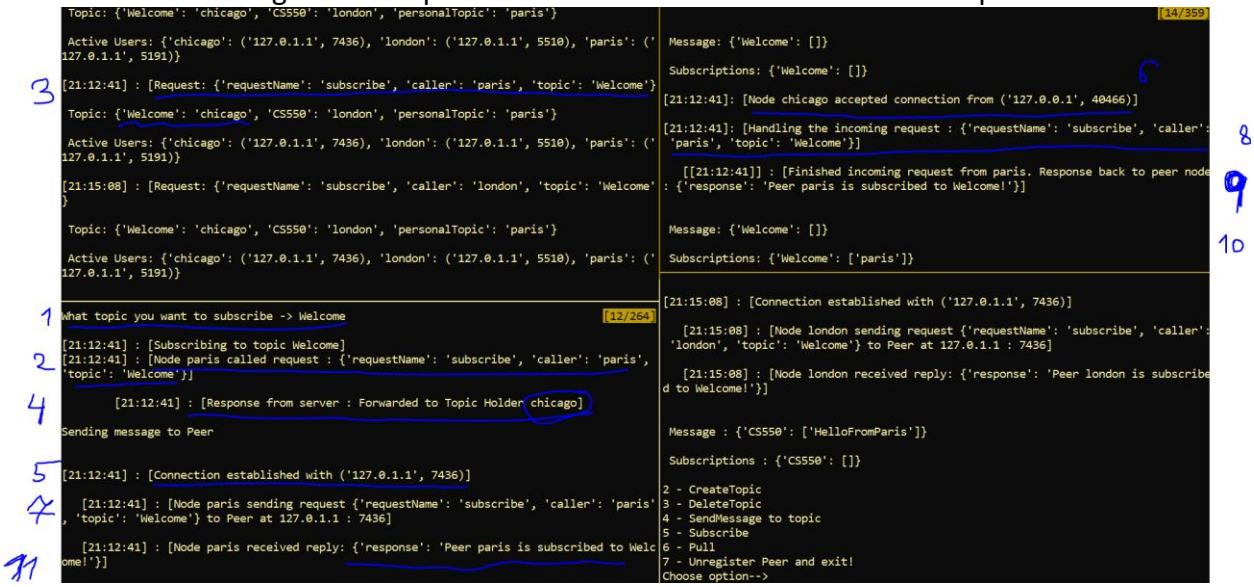


## d. Subscribe API.

1. Peer node Paris chooses to subscribe to the topic Welcome
2. It creates a request and sends to the index server for the holder of that topic
3. Index server deserializes the request and find the holder of the topic (Chicago) and sends it to Paris
4. Paris receives that forwarding response from Index server
5. Establishes a connection with the peer Chicago
6. Peer node Chicago accepts the connection
7. Peer Paris forwards the request to the Chicago node

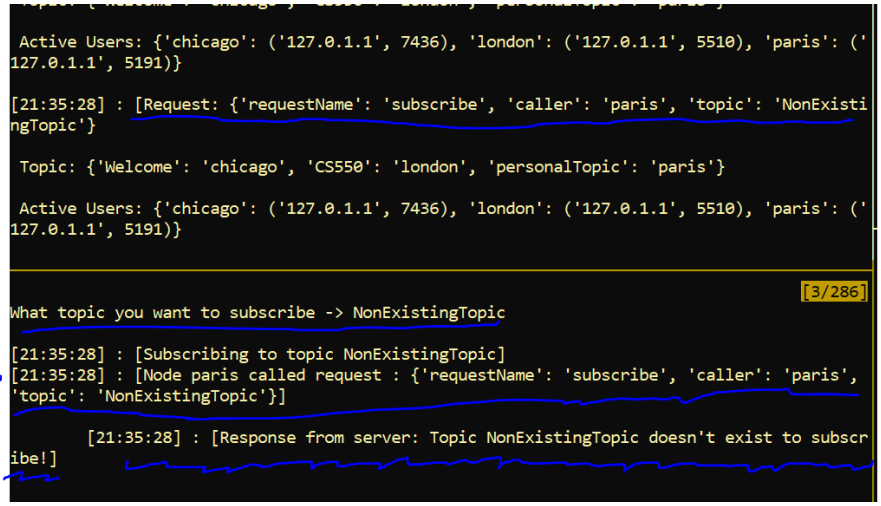
8/9/10 Chicago node fulfills the request and adds the Paris as subscribed to the topic welcome

11. Peer node Paris gets the response that it has been subscribed to the topic




**Request to subscribe to a topic that doesn't exist.**

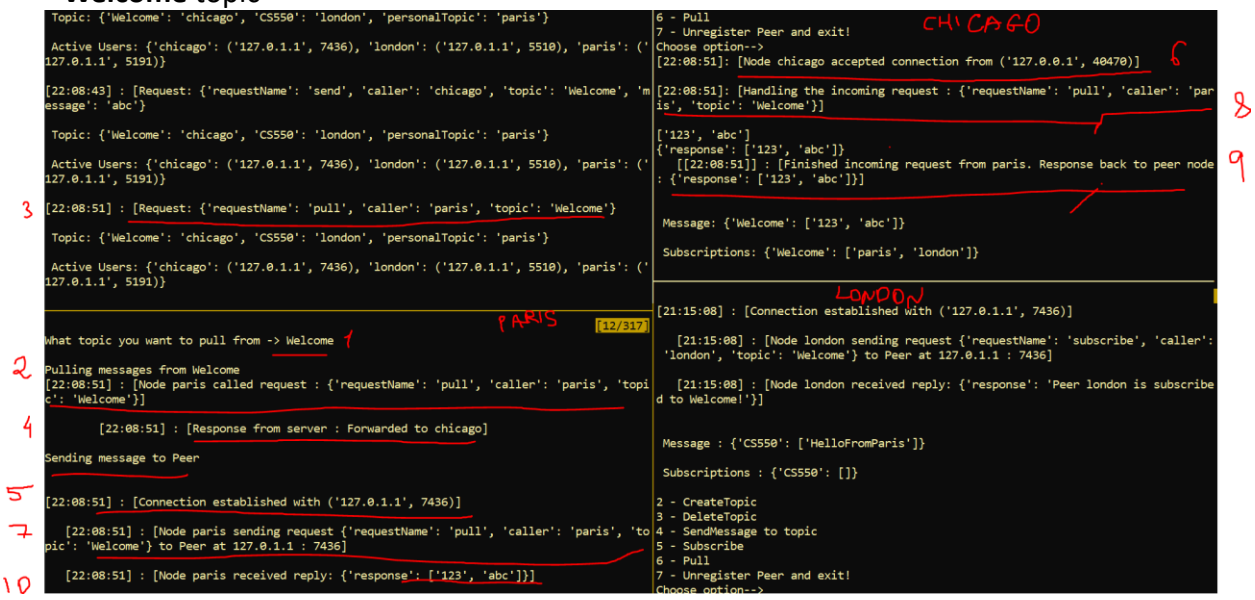
When Paris sends a request to subscribe to **NonExistingTopic,** the Index server checks for that topic and sends back the response as "**Topic NonExistingTopic doesn't exist to subscribe!"**



## e. Pull API.

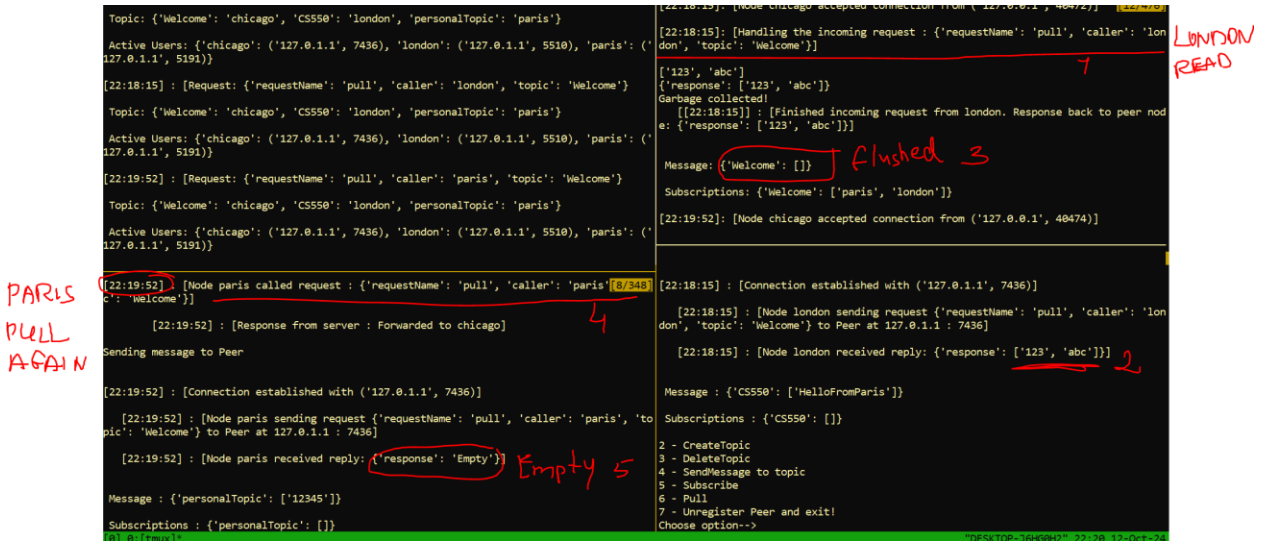
1. Paris request to pull from topic **Welcome**
2. Create request to send to index server
3. Index server deserializes the request and find the holder of the topic (Chicago)
4. Paris gets back response and needs to forward the request to the peer Chicago
5. Paris creates connection with Chicago
6. Chicago accepts the incoming connection from Paris
7. Paris forwards the request to Chicago
8. Chicago handles the incoming request and fetches the unread message from **Welcome** topic
9. Send back the response [**123, abc]**
10. Paris receives the message **[123, abc]**

    **Note:** The garbage collection hasn't been called because London who subscribed to the Welcome hasn't read the message. That's why Chicago doesn't flush the message in the **Welcome** topic




After London pulls from the topic Welcome, the node Chicago calls garbage collections and all messages are gone. Then, Paris pull again from Welcome topic and gets empty response as we expected.

1. London send request to Chicago to pull from Welcome
2. Since all subscribers have read the messages, Chicago flushes all messages and garbage collected
3. London receives **123, abc** which it called previously
4. Paris pull again from topic Welcome and gets empty message. It works correct Great!



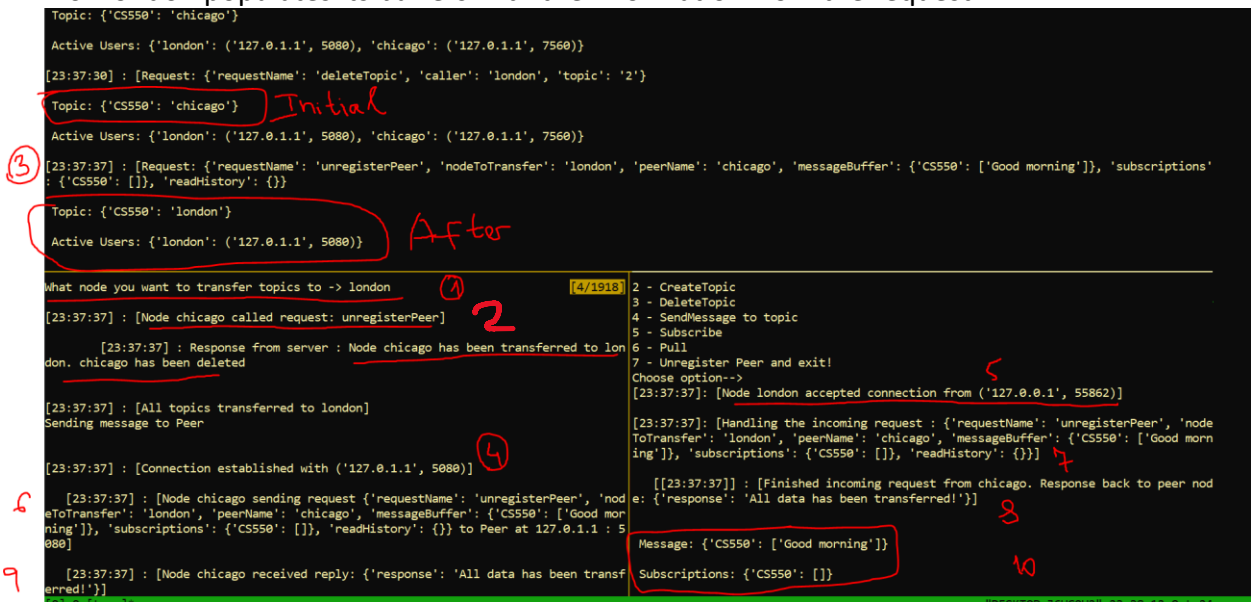

## f. Unregister API.

When the peer node unregisters, it's asked 2 questions whether it wants to delete everything (1) or transfer topics to other peers(2). In the picture below, it chooses to delete everything. As you can see, Initial and after. The topic of Chicago hello was in the directory of the index server, but after it got deleted and Chicago was removed from list of active peers


If it chooses to transfer, then the user is prompted to specify the peer name it wants to transfer all the topic information. As you can see all data has been transferred to peer London

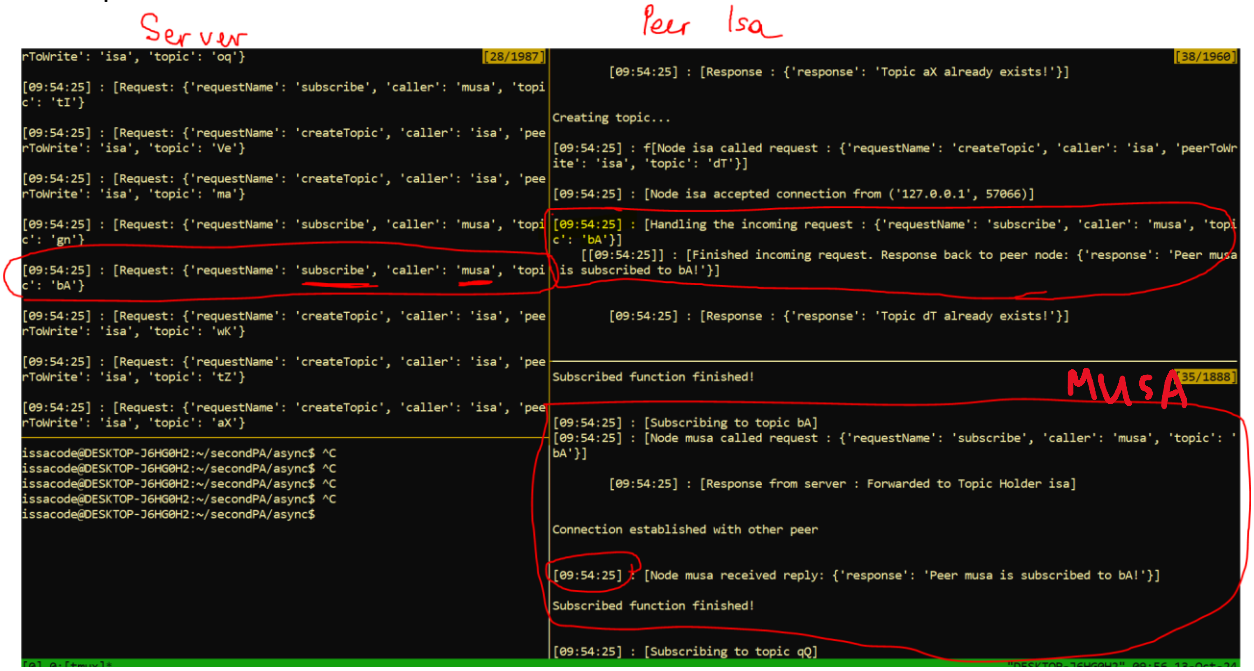
1- Chicago chooses London as a peer to transfer all info
2- It creates the request to the index server.

3- The Index server receives the request and finds the credentials of the London
4- Chicago establishes the connection with London
5- London accepts the incoming connection
6- Chicago send all information to London
7- London handles the request
8- London send back the confirmation of the request
9- Chicago receives the confirmation
10- London populates its buffers with the information from the request



# Ensuring multiple peer node can simultaneously publish and subscribe to a topic

Here peer node **musa** request to subscribe to the topic of node **Isa.** As it can be seen from the screen shot, right bottom is the node **musa is** requesting to subscribe to topic bA, why node **isa** is constantly creating topic in his end. While creating the topic, **peer Isa** can also execute the request from the **peed Musa** and subscribe him to one of his topic. It worked!




Let's consider when 2 nodes(Paris, London) subscribing simultaneously to topics, while 1 node(Chicago) is publishing topics. When one of 2 nodes that want to subscribe to a topic, server will provide the connection credentials of the node that holds that topic - C. After that node P or L will establish connection with node C and node C will receive the same request and execute it while it constantly publishing the topics.
**Red** - As you can observer, node C sending continuous create topic requests to the server. Then, it receives node P requests simultaneously subscribe to a topic, once server receives it and check in the buffer if node C is the holder, it sends back the credentials -09:57:31

09:57:31 - Immediately node P forwards the request to node C
09:57:31 – Node C receives it (top right ) and fulfills the request. In the response it's stating that Peer Paris is subscribed to Ik!
After Peer P receives the same response and subscribe function finished printed out.
Same scenario applies to the blue mentioned circles.

**Note: Peer Chicago simultaneously handles the publishing the topic and handling the incoming subscribe requests from the peers Paris and London.**

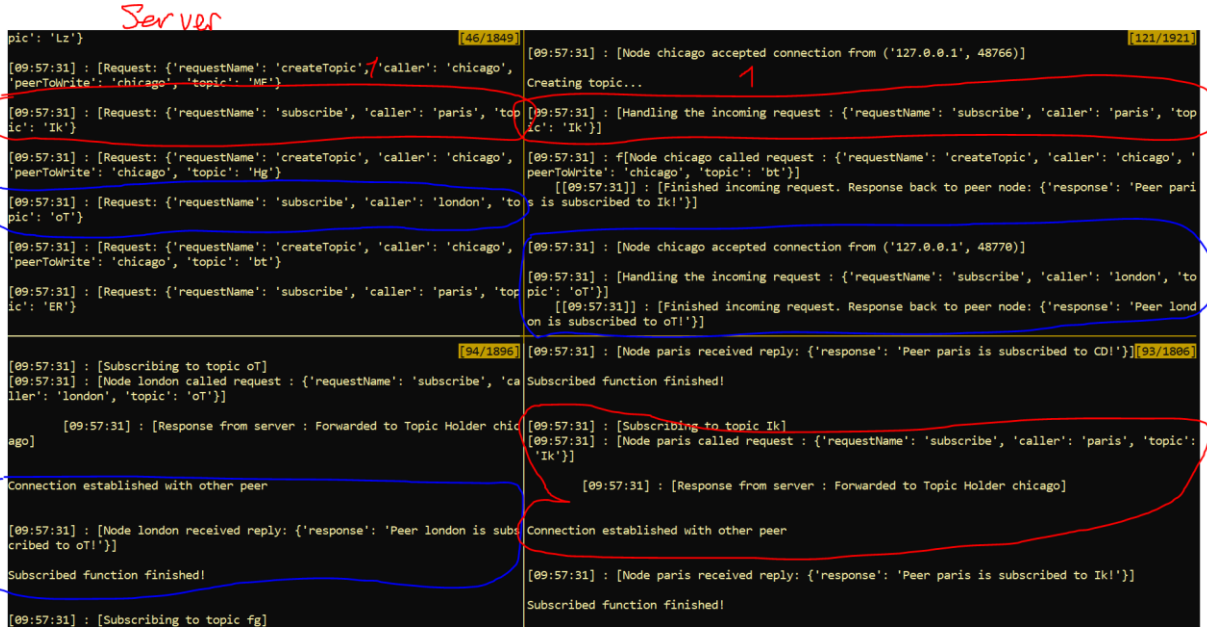
Server

Server

CHICAGO publ

London subs

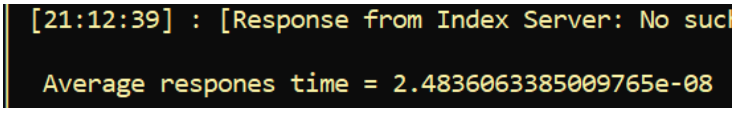
PARIS Subs

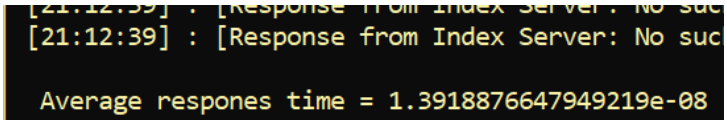


# Measuring the average response time

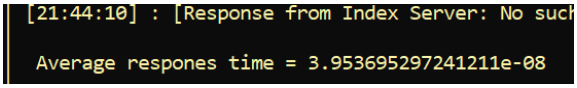
**Note:** each node makes 100,000 requests**. Please be aware** when benchmarking average response time, server first needs to populate the topics buffer with data. So, give it some time. You will be ready to call benchmark for queries once it shows Server is running on (IP, Port). Every node submits 100,000 requests. We created dummy data in the serverForBenchmarking 1 million random topic. Simultaneously multiple nodes query the index server for random topic names and doing so we calculate the average response time. We can see that as the number of nodes increase, the average response time also increase. It's also due to the fact that we use the dictionary which gives us *O(1)* access time and server is local. If server was somewhere outside of local network and topics were stored in different data structure, they it could be very long time to get the response. Average response time will not be sweet.

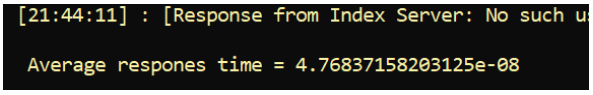
**2 Peer nodes: Average = 1.937747e-08**

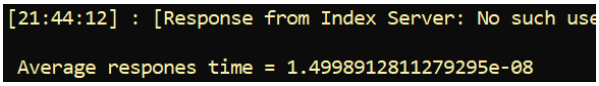


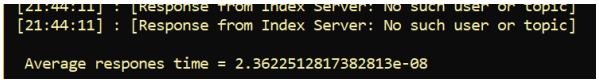




**4 Peer nodes: Average= 3.14605274e-08**

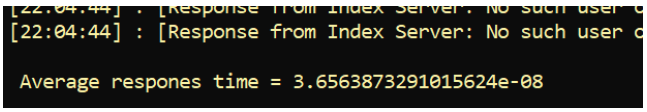


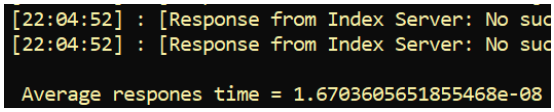



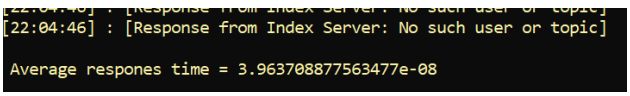



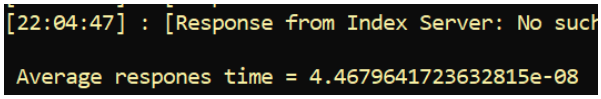



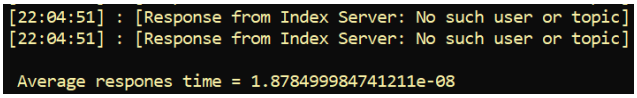

**8 Peer nodes: Average= 3.10340522e-08**

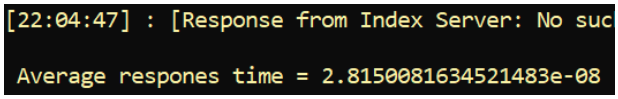


```
[22:04:50] : [Response from Index Server: icFy]

Average respones time = 4.36854362487793e-08
```

```
[22:04:50] : [Response from Index Server: No such user or topic]

Average respones time = 2.0067691802978514e-08
```

**10 Peer nodes: Average= 5.46002421e-08**

```
[22:30:28] : [Response from Index Server: No such u

 Average respones time = 7.421016693115234e-08
```

```
[22:30:38] : [Response from Index Server: No suc

 Average respones time = 5.64122200012207e-08
```

```
[22:30:14] : [Response from Index Server: No such u

 Average respones time = 7.144212722778321e-08
```

```
[22:30:32] : [Response from Index Server: No suc

 Average respones time = 5.2685737609863284e-08
```

```
[22:30:42] : [Response from Index Server: No such

 Average respones time = 2.8297901153564454e-08
```

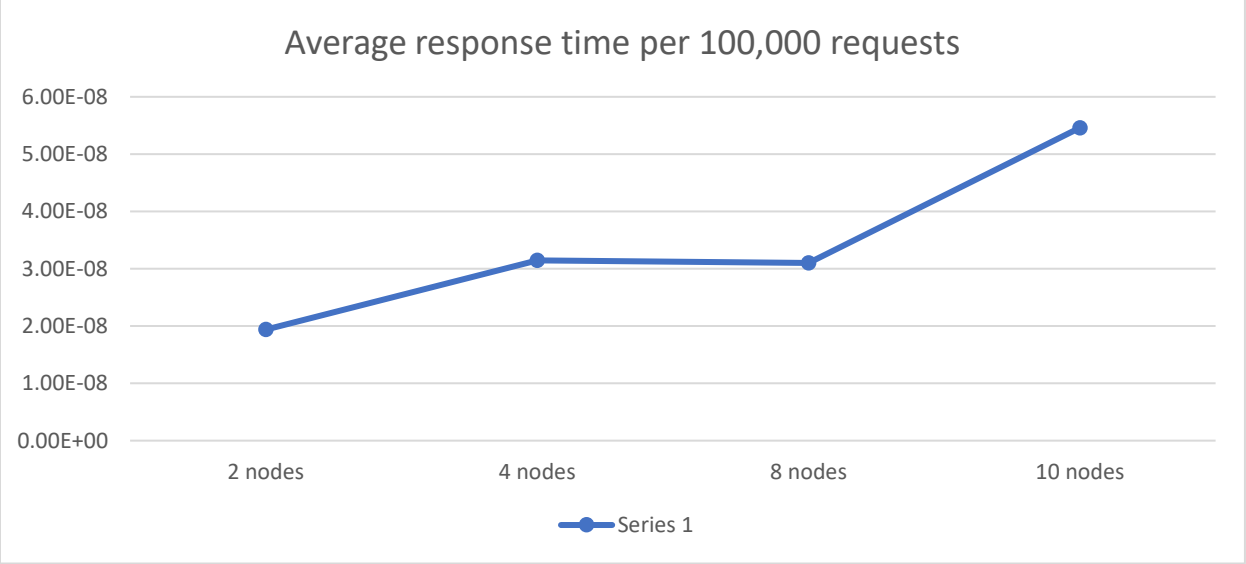
```
[22:30:37] : [Response from Index Server: No s

 Average respones time = 7.03597068786621e-08
```

```
[22:30:42] : [Response from Index Server: No suc

 Average respones time = 2.183675765991211e-08
```

```
[22:30:42] : [Response from Index Server: No such

 Average respones time = 2.6478767395019533e-08
```

```
[22:30:31] : [Response from Index Server: No su

Average respones time = 9.16290283203125e-08
```

```
[22:30:42] : [Response from Index Server: No such us

 Average respones time = 5.264759063720703e-08
```



# Benchmarking

| Requests | Create | Delete | Send | Subscribe | Pull |
|---|---|---|---|---|---|
| 1 Node | 54720 | 87323 | 61516 | 70646 | 101631 |
| 2 Node | 74243 | 59127 | 67330 | 54371 | 118180 |
| 3 Node | 61050 | 64534 | 71953 | 59352 | 62650 |
| 4 Node | 46841 | 44737 | 40552 | 46853 | 44724 |
| 5 Node | 33562 | 38909 | 21247 | 24609 | 31356 |
| 6 Node | 31956 | 47095 | 25591 | 29428 | 45787 |
| 7 Node | 35600 | 43421 | 36086 | 37194 | 42308 |
| 8 Node | 33190 | 30162 | 31691 | 27542 | 36311 |

| Throughput | Create | Delete | Send | Subscribe | Pull |
|---|---|---|---|---|---|
| 1 Node | 182.4 | 291.07 | 205.05 | 235.48 | 338.77 |
| 2 Node | 247 | 196 | 224 | 181 | 394 |
| 3 Node | 203.5 | 215 | 239 | 198 | 208.5 |
| 4 Node | 156.3 | 148.7 | 135 | 155.6 | 149.3 |
| 5 Node | 111.85 | 129.14 | 70.432 | 81.95 | 113.97 |
| 6 Node | 107.96 | 154.8 | 82.73 | 98.61 | 150.9 |
| 7 Node | 116.7 | 144.74 | 119.28 | 123.04 | 141.6 |
| 8 Node | 109.90 | 101.23 | 105.606 | 94.486 | 121.79 |

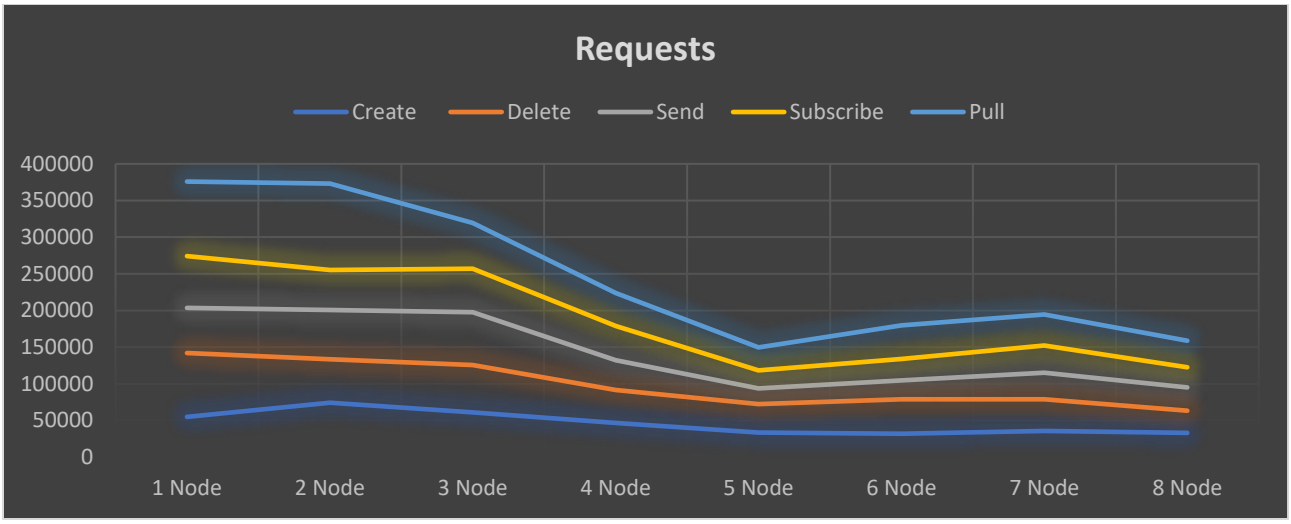
**Note that latency has been rounded at 5$^{th}$ digit after the decimal point**

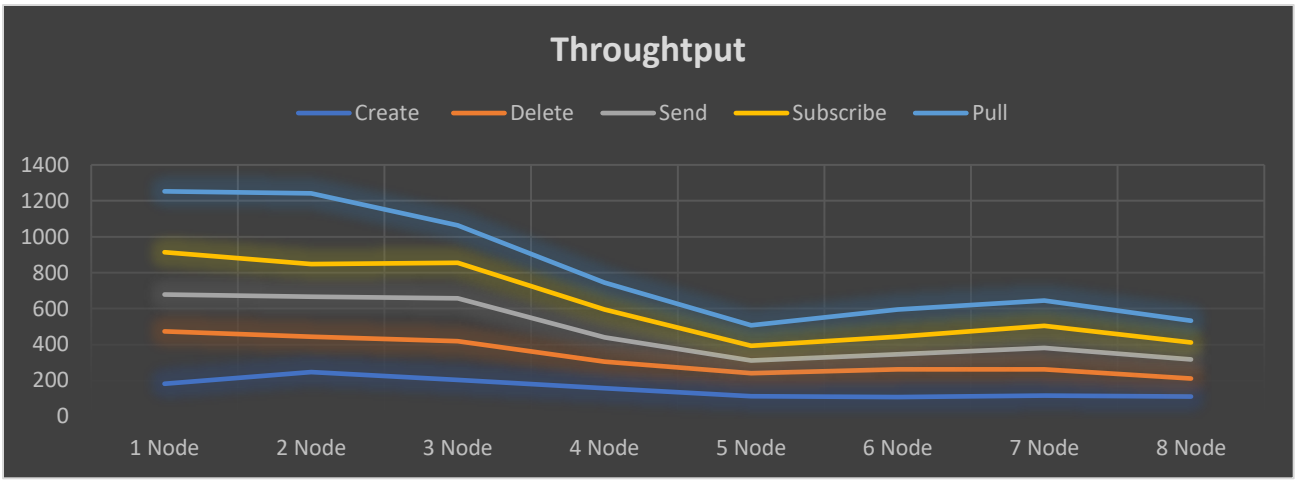
| Latency | Create | Delete | Send | Subscribe | Pull |
|---|---|---|---|---|---|
| 1 Node | 0.00548 | 0.00343 | 0.00487 | 0.00587 | 0.00295 |
| 2 Node | 0.00403 | 0.00507 | 0.00445 | 0.00551 | 0.00253 |
| 3 Node | 0.00409 | 0.00463 | 0.00415 | 0.00504 | 0.00478 |
| 4 Node | 0.00635 | 0.00673 | 0.00743 | 0.00649 | 0.00671 |
| 5 Node | 0.00886 | 0.00769 | 0.01418 | 0.01223 | 0.00972 |
| 6 Node | 0.00953 | 0.00639 | 0.01191 | 0.01802 | 0.00665 |
| 7 Node | 0.00864 | 0.00685 | 0.00831 | 0.00803 | 0.00709 |
| 8 Node | 0.00881 | 0.10320 | 0.00983 | 0.01076 | 0.00829 |

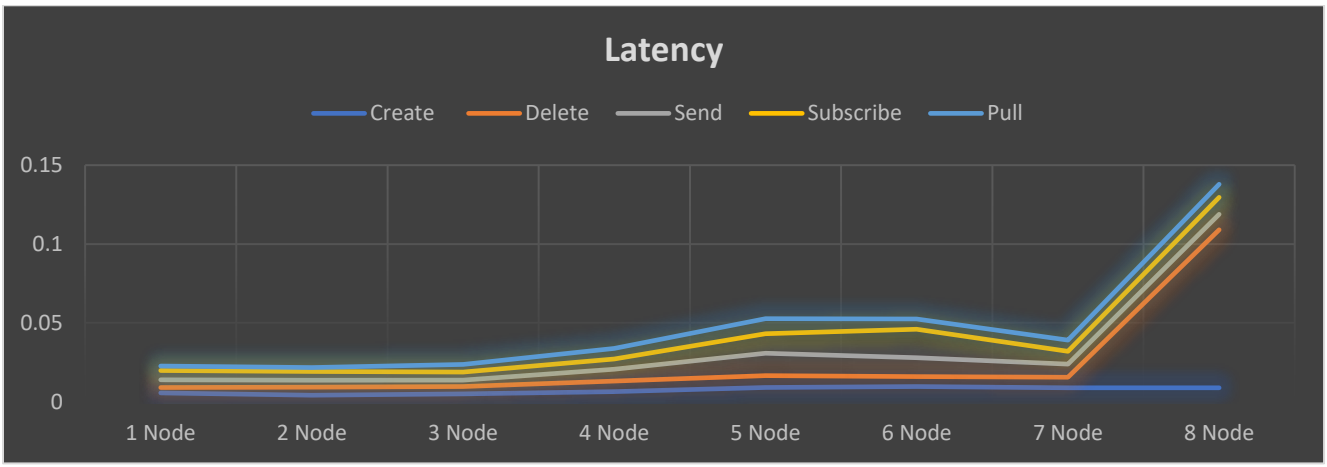
First when we have up to 3 nodes, most of the API requests remains the almost the same, but starting from 4 nodes concurrently sending APIs to the index server. Due to the fact that it puts overload on the server to switch and fulfill the requests, it's not surprising that after one point server will decrease in productivity even if it supports concurrency. Having to switch every time will cost more time than actual time needed to finish the request.

Similarly, as the number of requests decrease as number of nodes increase, throughput will be affected as well. It shows that starting from 5 nodes, it declines.

Additionally, as the throughput decrease, it will consequently lead to the more latency in the system. All incoming requests should be addressed so that nodes might need to wait for their turn for server to take care. Therefore, the more requests in the system, the more you might need to wait.



# 1 Peer node

## a. Create Topic

```
Creating topic...

[10:37:08] : f[Node isa called request : {'requestName': 'createTopic', 'caller': 'isa', 'peerToWrite':
 'isa', 'topic': 'jSQx'}]

       [10:37:08] : [Response from server: Approved]


 Number of request : 54720, timePassed: 5 mins, throughput : 182.4 Latency = 0.005479630535980415
```

## b. Delete Topic

```
Deleting topic...
[10:52:08] : [Node isa called request : {'requestName': 'deleteTopic', 'caller': 'isa', 'topic': 'xYss'}]

       [10:52:08] : [Response from server: Topic xYss does not exist]


 Number of request : 87323, timePassed: 5 mins, throughput : 291.07666666666665 Latency = 0.003433894072475
```

## c. Send

```
Sending message XuVr to ngAQ

[11:05:14] : [Node isa called request : {'requestName': 'send', 'caller': 'isa', 'topic': 'ngAQ', 'message': 'X
uVr'}]

       [11:05:14] : [Response from server: Topic ngAQ does not exist]


 Number of request : 61516, timePassed: 5 mins, throughput : 205.05333333333334 Latency = 0.004874404656173148
```

## d. Subscribe

```
Subscribed function finished!


 Number of request : 70646, timePassed: 5 mins, throughput : 235.48666666666668 Latency = 0.004244785992
```

## e. Pull

```
Pulling messages from WfMo
[11:23:10] : [Node isa called request : {'requestName': 'pull', 'caller': 'isa', 'topic': 'WfMo'}]

       [11:23:10] : [Response from server: Topic WfMo does not exist]


 Number of request : 101631, timePassed: 5 mins, throughput : 338.77 Latency = 0.0029503138682245708
```

## 2 Peer Nodes

**For the benchmarking of 2 or more peers, we will consider their average of nodes connected for the graph!**

### a. Create Topic

```
Creating topic...
[11:32:21] : f[Node Peer1 called request : {'requestName': 'createTopic', 'caller': 'Peer1', 'peerToWrite': 'Peer1', 'topic': 'fYYz'}]
        [11:32:21] : [Response from server: Approved]

 Number of request : 74618, timePassed: 5 mins, throughput : 248.72666666666666 Latency = 0.004018766877670961
```

```
Creating topic...
[11:32:21] : f[Node Peer2 called request : {'requestName': 'createTopic', 'caller': 'Peer2', 'peerToWrite': 'Peer2', 'topic': 'SaOM'}]
        [11:32:21] : [Response from server: Approved]

 Number of request : 73869, timePassed: 5 mins, throughput : 246.23 Latency = 0.00405907566691441
```

### b. Delete Topic

```
Deleting topic...
[12:05:03] : [Node Peer1 called request : {'requestName': 'deleteTopic', 'caller': 'Peer1', 'topic': 'fTGy'}]
        [12:05:03] : [Response from server: Topic fTGy does not exist]

 Number of request : 59349, timePassed: 5 mins, throughput : 197.83 Latency = 0.005053744589031676
```

```
Deleting topic...
[12:05:06] : [Node Peer2 called request : {'requestName': 'deleteTopic', 'caller': 'Peer2', 'topic': 'aTaq'}]
        [12:05:06] : [Response from server: Topic aTaq does not exist]

 Number of request : 58905, timePassed: 5 mins, throughput : 196.35 Latency = 0.005091185719925806
```

### c. Send

```
Sending message XDBC to Kjqq
[12:14:48] : [Node Peer2 called request : {'requestName': 'send', 'caller': 'Peer2', 'topic': 'Kjqq', 'message': 'XDBC'}]
        [12:14:48] : [Response from server: Topic Kjqq does not exist]

 Number of request : 67361, timePassed: 5 mins, throughput : 224.53666666666666 Latency = 0.004451442505631243
```

```
Sending message peSD to XLvF
[12:14:49] : [Node Peer1 called request : {'requestName': 'send', 'caller': 'Peer1', 'topic': 'XLvF', 'message': 'peSD'}]
        [12:14:49] : [Response from server: Topic XLvF does not exist]

 Number of request : 67299, timePassed: 5 mins, throughput : 224.33 Latency = 0.0044558169762091565
```

### d. Subscribe

```
Subscribed function finished!

 Number of request : 54117, timePassed: 5 mins, throughput : 180.39 Latency = 0.005541720098580107
```

```
Subscribed function finished!

[12:24:18] : [Subscribing to topic yGlc]
[12:24:18] : You can't subscribe to your own topic

 Number of request : 54626, timePassed: 5 mins, throughput : 182.08666666666667 Latency = 0.005489815338997063
```

### e. Pull

```
Pulling messages from MNwZ
[12:47:17] : [Node Peer1 called request : {'requestName': 'pull', 'caller': 'Peer1', 'topic': 'MNwZ'}]

        [12:47:17] : [Response from server: Topic MNwZ does not exist]


 Number of request : 118374, timePassed: 5 mins, throughput : 394.58 Latency = 0.0025326527526347334
```

```
Pulling messages from JMbX
[12:47:18] : [Node Peer2 called request : {'requestName': 'pull', 'caller': 'Peer2', 'topic': 'JMbX'}]

       [12:47:18] : [Response from server: Topic JMbX does not exist]


 Number of request : 117987, timePassed: 5 mins, throughput : 393.29 Latency = 0.002540888888119744
[second_se0:python3*
```

## 3 Peer Nodes

### a. Create Topic

```
Creating topic...

[12:57:18] : f[Node Peer1 called request

        [12:57:18] : [Response from serv


 Number of request : 60973,
timePassed: 5 mins,
throughput : 203.24333333333334
Latency = 0.004918255324306854
```

```
Creating topic...

[12:57:18] : f[Node Peer2 called req

        [12:57:18] : [Response from


 Number of request : 61036,
timePassed: 5 mins,
throughput : 203.45333333333335
Latency = 0.004913256594765834
```

```
Creating topic...

[12:57:18] : f[Node Peer3 called r

        [12:57:18] : [Response fro


 Number of request : 61167,
timePassed: 5 mins,
throughput : 203.89
Latency = 0.004902525440961766
```

**b. Delete Topic**

```
Deleting topic...
[13:07:41] : [Node Peer1 called

        [13:07:41] : [Response f

 Number of request : 64620,
timePassed: 5 mins,
throughput : 215.4
Latency = 0.00464117810365328
```

```
Deleting topic...
[13:07:44] : [Node Peer2 called

        [13:07:44] : [Response f

 Number of request : 64787,
timePassed: 5 mins,
throughput : 215.95666666666668
Latency = 0.004629096075599655
```

```
Deleting topic...
[13:07:45] : [Node Peer3 called

        [13:07:45] : [Response fr

 Number of request : 64445,
timePassed: 5 mins,
throughput : 214.81666666666666
Latency = 0.00465378781639121
```

**c. Send**

```
Sending message qmKO to Mfet
[13:19:09] : [Node Peer1 called request : {'requestName': 'send', 'caller': 'Peer1', 'topic': 'Mfet', 'message': 'qmK

        [13:19:09] : [Response from server: Topic Mfet does not exist]

 Number of request : 72022, timePassed: 5 mins, throughput : 240.07333333333332 Latency = 0.004163706009014098
```

```
Sending message GupN to LHSH
[13:19:09] : [Node Peer2 called request : {'requestName': 'send', 'caller': 'Peer2', 'topic': 'LHSH', 'message'

        [13:19:09] : [Response from server: Topic LHSH does not exist]

 Number of request : 72377, timePassed: 5 mins, throughput : 241.25666666666666 Latency = 0.004143136733883771
```

```
Sending message ORul to HvVy
[13:19:11] : [Node Peer3 called request : {'requestName': 'send', 'caller': 'Peer3', 'topic': 'HvVy', 'message': 'ORul'}]

        [13:19:11] : [Response from server: Topic HvVy does not exist]

 Number of request : 71673, timePassed: 5 mins, throughput : 238.91 Latency = 0.00418435032167153
```

**d. Subscribe**

```
[13:30:16] : [Node Peer1 called request

        [13:30:16] : [Response from ser

Subscribed function finished!

 Number of request : 59593,
timePassed: 5 mins,
throughput : 198.64333333333335
Latency = 0.005032438960059456
```

```
[13:30:17] : [Subscribing to top
[13:30:17] : [Node Peer2 called

        [13:30:17] : [Response f

Subscribed function finished!

 Number of request : 59444,
timePassed: 5 mins,
throughput : 198.14666666666668
Latency = 0.005044957440519195
```

```
Subscribed function finished!

 Number of request : 59268,
timePassed: 5 mins,
throughput : 197.56
Latency = 0.005059838130051383
```

**e. Pull**

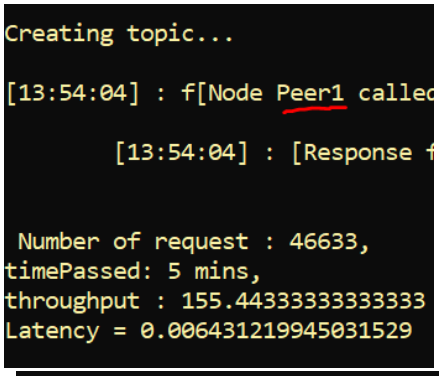
```
Pulling messages from jpgg
[13:42:42] : [Node Peer1 called rec

        [13:42:42] : [Response from

 Number of request : 62624,
timePassed: 5 mins,
throughput : 208.74666666666667
Latency = 0.004788338441323673
```

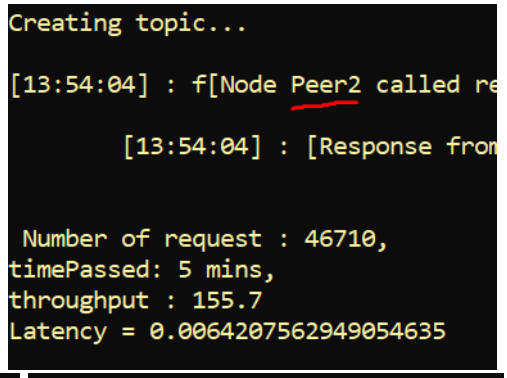
```
Pulling messages from sffi
[13:42:41] : [Node Peer2 called

        [13:42:41] : [Response fr

 Number of request : 62593,
timePassed: 5 mins,
throughput : 208.64333333333335
Latency = 0.004790626038353611
```

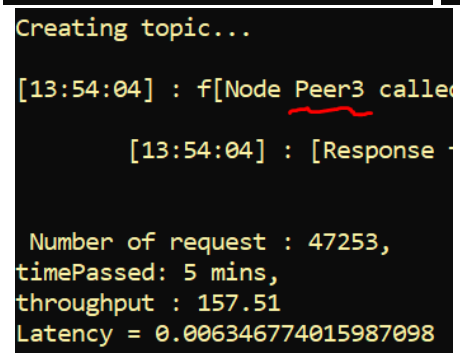
```
Pulling messages from VJys
[13:42:43] : [Node Peer3 called

        [13:42:43] : [Response f

 Number of request : 62699,
timePassed: 5 mins,
throughput : 208.99666666666667
Latency = 0.004782405975197725
```

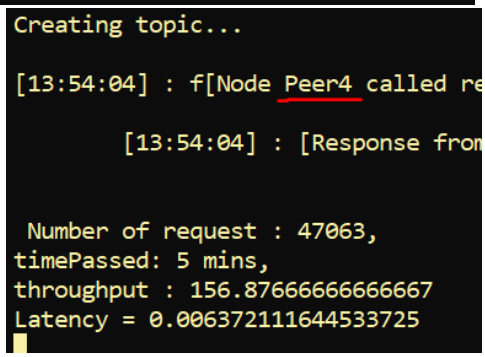
## 4 Peer Nodes

### a. Create Topic

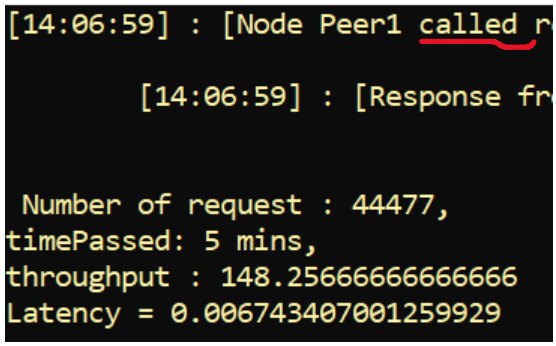


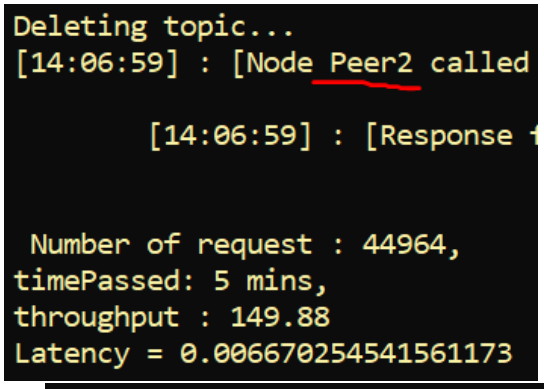



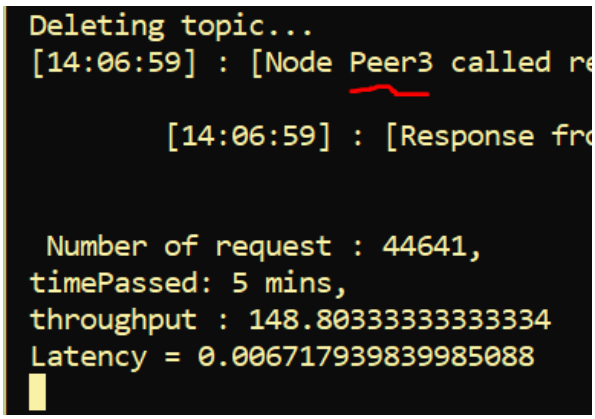



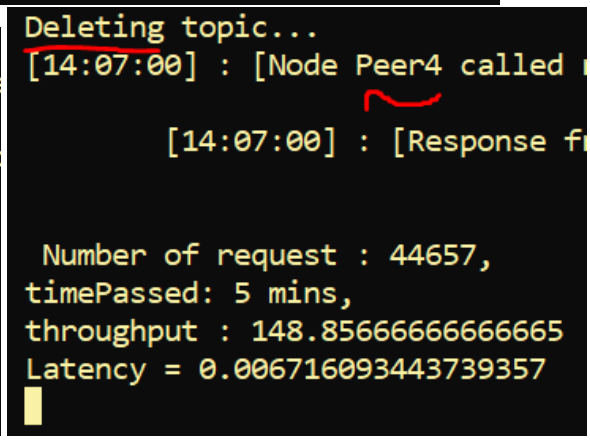




### b. Delete Topic

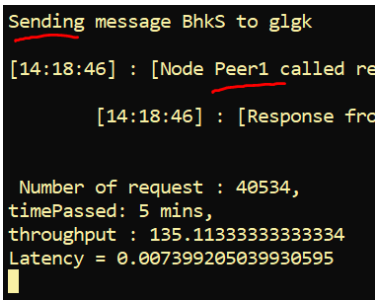


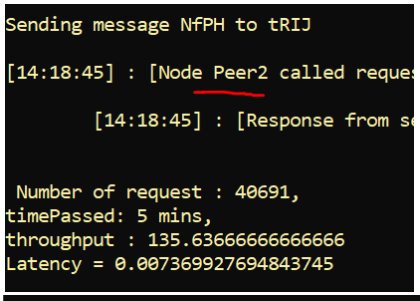



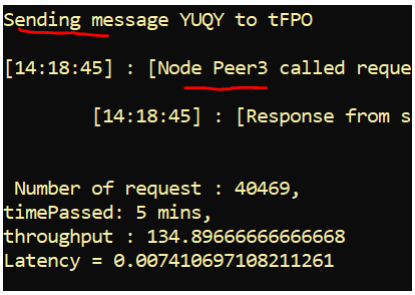



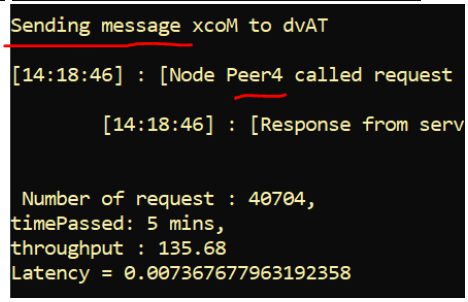



## c. Send

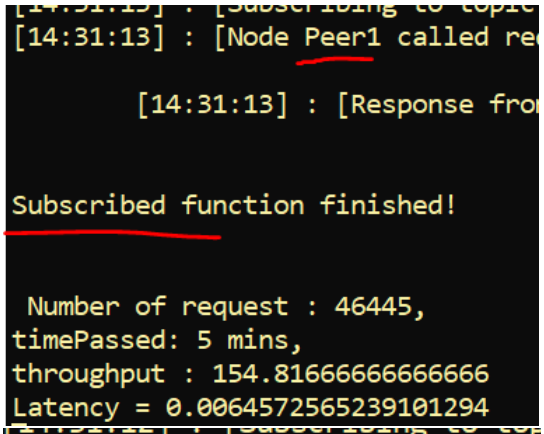


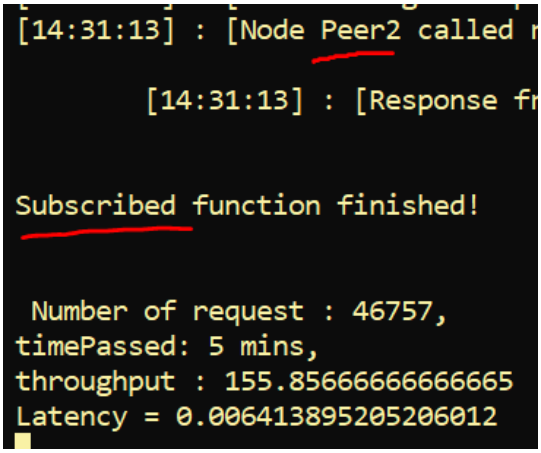



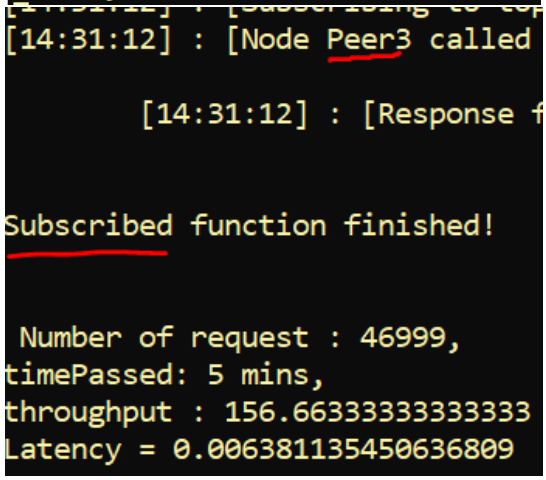



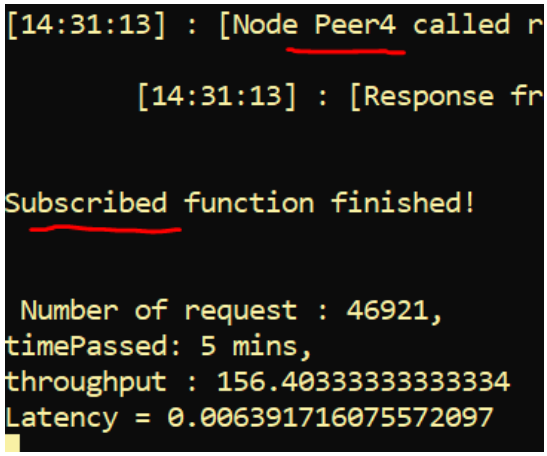




## d. Subscribe

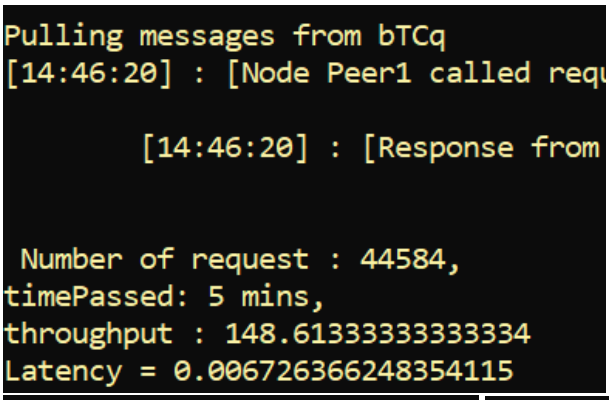


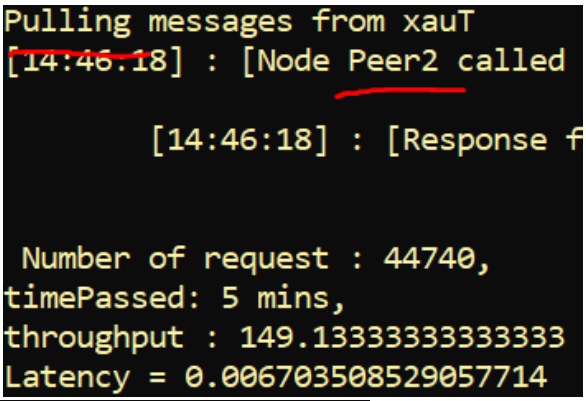



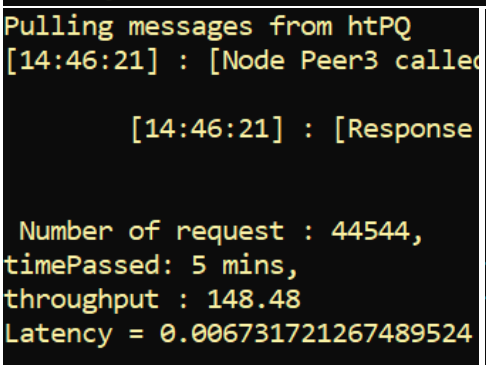



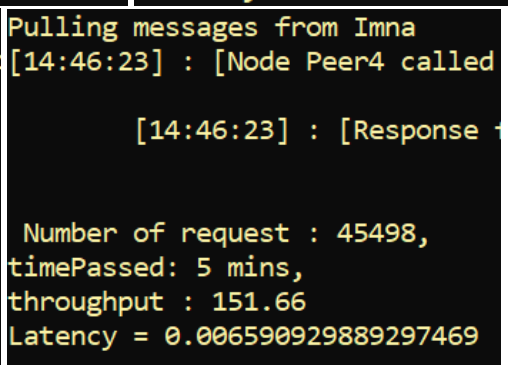



## e. Pull



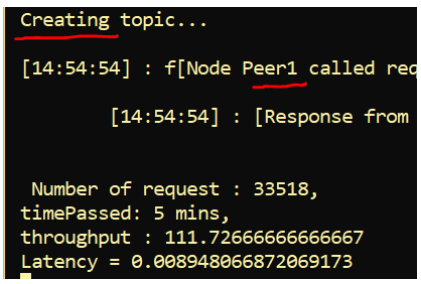



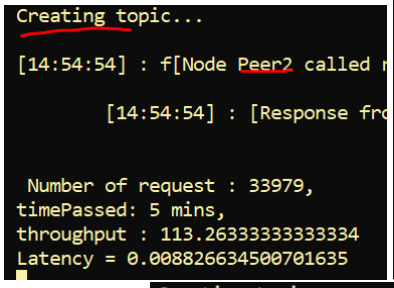



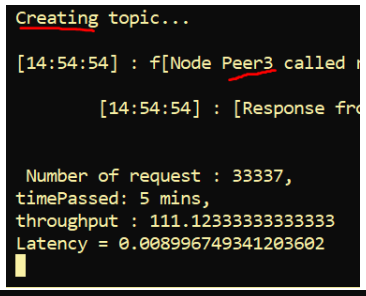



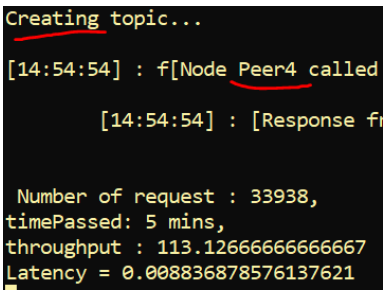

## 5 Peer Nodes

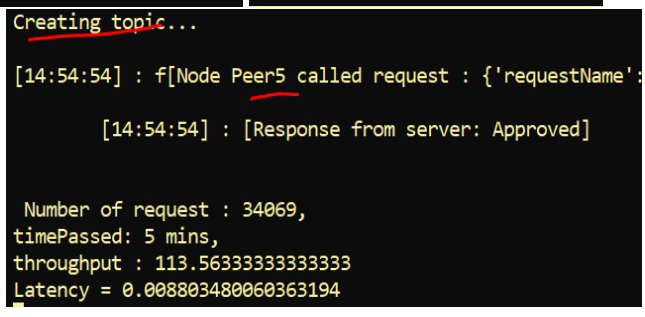
### a. Create Topic



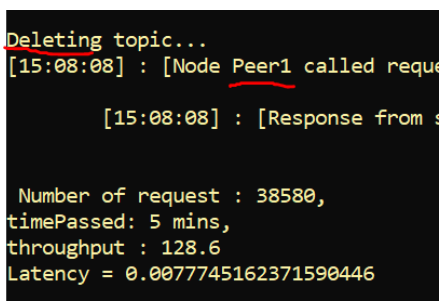



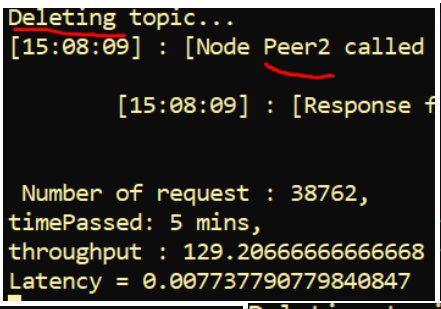



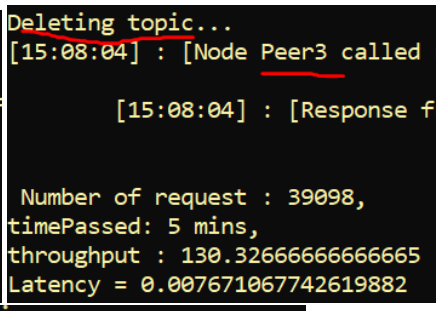



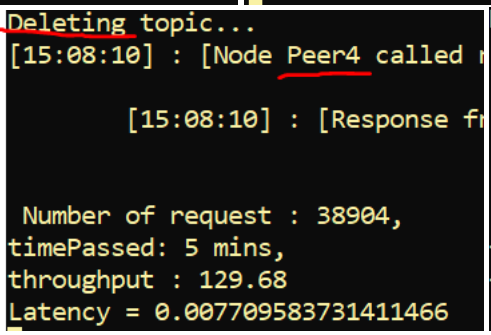



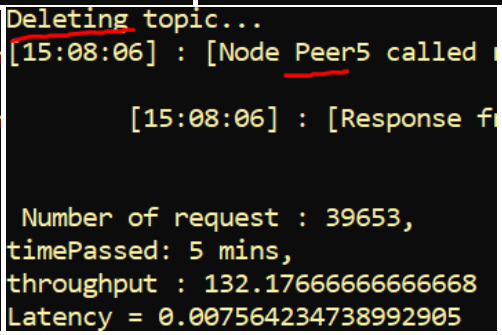
## b. Delete Topic



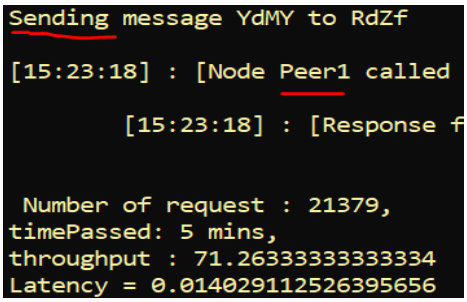



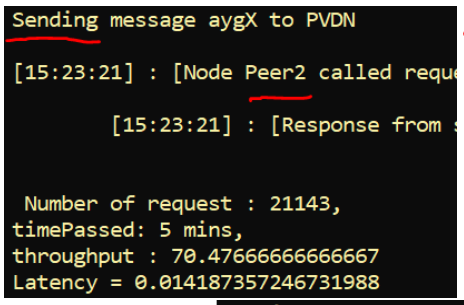



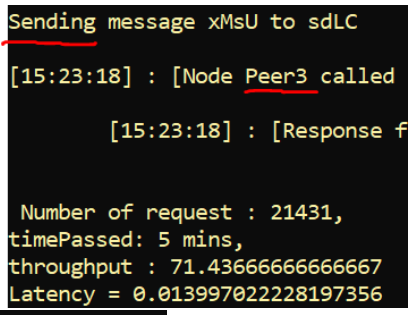



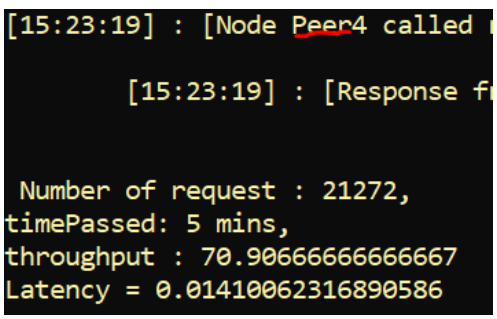



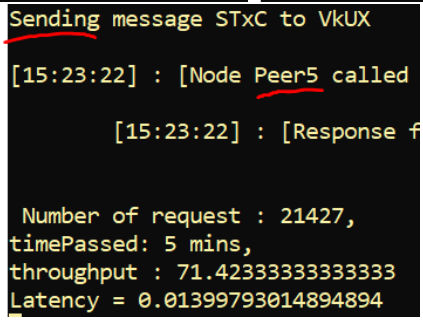

## c. Send Topic



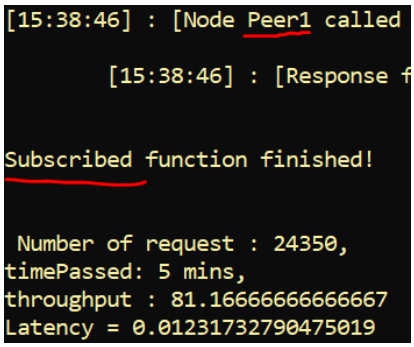



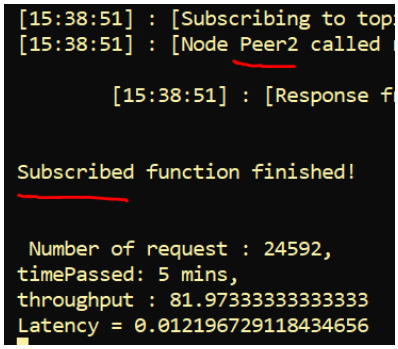



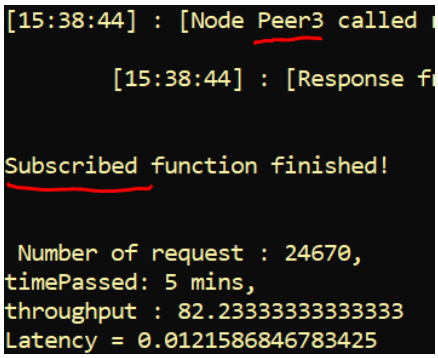



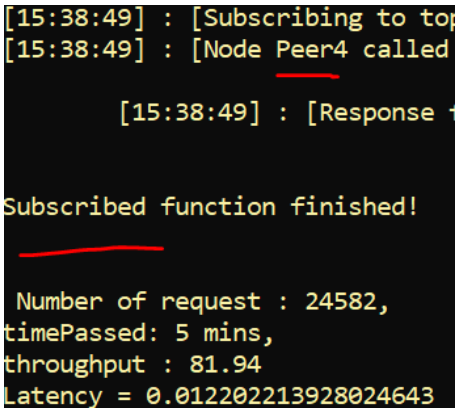



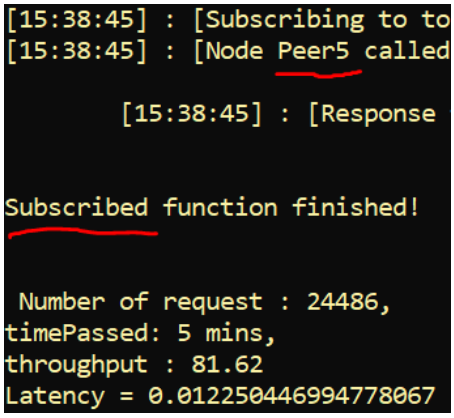
## d. Subscribe



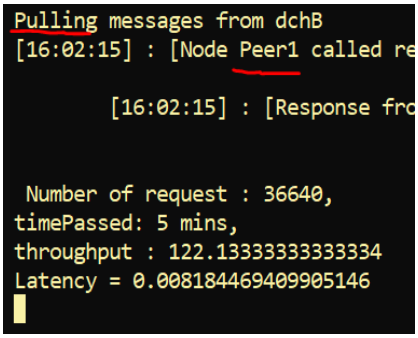



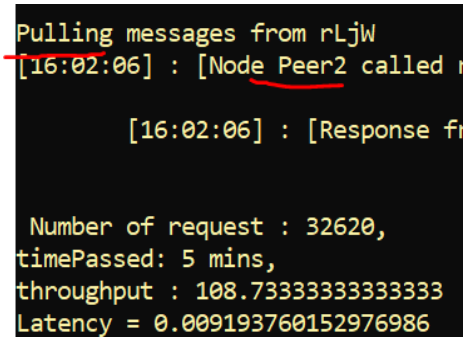



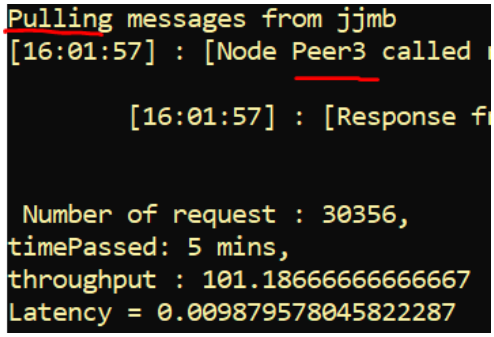



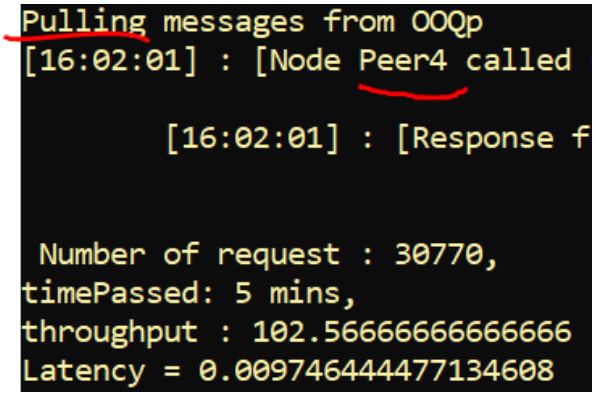



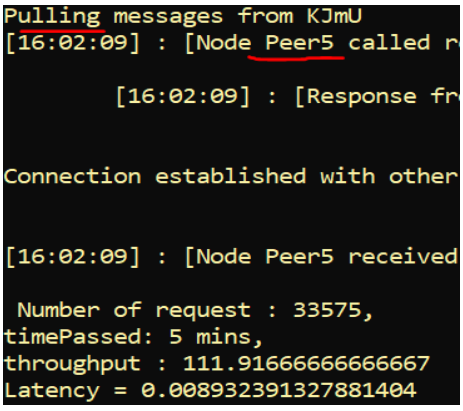

## e. Pull



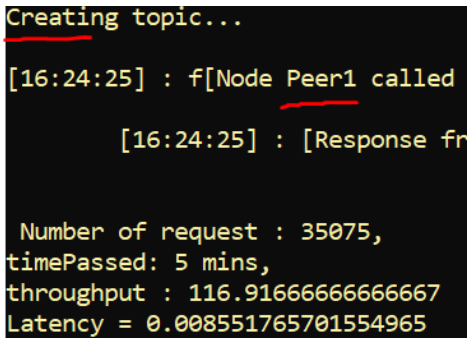



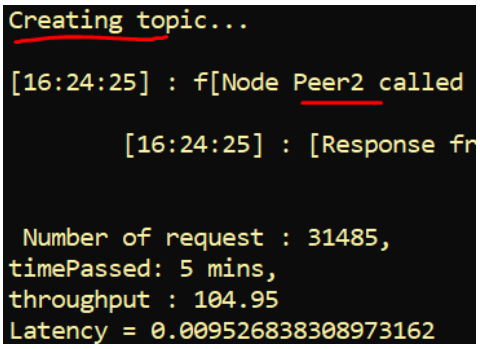



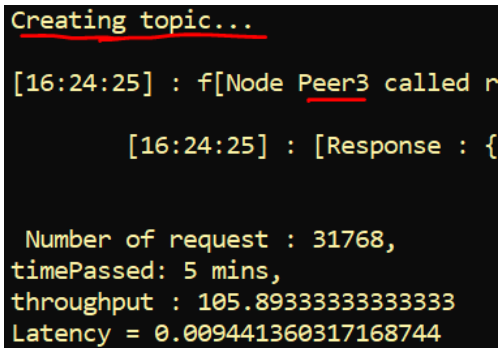



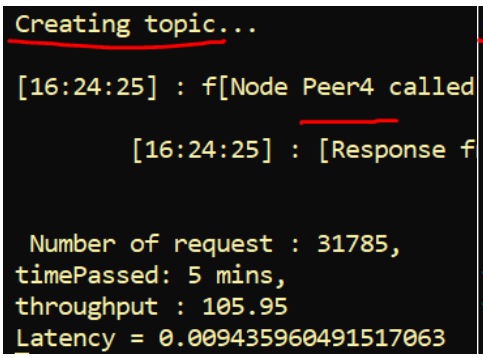
## 6 Peer Nodes

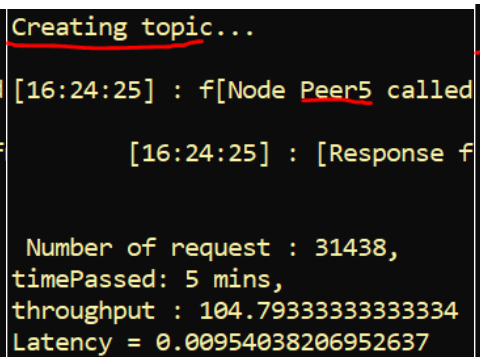
### a. Create Topic

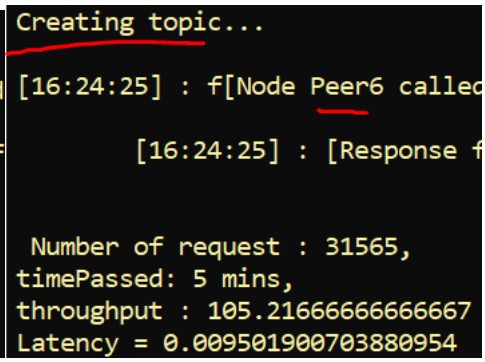


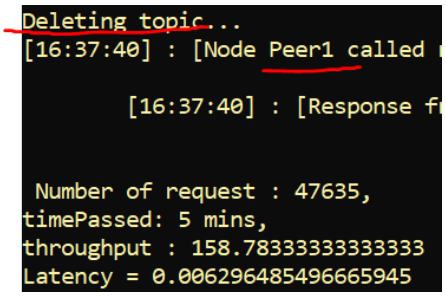



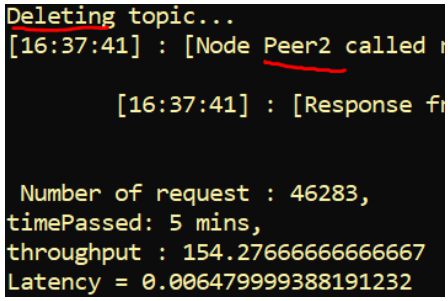



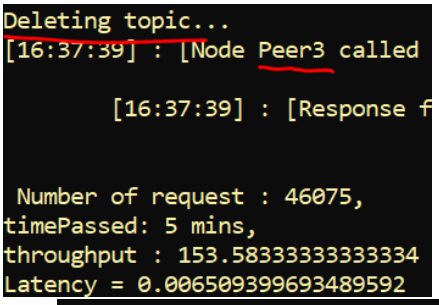



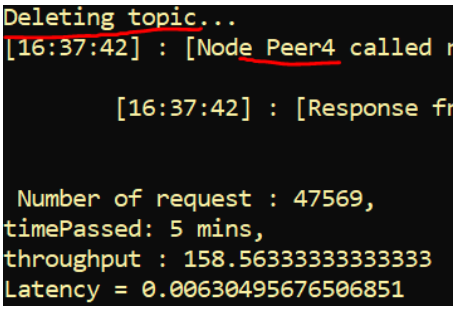



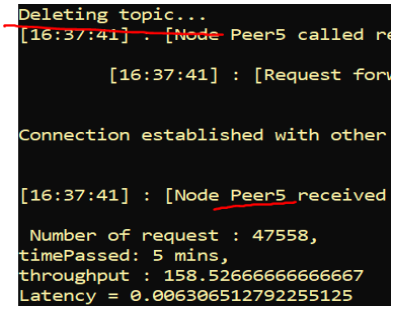

### b. Delete Topic

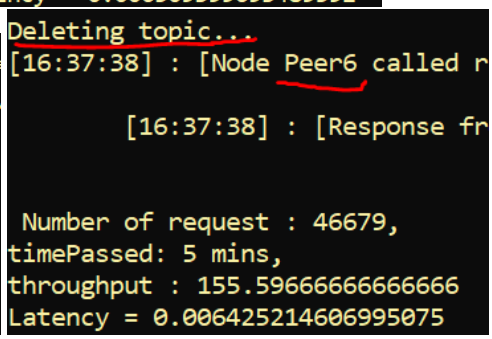


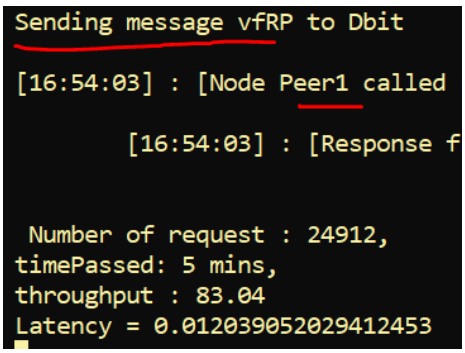



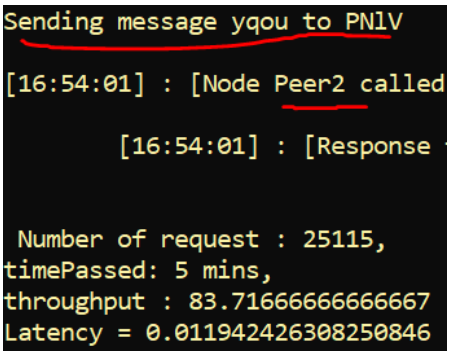



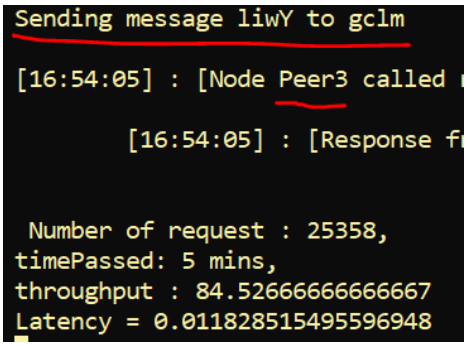



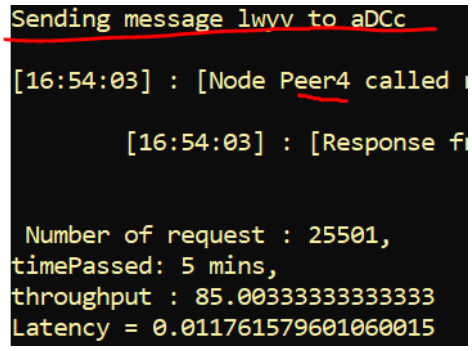



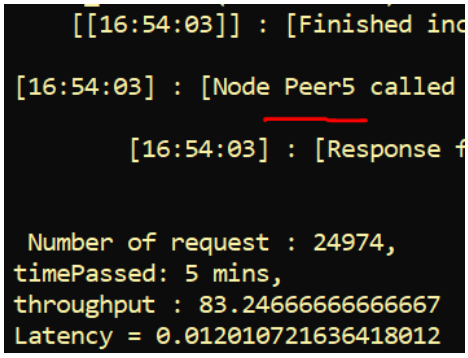
## c. Send Topic

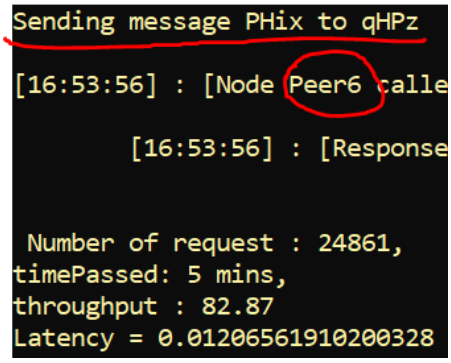


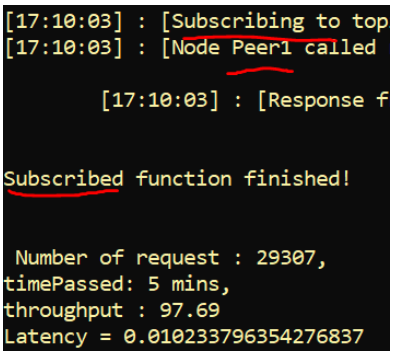



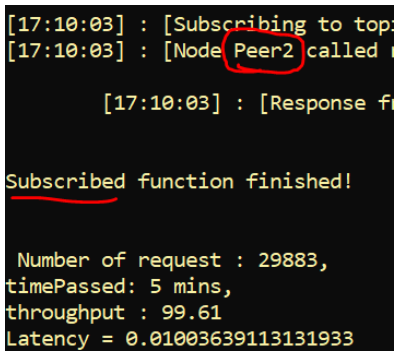



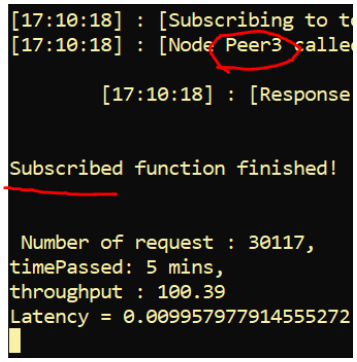



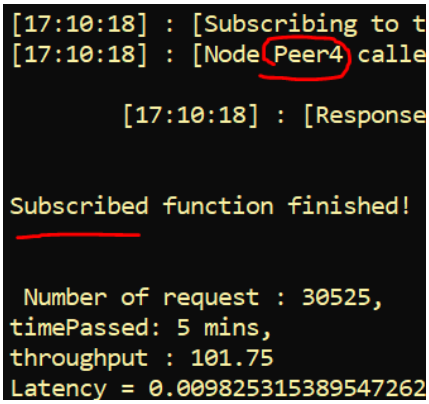



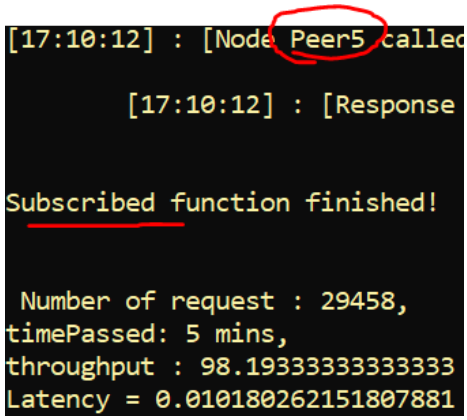

## d. Subscribe

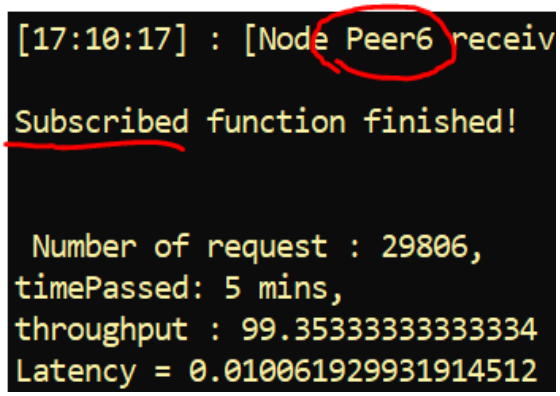


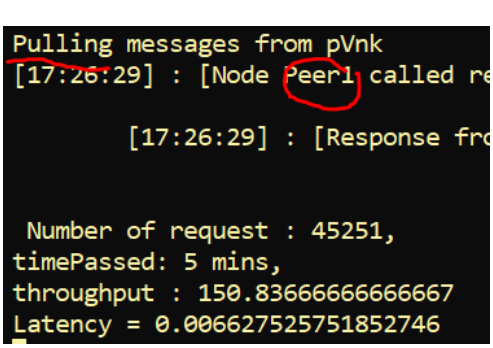



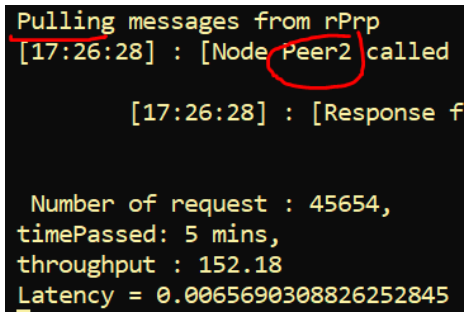



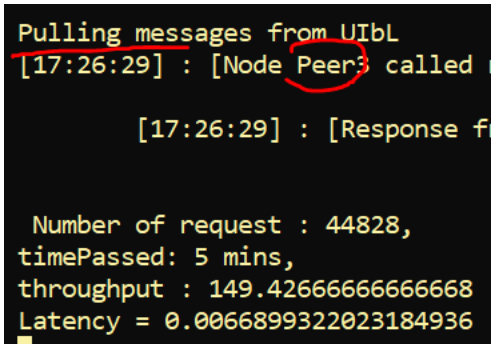



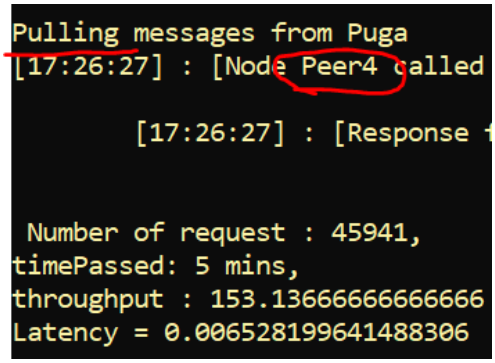



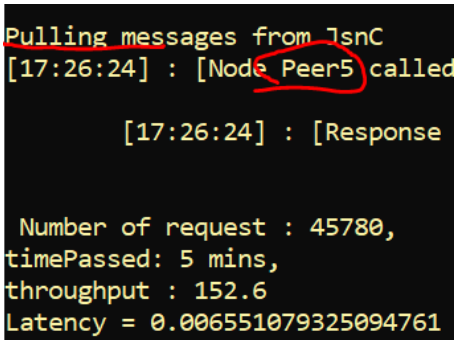
## e. Pull

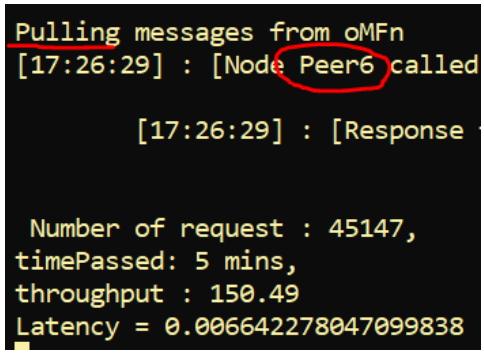


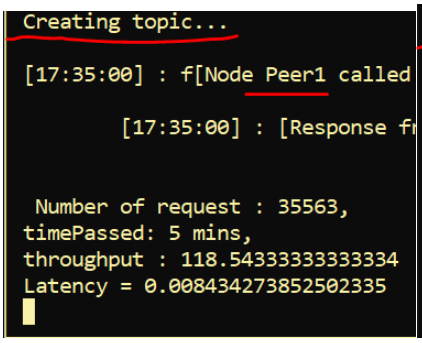



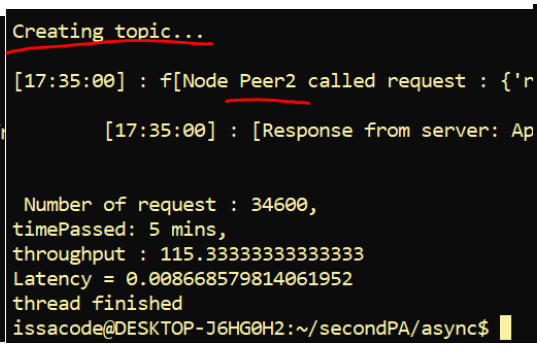



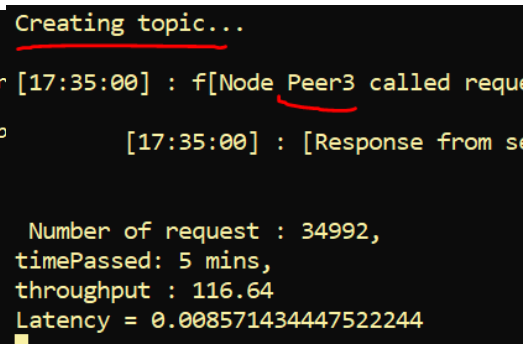



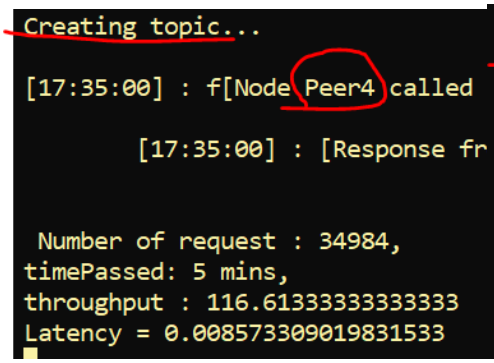

## 7 Peer Nodes

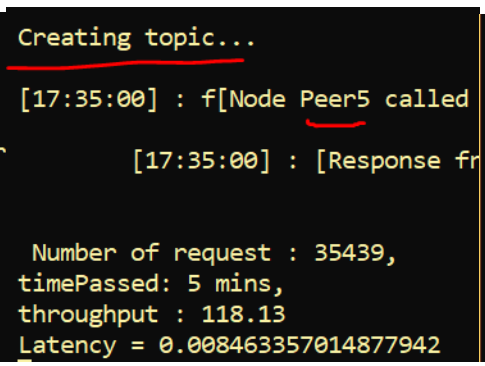
## a. Create Topic

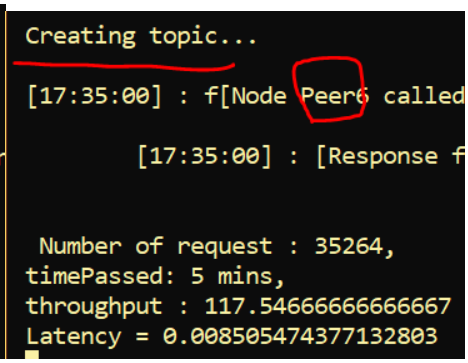


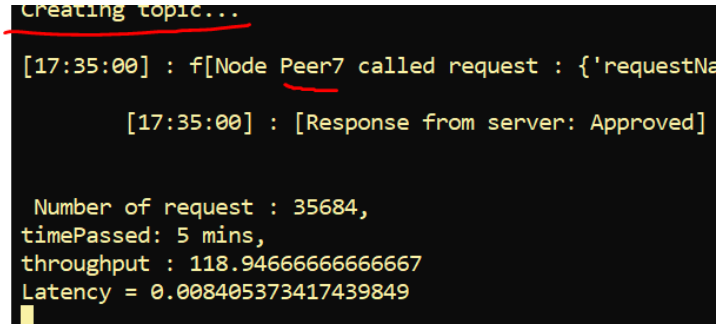



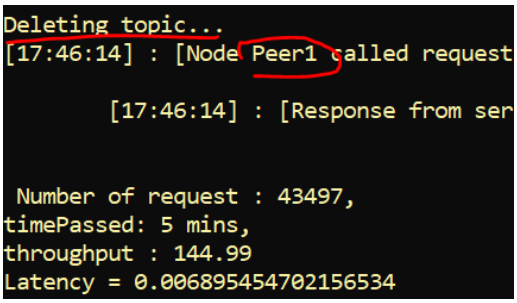



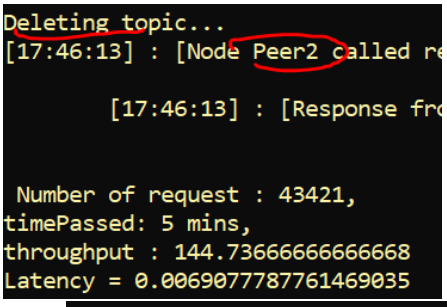



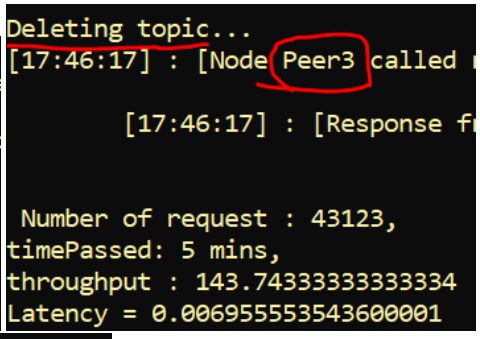



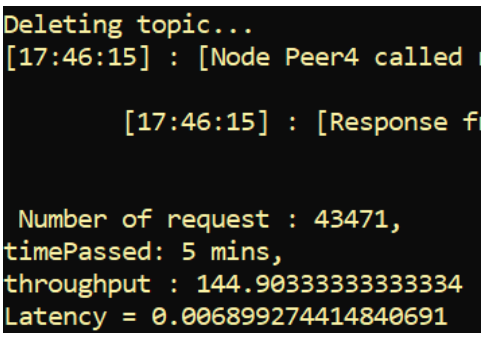


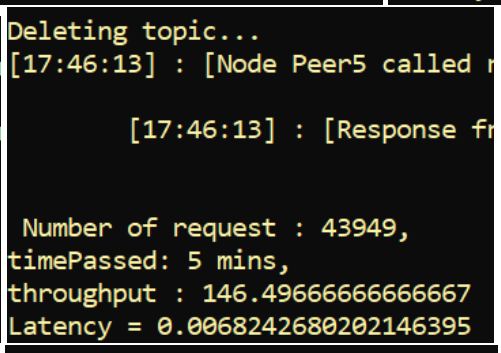

## b. Delete Topic

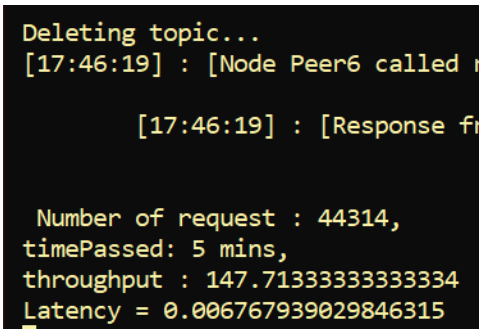


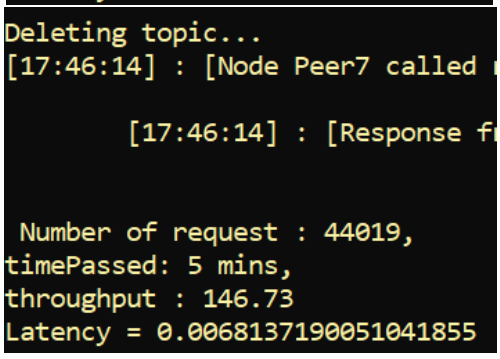



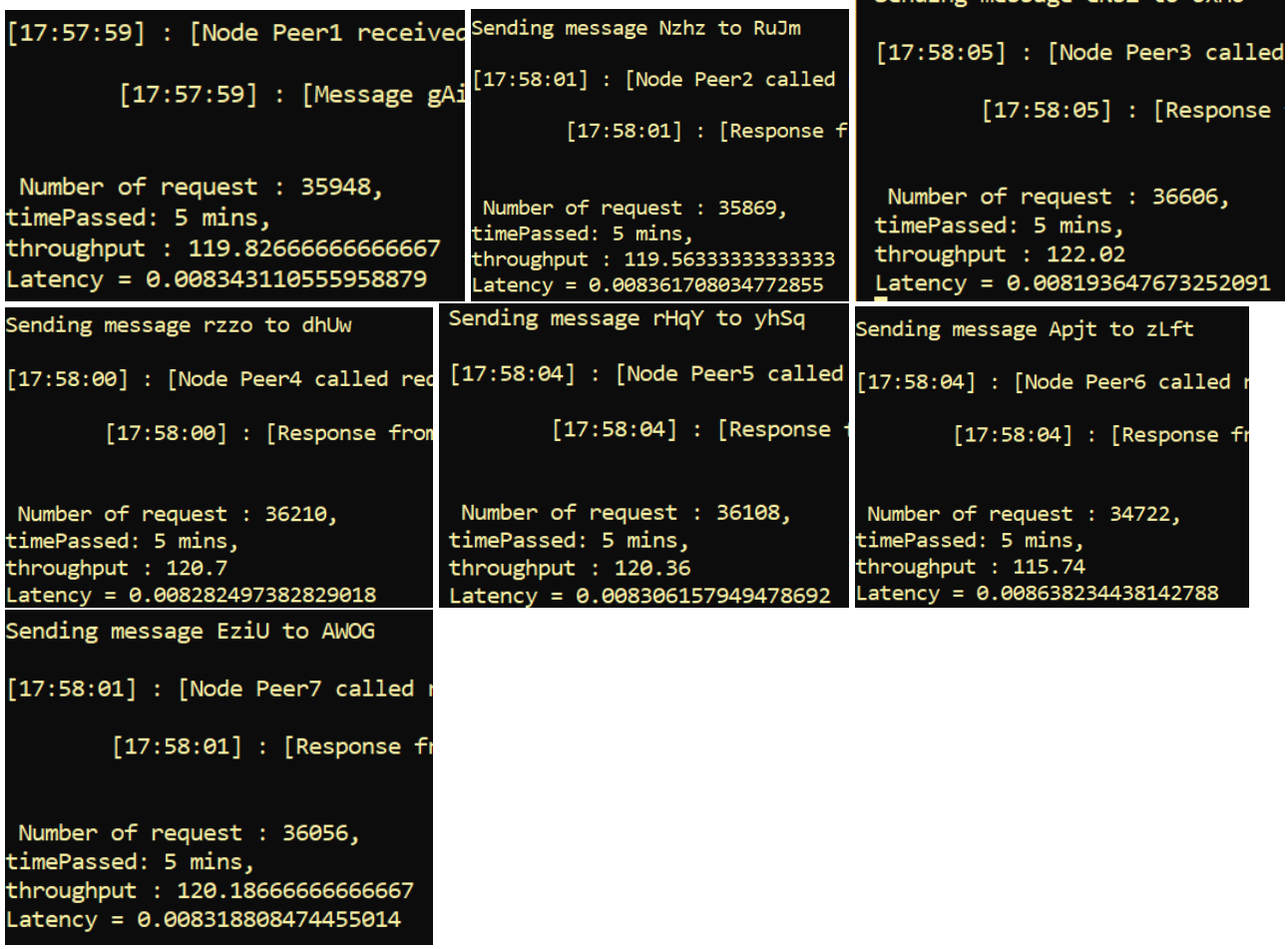



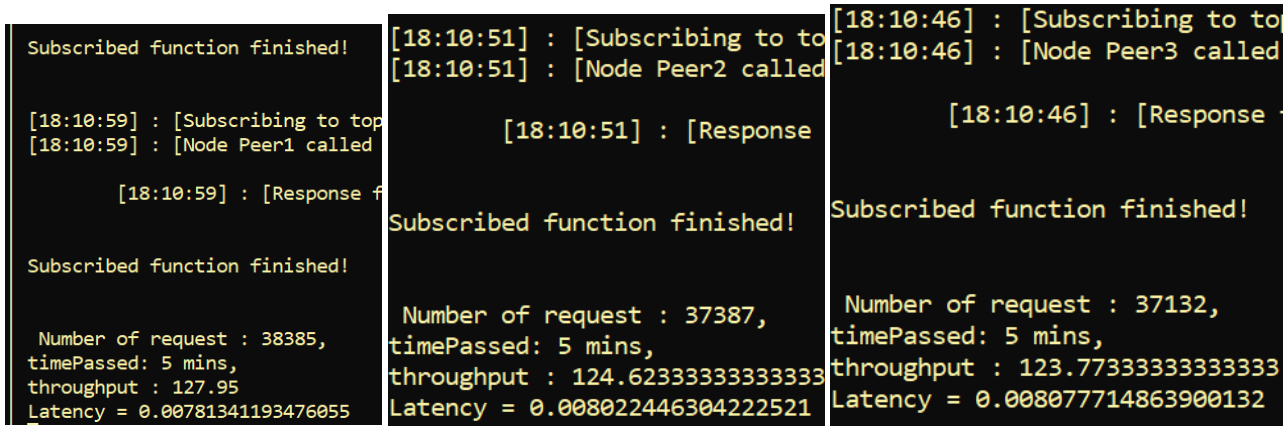



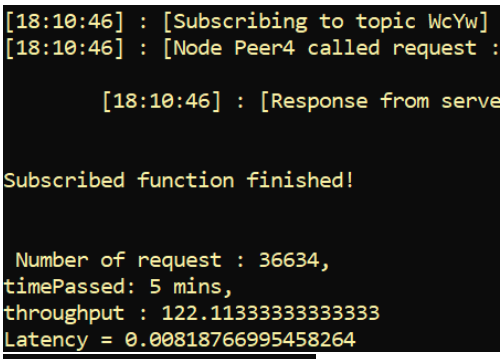



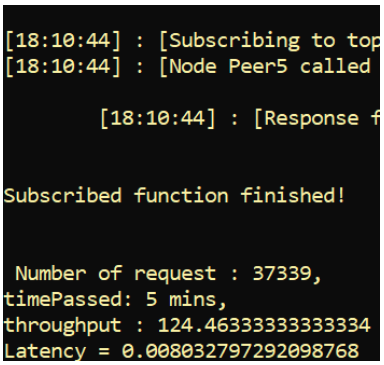
## c. Send

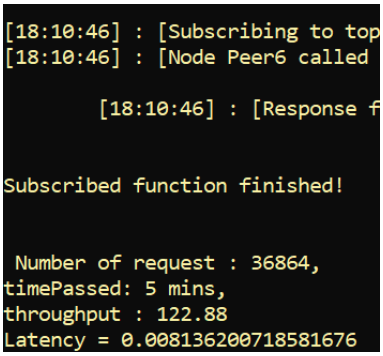


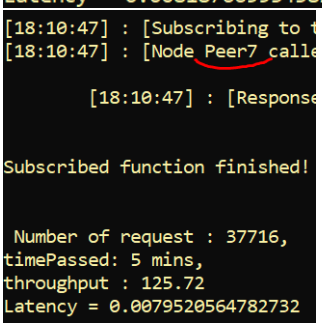

## d. Subscribe



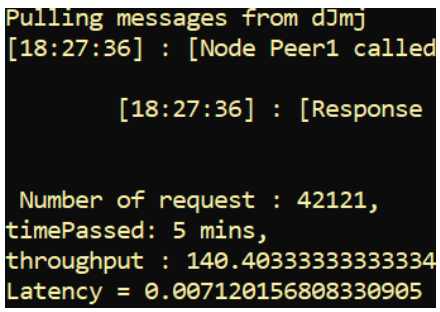

[18:10:46] : [Subscribing to topic WcYw]
[18:10:46] : [Node Peer4 called request :
[18:10:46] : [Response from serve
Subscribed function finished!
Number of request : 36634,
timePassed: 5 mins,
throughput : 122.11333333333333
Latency = 0.00818766995458264

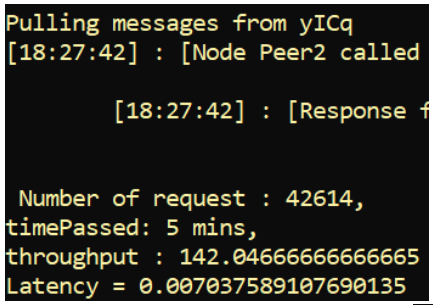

[18:10:44] : [Subscribing to top
[18:10:44] : [Node Peer5 called
[18:10:44] : [Response f
Subscribed function finished!
Number of request : 37339,
timePassed: 5 mins,
throughput : 124.46333333333334
Latency = 0.008032797292098768

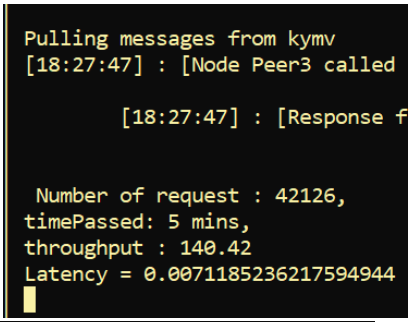

[18:10:46] : [Subscribing to top
[18:10:46] : [Node Peer6 called
[18:10:46] : [Response f
Subscribed function finished!
Number of request : 36864,
timePassed: 5 mins,
throughput : 122.88
Latency = 0.008136200718581676

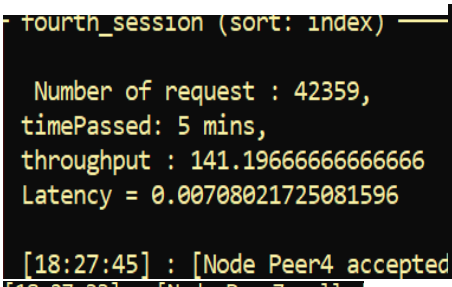

[18:10:47] : [Subscribing to t
[18:10:47] : [Node Peer7 calle
[18:10:47] : [Response
Subscribed function finished!
Number of request : 37716,
timePassed: 5 mins,
throughput : 125.72
Latency = 0.0079520564782732

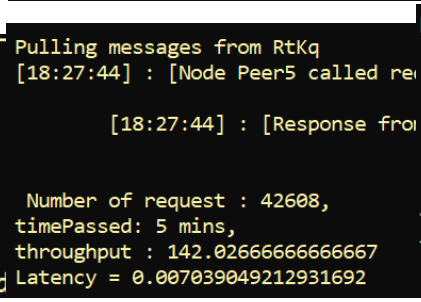
## e. Pull

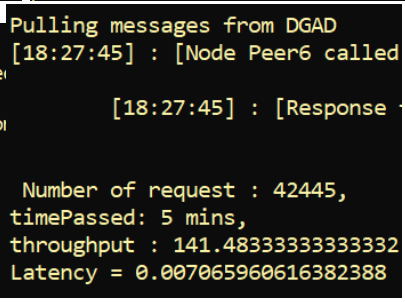

Pulling messages from dJmj
[18:27:36] : [Node Peer1 called
[18:27:36] : [Response
Number of request : 42121,
timePassed: 5 mins,
throughput : 140.40333333333334
Latency = 0.007120156808330905

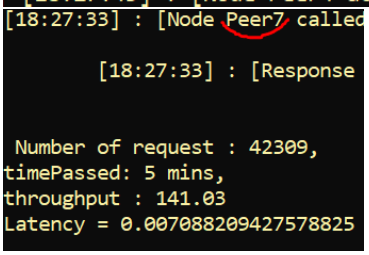

Pulling messages from yICq
[18:27:42] : [Node Peer2 called
[18:27:42] : [Response f
Number of request : 42614,
timePassed: 5 mins,
throughput : 142.04666666666665
Latency = 0.007037589107690135


Pulling messages from kymv
[18:27:47] : [Node Peer3 called
[18:27:47] : [Response f
Number of request : 42126,
timePassed: 5 mins,
throughput : 140.42
Latency = 0.007118523621759494


fourth_session (sort: index)
Number of request : 42359,
timePassed: 5 mins,
throughput : 141.19666666666666
Latency = 0.00708021725081596
[18:27:45] : [Node Peer4 accepted

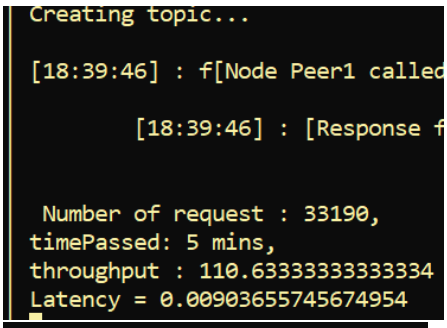

Pulling messages from RtKq
[18:27:44] : [Node Peer5 called re
[18:27:44] : [Response fro
Number of request : 42608,
timePassed: 5 mins,
throughput : 142.02666666666667
Latency = 0.007039049212931692

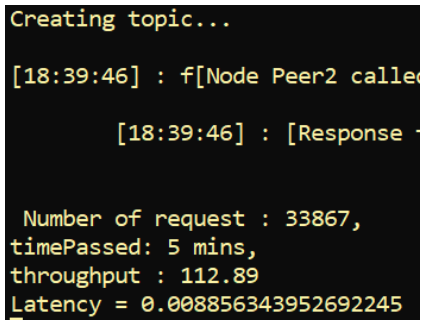

Pulling messages from DGAD
[18:27:45] : [Node Peer6 called
[18:27:45] : [Response
Number of request : 42445,
timePassed: 5 mins,
throughput : 141.48333333333332
Latency = 0.007065960616382388

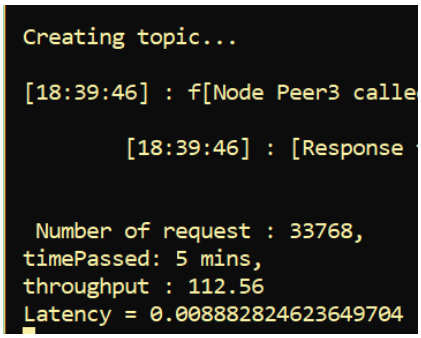

[18:27:33] : [Node Peer7 called
[18:27:33] : [Response
Number of request : 42309,
timePassed: 5 mins,
throughput : 141.03
Latency = 0.007088209427578825

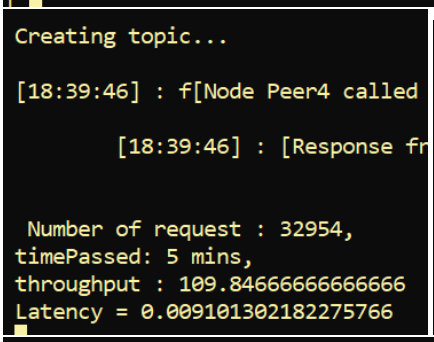
## 8 Peer Nodes

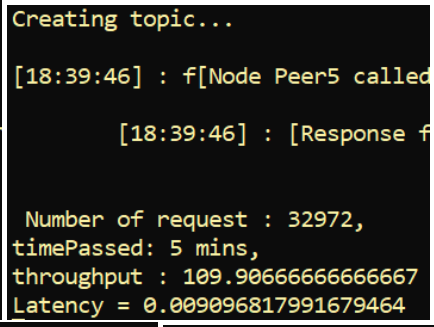
### a. Create Topic

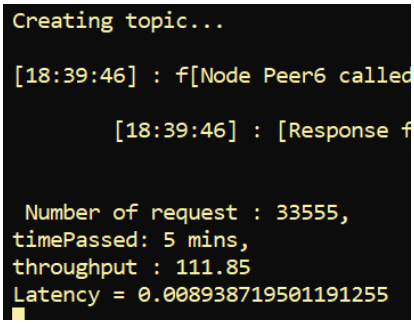


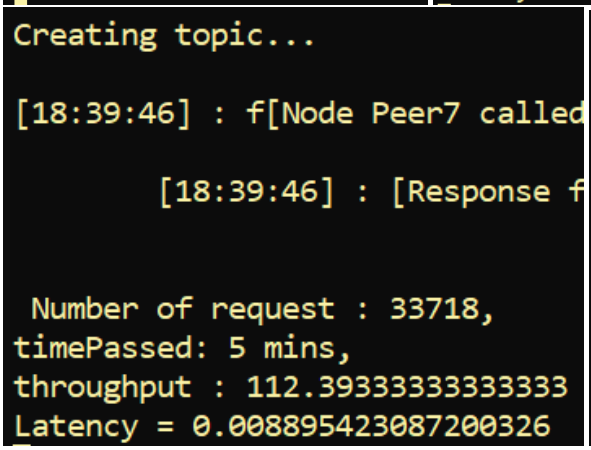



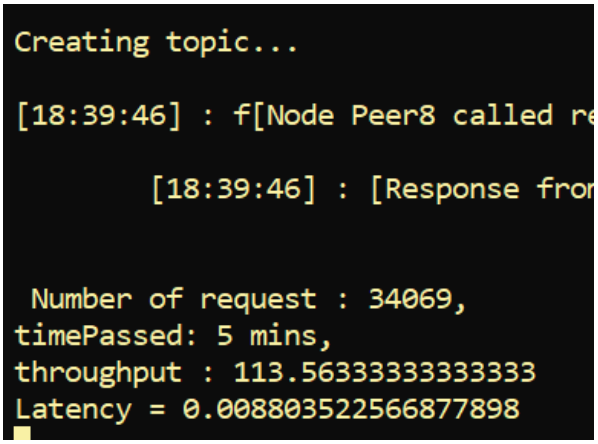



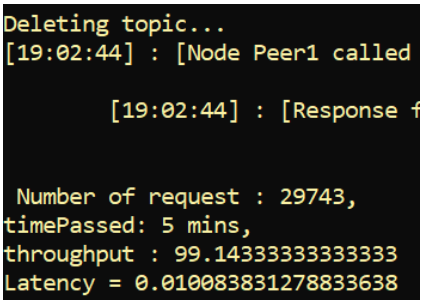



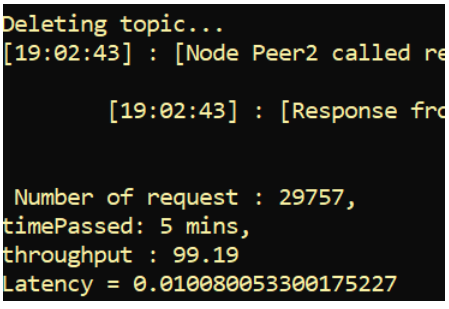



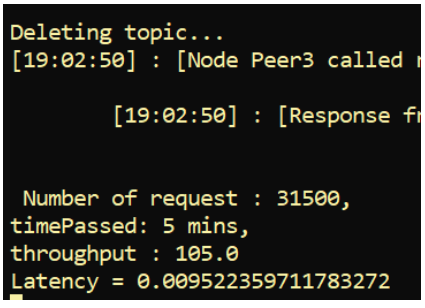



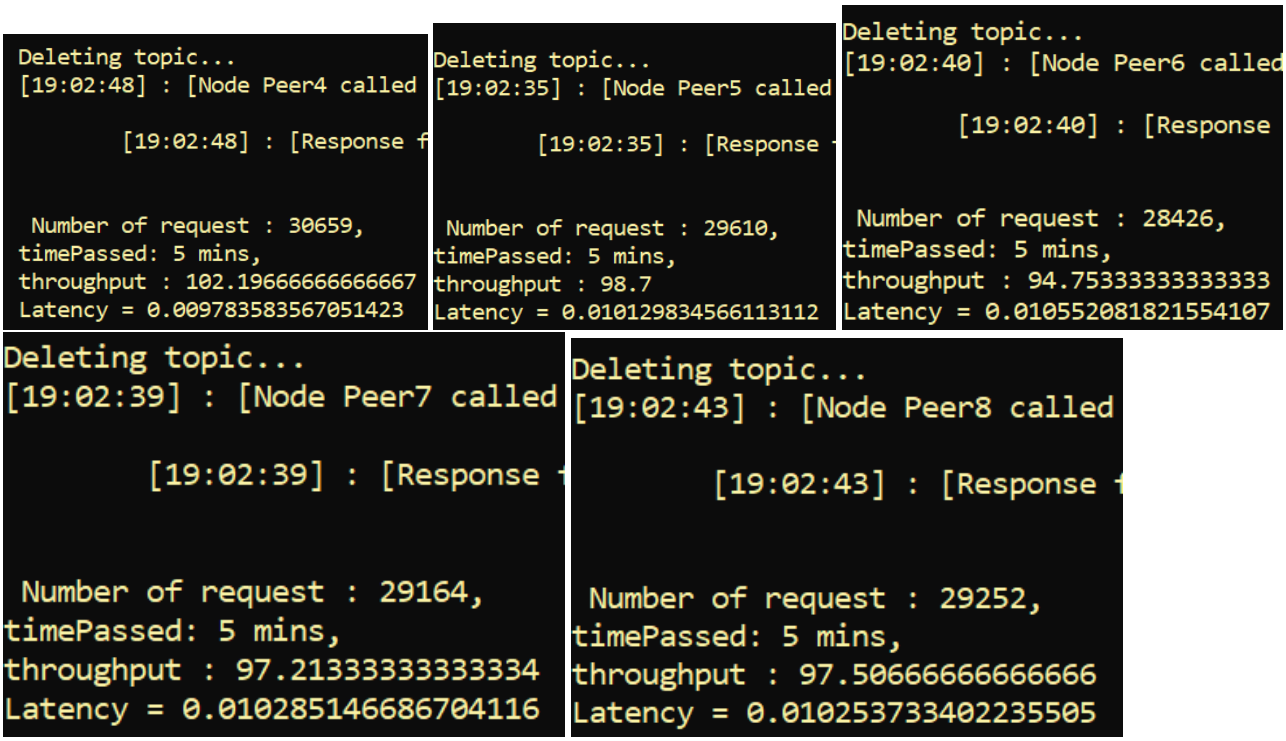




### b. Delete Topic

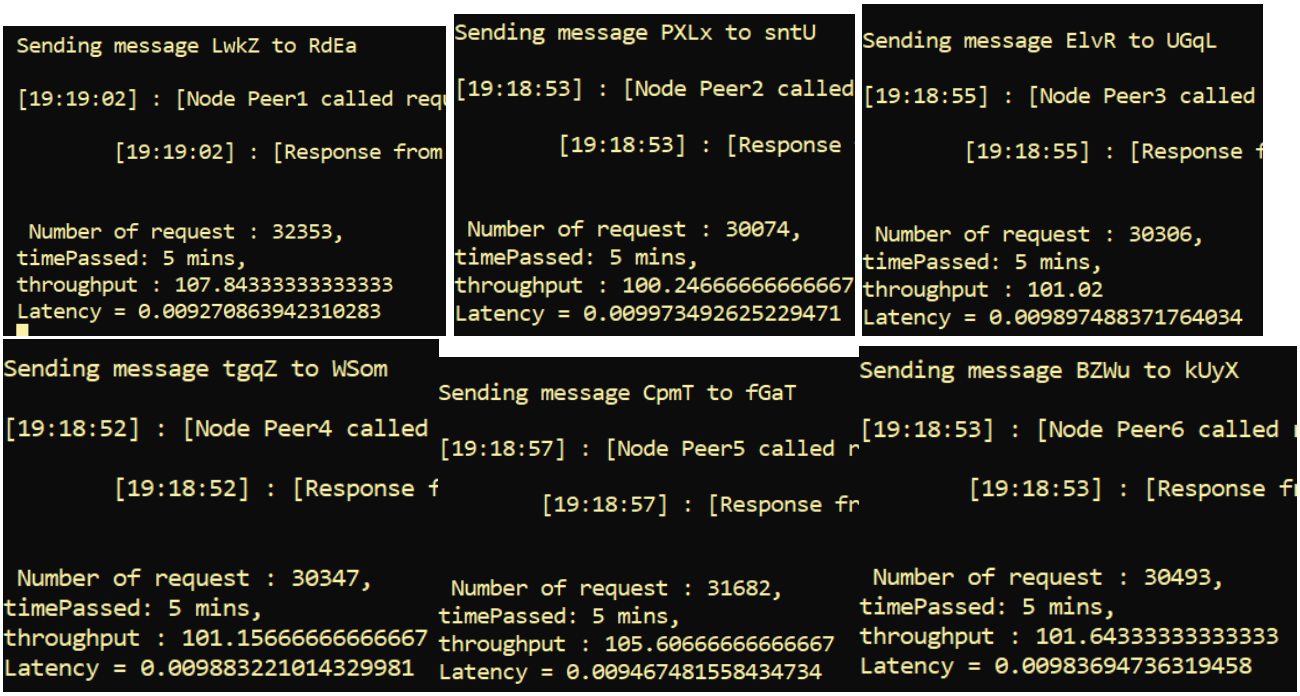



Deleting topic...
[19:02:48] : [Node Peer4 called
[19:02:48] : [Response f
Number of request : 30659,
timePassed: 5 mins,
throughput : 102.19666666666667
Latency = 0.009783583567051423
Deleting topic...
[19:02:35] : [Node Peer5 called
[19:02:35] : [Response
Number of request : 29610,
timePassed: 5 mins,
throughput : 98.7
Latency = 0.010129834566113112
Deleting topic...
[19:02:40] : [Node Peer6 called
[19:02:40] : [Response
Number of request : 28426,
timePassed: 5 mins,
throughput : 94.75333333333333
Latency = 0.010552081821554107
Deleting topic...
[19:02:39] : [Node Peer7 called
[19:02:39] : [Response
Number of request : 29164,
timePassed: 5 mins,
throughput : 97.21333333333334
Latency = 0.010285146686704116
Deleting topic...
[19:02:43] : [Node Peer8 called
[19:02:43] : [Response
Number of request : 29252,
timePassed: 5 mins,
throughput : 97.50666666666666
Latency = 0.010253733402235505

## c. Send


Sending message LwkZ to RdEa
[19:19:02] : [Node Peer1 called requ
[19:19:02] : [Response from
Number of request : 32353,
timePassed: 5 mins,
throughput : 107.84333333333333
Latency = 0.009270863942310283
Sending message PXLx to sntU
[19:18:53] : [Node Peer2 called
[19:18:53] : [Response
Number of request : 30074,
timePassed: 5 mins,
throughput : 100.24666666666667
Latency = 0.009973492625229471
Sending message ElvR to UGqL
[19:18:55] : [Node Peer3 called
[19:18:55] : [Response f
Number of request : 30306,
timePassed: 5 mins,
throughput : 101.02
Latency = 0.009897488371764034
Sending message tgqZ to WSom
[19:18:52] : [Node Peer4 called
[19:18:52] : [Response f
Number of request : 30347,
timePassed: 5 mins,
throughput : 101.15666666666667
Latency = 0.009883221014329981
Sending message CpmT to fGaT
[19:18:57] : [Node Peer5 called r
[19:18:57] : [Response fr
Number of request : 31682,
timePassed: 5 mins,
throughput : 105.60666666666667
Latency = 0.009467481558434734
Sending message BZWu to kUyX
[19:18:53] : [Node Peer6 called
[19:18:53] : [Response f
Number of request : 30493,
timePassed: 5 mins,
throughput : 101.64333333333333
Latency = 0.00983694736319458

```
Sending message DNiT to IXaC

[19:18:53] : [Node Peer7 called

        [19:18:53] : [Response

 Number of request : 30193,
timePassed: 5 mins,
throughput : 100.64333333333333
Latency = 0.0099339444501558
```

```
Sending message ITfo to IlDC

[19:18:59] : [Node Peer8 called

        [19:18:59] : [Response

 Number of request : 31292,
timePassed: 5 mins,
throughput : 104.30666666666667
Latency = 0.009584075553054839
```

## d. Subscribe

```
[19:32:55] : [Node Peer1 called

        [19:32:55] : [Response

 Subscribed function finished!

  Number of request : 29164,
 timePassed: 5 mins,
 throughput : 97.21333333333334
 Latency = 0.01028489979054764
```

```
[19:32:50] : [Subscribing to top
[19:32:50] : [Node Peer2 called

        [19:32:50] : [Response f

Subscribed function finished!

 Number of request : 27542,
timePassed: 5 mins,
throughput : 91.80666666666667
Latency = 0.010889314270324241
```

```
[19:32:47] : [Subscribing to topi
[19:32:47] : [Node Peer3 called r

        [19:32:47] : [Response fr

Subscribed function finished!

 Number of request : 30517,
timePassed: 5 mins,
throughput : 101.72333333333333
Latency = 0.009827356974138017
```

```
[19:32:51] : [Node Peer4 called re

       [19:32:51] : [Response fro

Subscribed function finished!

 Number of request : 28346,
timePassed: 5 mins,
throughput : 94.48666666666666
Latency = 0.010581700305431369
```

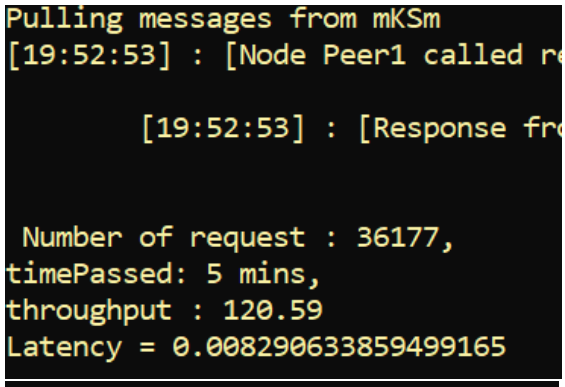
```
[19:32:48] : [Node Peer5 called r

       [19:32:48] : [Response fr

Subscribed function finished!

 Number of request : 27856,
timePassed: 5 mins,
throughput : 92.85333333333334
Latency = 0.010767956725966307
```

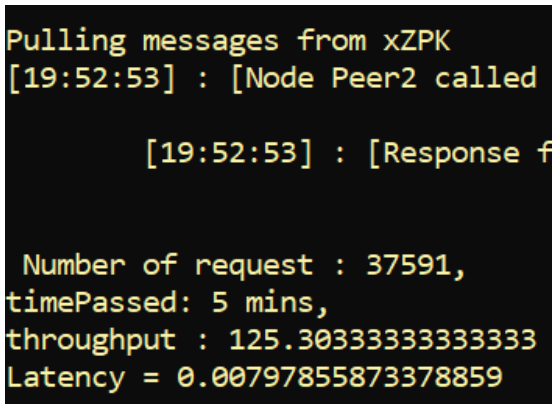
```
        [19:32:54] : [Response fro

Subscribed function finished!

 Number of request : 28252,
timePassed: 5 mins,
throughput : 94.17333333333333
Latency = 0.010616905220368757
```

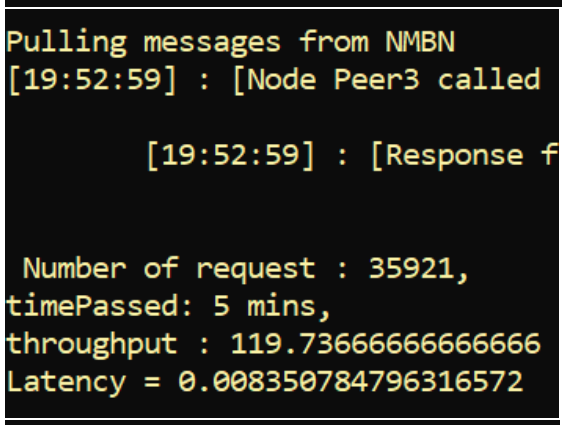
```
[19:32:50] : [Subscribing to topi
[19:32:50] : [Node Peer7 called r

       [19:32:50] : [Response fr

Subscribed function finished!

 Number of request : 27882,
timePassed: 5 mins,
throughput : 92.94
Latency = 0.010757706375838985
```

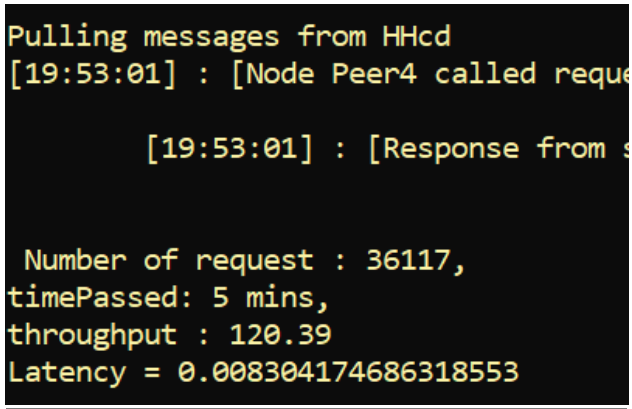
```
[19:32:49] : [Node Peer8 called

        [19:32:49] : [Response

Subscribed function finished!

 Number of request : 27978,
timePassed: 5 mins,
throughput : 93.26
Latency = 0.010720866890016332
```

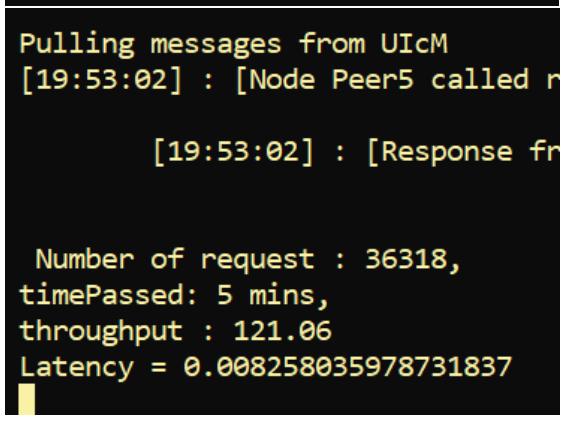
## e. Pull

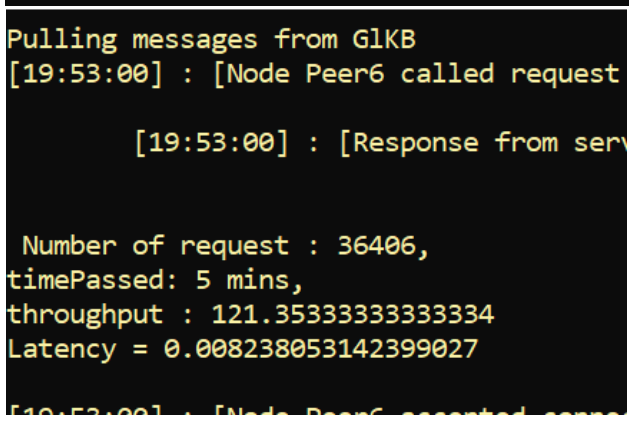

Pulling messages from mKSm
[19:52:53] : [Node Peer1 called r
[19:52:53] : [Response fr
Number of request : 36177,
timePassed: 5 mins,
throughput : 120.59
Latency = 0.008290633859499165

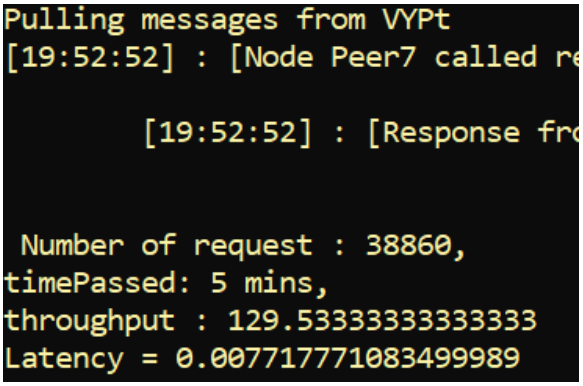

Pulling messages from xZPK
[19:52:53] : [Node Peer2 called
[19:52:53] : [Response f
Number of request : 37591,
timePassed: 5 mins,
throughput : 125.30333333333333
Latency = 0.00797855873378859

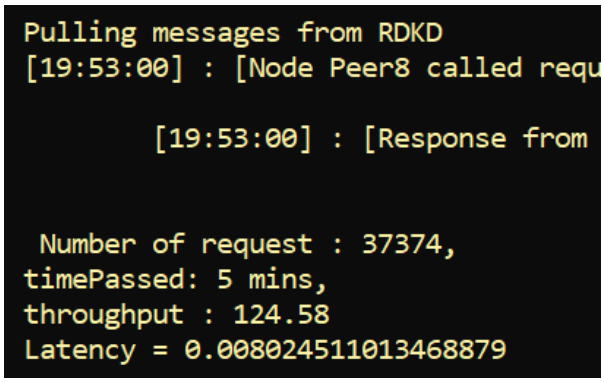

Pulling messages from NMBN
[19:52:59] : [Node Peer3 called
[19:52:59] : [Response f
Number of request : 35921,
timePassed: 5 mins,
throughput : 119.73666666666666
Latency = 0.008350784796316572


Pulling messages from HHcd
[19:53:01] : [Node Peer4 called reque
[19:53:01] : [Response from
Number of request : 36117,
timePassed: 5 mins,
throughput : 120.39
Latency = 0.008304174686318553


Pulling messages from UIcM
[19:53:02] : [Node Peer5 called r
[19:53:02] : [Response fr
Number of request : 36318,
timePassed: 5 mins,
throughput : 121.06
Latency = 0.008258035978731837


Pulling messages from GlKB
[19:53:00] : [Node Peer6 called request
[19:53:00] : [Response from serv
Number of request : 36406,
timePassed: 5 mins,
throughput : 121.35333333333334
Latency = 0.008238053142399027


Pulling messages from VYPt
[19:52:52] : [Node Peer7 called re
[19:52:52] : [Response fr
Number of request : 38860,
timePassed: 5 mins,
throughput : 129.53333333333333
Latency = 0.007717771083499989


Pulling messages from RDKD
[19:53:00] : [Node Peer8 called requ
[19:53:00] : [Response from
Number of request : 37374,
timePassed: 5 mins,
throughput : 124.58
Latency = 0.008024511013468879