P2P Report

Nina Wang(nw364) & Guangwei Jiang(gj94)

1.Overview

We implemented a distributed chat room application based on the provided proxy, including "p2p.c", "p2p.h", "process.c", implementing the rumormongering scheme and the anti-entropy, and implement Protocol according to the requirements of proxy API, than make a run and test.

2.Implementation Details

Proxy (proxy.py): acts as a middleman between the client and the server, starting the server process, establishing connections and forwarding messages. It is responsible for managing the lifecycle of the server instance and ensuring that messages are routed correctly. The send function in the provided proxy.py encodes what is being sent, so we decode the data as we receive it.

Server processes (process.c): each server process is responsible for handling incoming connections, processing client commands (e.g., sending messages, fetching chat logs), and executing the gossip and anti-entropy protocols for message propagation and synchronization across the P2P network, and determining whether it's a tcp (determining the type MSG,GET) or an udp(RUMOR,STATUS) after receiving a socket .

"p2p.h" and "p2p.c": are responsible for handling messages and commands, implementing the gossip protocol and anti-entropy mechanism, and providing various helper functions that support application operations. It contains logic to initialize data structures, maintain state for continuous server operation, handle timeouts, and trigger anti-entropy sessions.

Gossip Protocol: enables the propagation of information through the network by allowing peer nodes to forward information to randomly selected neighbors. It ensures wide dissemination of information even in case of network partitioning or failure.

Anti-entropy: is invoked periodically to ensure ultimate consistency throughout the network. The peer chooses a neighbor at random to exchange message digests (status messages), identifies any discrepancies in the logs, and requests missing messages to synchronize its state.

3.Analysis of Vulnerabilities

Denial of Service (DoS): An attacker can send a large number of false messages or connection requests to a peer device, consuming its resources and potentially causing it to crash or become unresponsive.

Man-in-the-Middle (MitM) Attacks: Without encryption, information exchanged between peer networks can be intercepted, read, or tampered with by an attacker, compromising the confidentiality and integrity of communications.

Impersonation attacks: an attacker can create multiple false identities to influence the network, disrupt gossip protocols, or bias the selection process in an anti-entropy mechanism.

Data Consistency: While gossip and anti-entropy mechanisms aim for ultimate consistency, immediate consistency cannot be guaranteed. This can lead to temporary discrepancies in chat logs viewed by different users.

4.Run & Test

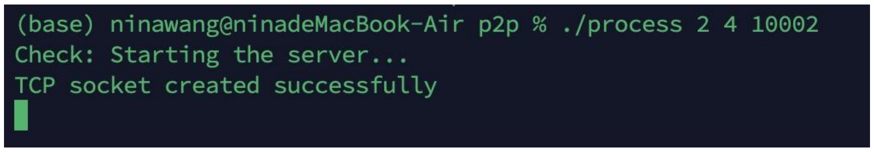
In the Makefile, we provide commands to compile the server and clean up the compiled objects for easy deployment and testing of the application, which we tested by running it with the macOS environment.

Run \`make\` to compile the project before Testing.

Since we need to simulate the communication of different servers on a single host, we need to turn on multiple terminals, one for the proxy and the others as different servers.

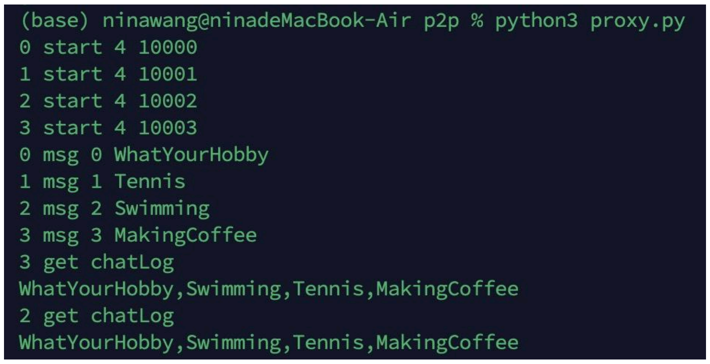
TESTING:

The command to start the servers is shown below, we need to start each server used for testing (e.g. if we need to use 4 servers for testing, we need to run the other servers in the same way and set the correct pid and port):



```
(base) ninawang@ninadeMacBook-Air p2p % ./process 2 4 10002
Check: Starting the server...
TCP socket created successfully
```

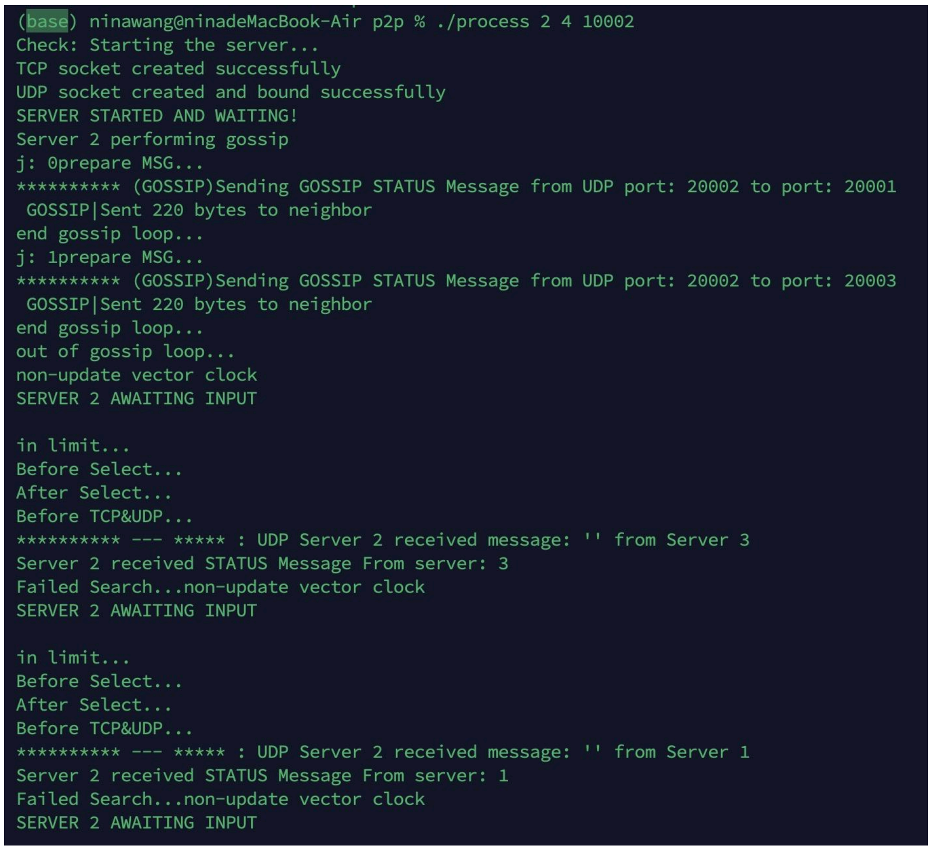
After that we can run the proxy in another terminal with the command \`Python3 proxy.py\`, after that we can enter commands in the proxy to test the protocol.



```
(base) ninawang@ninadeMacBook-Air p2p % python3 proxy.py
0 start 4 10000
1 start 4 10001
2 start 4 10002
3 start 4 10003
0 msg 0 WhatYourHobby
1 msg 1 Tennis
2 msg 2 Swimming
3 msg 3 MakingCoffee
3 get chatLog
WhatYourHobby,Swimming,Tennis,MakingCoffee
2 get chatLog
WhatYourHobby,Swimming,Tennis,MakingCoffee
```

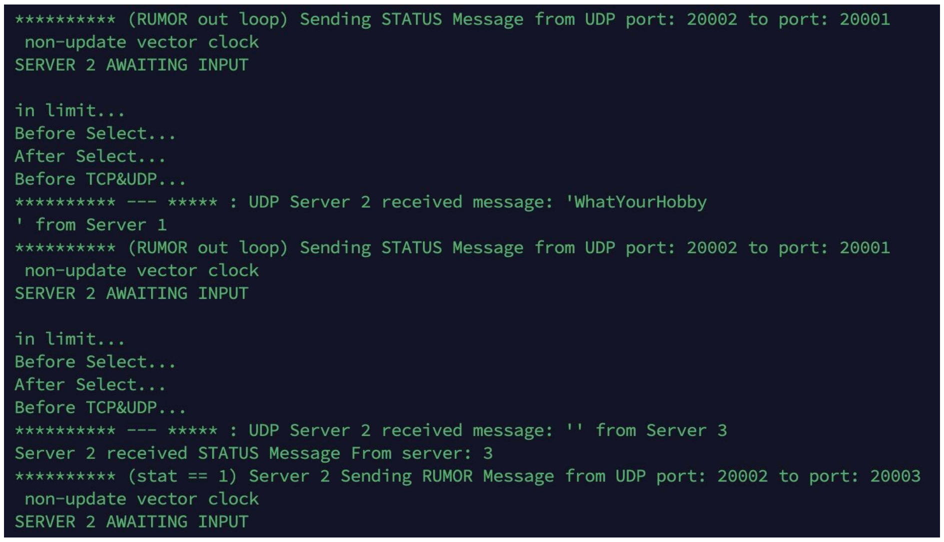
From the logs you can see that the 4 servers were created correctly, sent messages, and got the correct [chatLog]

We can also look at more information in each server we run to see the whole process:



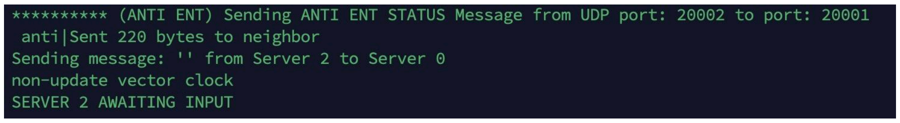
```
(base) ninawang@ninadeMacBook-Air p2p % ./process 2 4 10002
Check: Starting the server...
TCP socket created successfully
UDP socket created and bound successfully
SERVER STARTED AND WAITING!
Server 2 performing gossip
j: 0prepare MSG...
********** (GOSSIP)Sending GOSSIP STATUS Message from UDP port: 20002 to port: 20001
 GOSSIP|Sent 220 bytes to neighbor
end gossip loop...
j: 1prepare MSG...
********** (GOSSIP)Sending GOSSIP STATUS Message from UDP port: 20002 to port: 20003
 GOSSIP|Sent 220 bytes to neighbor
end gossip loop...
out of gossip loop...
non-update vector clock
SERVER 2 AWAITING INPUT

in limit...
Before Select...
After Select...
Before TCP&UDP...
********** --- ***** : UDP Server 2 received message: '' from Server 3
Server 2 received STATUS Message From server: 3
Failed Search...non-update vector clock
SERVER 2 AWAITING INPUT

in limit...
Before Select...
After Select...
Before TCP&UDP...
********** --- ***** : UDP Server 2 received message: '' from Server 1
Server 2 received STATUS Message From server: 1
Failed Search...non-update vector clock
SERVER 2 AWAITING INPUT
```

(Send and receive STATUS message)



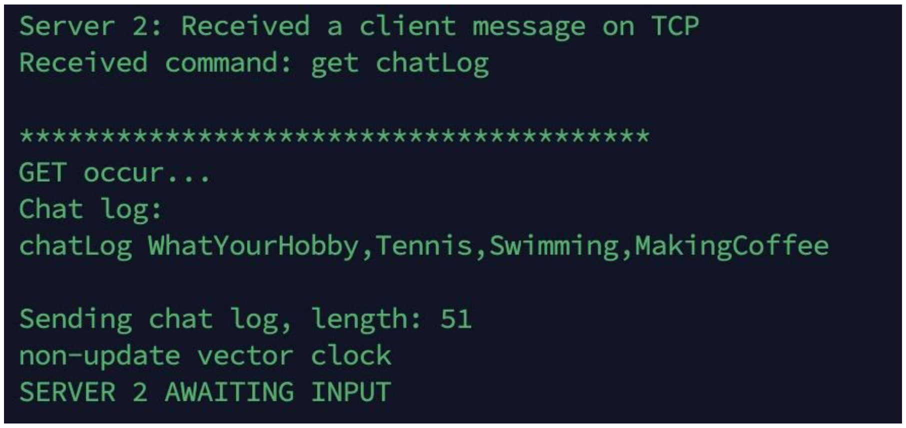
```
********** (RUMOR out loop) Sending STATUS Message from UDP port: 20002 to port: 20001
 non-update vector clock
SERVER 2 AWAITING INPUT

in limit...
Before Select...
After Select...
Before TCP&UDP...
********** --- ***** : UDP Server 2 received message: 'WhatYourHobby
' from Server 1
********** (RUMOR out loop) Sending STATUS Message from UDP port: 20002 to port: 20001
 non-update vector clock
SERVER 2 AWAITING INPUT

in limit...
Before Select...
After Select...
Before TCP&UDP...
********** --- ***** : UDP Server 2 received message: '' from Server 3
Server 2 received STATUS Message From server: 3
********** (stat == 1) Server 2 Sending RUMOR Message from UDP port: 20002 to port: 20003
 non-update vector clock
SERVER 2 AWAITING INPUT
```

(Receive STATUS message and Send RUMOR message with text)



```
********** (ANTI ENT) Sending ANTI ENT STATUS Message from UDP port: 20002 to port: 20001
 anti|Sent 220 bytes to neighbor
Sending message: '' from Server 2 to Server 0
non-update vector clock
SERVER 2 AWAITING INPUT
```

(Anti-entropy)



```
Server 2: Received a client message on TCP
Received command: get chatLog

***************************************
GET occur...
Chat log:
chatLog WhatYourHobby,Tennis,Swimming,MakingCoffee

Sending chat log, length: 51
non-update vector clock
SERVER 2 AWAITING INPUT
```

(GET chatLog of server 2)