FINAL REPORT: GOSSIP PROTOCOL

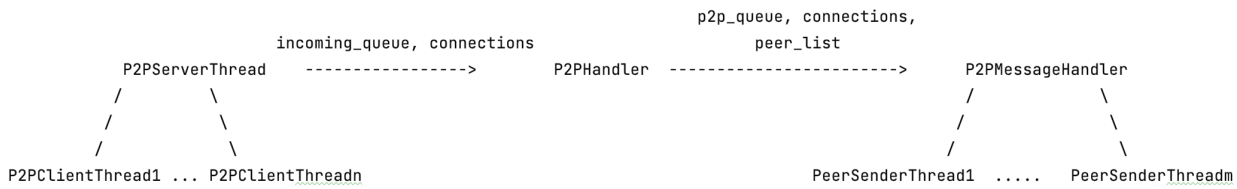
1. ARCHITECTURE

A. Logical structure

Classes :

1. Message - to handle each type of API and P2P message
2. MessageStorage - to handle the cache for announced messages and subscribers
3. ServerThread and client threads - inheriting from thread, handle creation of API/P2P server and their received connections
4. AnnounceMessageHandler - inheriting from thread, to act on received announce messages
5. NotifThread - inheriting from thread, to propagate notification messages
6. P2PHandler - thread to process received messages from other peers and take appropriate actions
7. P2PMessageHandler - thread to handle propagation of messages to other peers
8. PeerSenderThread - thread to send p2p message to a given peer

B. P2P Protocol Architecture



The peer list contains p2p server addresses of other peers that we know. A P2PServerThread starts a server for P2P connections and a new client thread is started every time a new connection is received. All received messages on the P2P layer are passed to an 'incoming_queue' along with the sender info. A P2PHandler is a thread which listens on the queue and processes the incoming messages. Following actions are taken according to received message type:

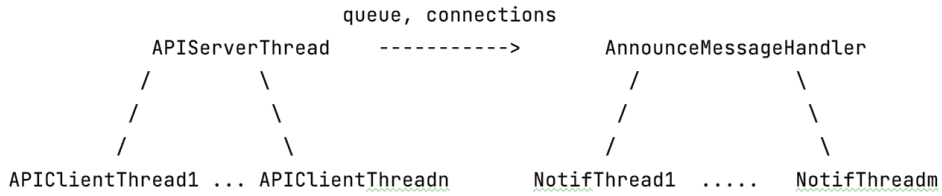
1. PUSH : Another peer has pushed their server address. We save it to our peer_list.
2. PULL : Another peer wants our known peers. We create a message with peers from our peer list and is put on the p2p_queue from where the message is later sent to the requester.
3. PULL RESPONSE : We had sent a pull request to a peer and they have responded with their peer list. We add it to our list and also create new connections to them if we have capacity.
4. SEND CONTENT : This is to send arbitrary content like Announce messages received from API layer. Another peer has sent us such a message. If it is found to be of type Announce, we propagate this to other peers after reducing ttl and also inform the API layer by putting it onto the announce_queue.

5. CONNECTION CLOSED : One of the p2p connections was dropped, we go through the peer list and add new connections to refill the capacity.

C. API Layer and P2P Layer Integration

announce_queue and p2p_queue are shared between the API and P2P layer. Announce messages are exchanged between these layers through these queues. On receiving announce message on the API front, the subscribed modules are informed via NOTIFICATION messages and the message is put to the p2p queue from where it is transmitted to all known peers. On receiving an announce message on the P2P layer, we forward it to API layer by putting it on the announce_queue from where it is passed again to subscribers and the message is also passed on to other peers after reducing the ttl.

D. API Process Architecture



APIServerThread starts server for API connections. When client connection is received, new APIClientThread is spawned for API connections and received messages are processed.

Another thread, AnnounceMessageHandler, is started from main to wait on queue of announced messages and spawn new threads for message spreading.

Shared resources between server and messagehandler threads : MessageStorage, announce_message_queue(queue), connections(dict)

On receiving message in a client connection, it is processed and appropriate action is taken in the client thread.

If announce message received, it is added to the announce message queue.

messagehandler retrieves the item from queue and spawns threads to send Notification messages to each of the subscribers.

Each subscriber thread checks if connection already exists for the subscriber and reuses it to send message. If not, new connection is created and message is sent.

E. Networking

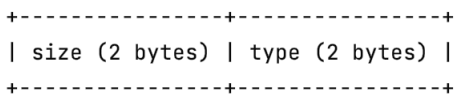
- socket: built-in socket module

socket connections handled by ServerThread using child ClientThreads. Connections are reused when notification message needs to be sent else new socket client opened in NotifThread/ PeerSenderThread.

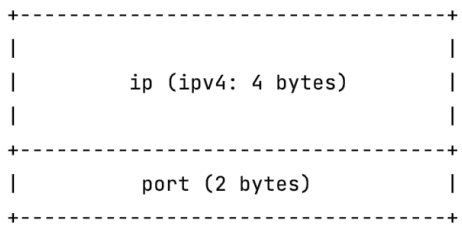
2. PEER-TO-PEER PROTOCOL

A. Message format

i. Header: each message in the p2p protocol starts with a header of the following form (same as in the API protocol)

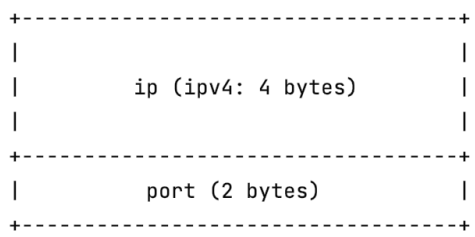


ii. Pull (type: 504): a node should be able to ask other peers for IDs. It sends it's own IP address and port.



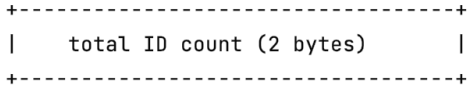
iii. Response (type: 505): a node should to be able to answer a pull request, sending it's IDs to the receiver.

iv. Push (type: 506): a node should be able to push its own IP and port to a random peer.

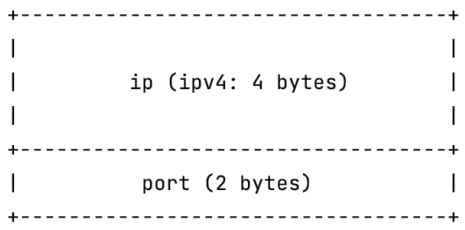


v. Send_Content (type: 507): a node should be able to send arbitrary binary content to another node. Can be used to send announce messages.

vi. Format for IDs: IDs should start with the total number of IDs included:



And continue with that number of the following:



B. Reasoning

We have chosen to go for a push-pull approach. Thus we need four p2p-messages: push, pull, a response to the pull request and a message type to send arbitrary content to peers.

C. Exception handling

If the IDs of any of the messages are corrupted, i.e. not in the specified format, the message is simply discarded. If a connection breaks unexpectedly, any data read from the connection is discarded. responseTimeout, if the client takes too long to respond, connection is closed. messageLengthExceeded, when the received message is longer than max expected length of 64K, the message is discarded.

3. HOW TO RUN

A. Installing and running

Install Python3 on your machine. Download the project and from the root directory of the project, run following to install requirements

pip install -r requirements.txt

To run the project :

python3 main.py

B. Sample run

Example run to test P2P functionality (Use --help for sample_peer.py to see options) :

--Running as bootstrapper, mention bootstrapper ip in config

python3 sample_peer.py -a "127.0.0.1" -p 8888 -pr "127.0.0.1:8080" "192.0.0.1:7777" "192.168.1.1:8080"

--Starting peer known by bootstrapper

python3 sample_peer.py -a "127.0.0.1" -p 8080 -pr 192.158.1.38:7777

--Running main function

python3 main.py

--Sending announce message, use p2p address of our server

python3 sample_peer.py -a "127.0.0.1" -p 7777 -da "127.0.0.1" -dp 6001 -ac send_announce

4. FUTURE WORK

Security measures yet to be implemented

5. WORKLOAD DISTRIBUTION

A. Nicolas:

Structure for the program to run (server/request handler thread classes, main), config parsing, exceptions, P2P protocol design, logging

B. Reshma:

API conformity, storage for messages, message and messagestorage classes and related tests, Process architecture design and implementation using thread classes as described, API workflow with sample clients operational, P2P message formats implementation, P2P protocol implementation, verifying functionality, example run setup, improving readability using comments, final report

6. EFFORT SPENT

A. Nicolas:

Config parsing ~3h, exceptions ~1h, structure for the program to run ~6h, P2P protocol design ~3h, logging ~2h

B. Reshma:

API conformity ~3h, design for messages and storage ~2h, defining classes and tests ~4h, process arch design ~5h, process arch impl. ~4h, ensuring working API workflow ~1h,

p2p message design ~2h, p2p protocol impl ~5h, p2p protocol testing and correction ~2h, example run setup ~2h, improving readability ~1h, final report ~2h