Peer-to-Peer Systems and Security

Team 23, SS21

Midterm Report

Alexander Liebald, Kevin Ploch

July 2021

1 Changes to assumptions made in the initial report

We continued to work on our understanding of Gossip and at the same time refined the flow chart, which we also used in the initial report. Since we kept major previous versions of it, it is possible to see how the project matured over the last months. The current version can be seen in figure 1.

1.1 Asynchronous I/O

Since the initial report, we replaced all uses of multi threading with Pythons AsyncIO (Asynchronous I/O). This helped to avoid race conditions without the need to implement potentially more complex synchronization. Additionally, AsyncIO is very pleasant to use.

1.2 Variables in config

We revised our understanding of some of the variables regarding peer connections in the config. Even though there was no definition given in the initial report, this had influence on the current version of our flow chart, which was given in the initial report. We removed cache_size, since its definition in the specification and in an answer on Moodle (27. may) where perceived as confusing and inconsistent. We decided to use min_connections instead, combined with max_connections and degree, which are specified in section 2.1.4.

2 Architecture

2.1 Logical structure

To provide a clear and understandable architecture, we decided to use a class based approach. A representation of these classes can be seen in figure 2. Besides these classes we also used smaller modules, which provide some functionality,

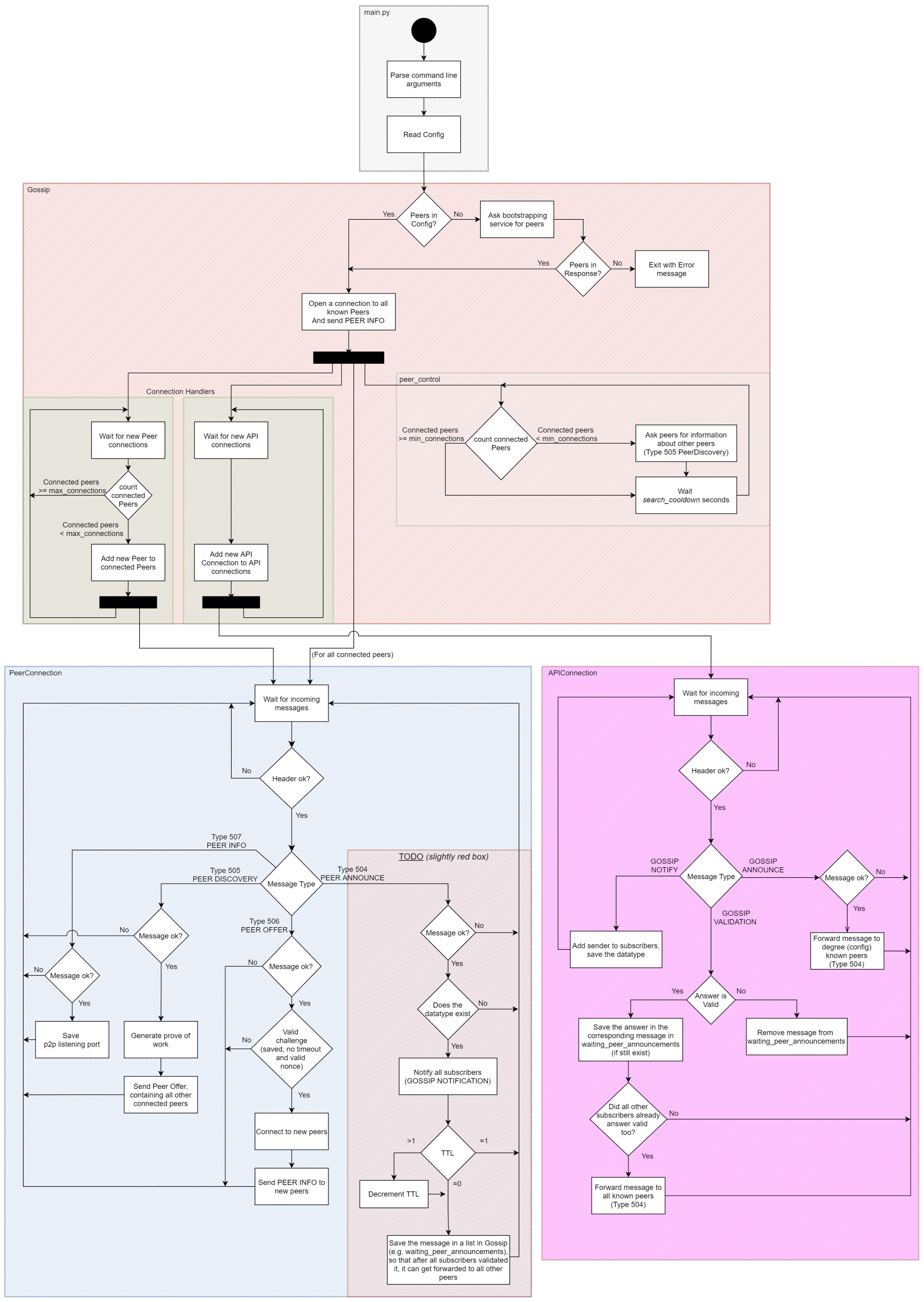
main.py
Parse command line arguments
Read Config
Gossip
Peers in Config?
Yes
No
Ask bootstrapping service for peers
Peers in Response?
Yes
No
Exit with Error message
Open a connection to all known Peers And send PEER INFO
Connection Handlers
Wait for new Peer connections
count connected Peers
Connected peers >= max_connections
Connected peers < max_connections
Add new Peer to connected Peers
Wait for new API connections
Add new API Connection to API connections
peer_control
count connected Peers
Connected peers >= min_connections
Connected peers < min_connections
Ask peers for information about other peers (Type 505 PeerDiscovery)
Wait *search_cooldown* seconds
(For all connected peers)
PeerConnection
Wait for incoming messages
Header ok?
No
Yes
Message Type
Type 507 PEER INFO
Type 505 PEER DISCOVERY
Type 506 PEER OFFER
Type 504 PEER ANNOUNCE
Message ok?
No
Yes
Message ok?
No
Yes
Save p2p listening port
Generate prove of work
Send Peer Offer, containing all other connected peers
Message ok?
No
Yes
Valid challenge (saved, no timeout and valid nonce)
No
Yes
Connect to new peers
Send PEER INFO to new peers
TODO *(slightly red box)*
Message ok?
No
Yes
Does the datatype exist
No
Yes
Notify all subscribers (GOSSIP NOTIFICATION)
TTL
>1
=1
=0
Decrement TTL
Save the message in a list in Gossip (e.g. waiting_peer_announcements), so that after all subscribers validated it, it can get forwarded to all other peers
APIConnection
Wait for incoming messages
Header ok?
No
Yes
Message Type
GOSSIP NOTIFY
GOSSIP ANNOUNCE
GOSSIP VALIDATION
Add sender to subscribers, save the datatype
Message ok?
No
Yes
Forward message to degree (config) known peers (Type 504)
Answer is Valid
Yes
No
Save the answer in the corresponding message in waiting_peer_announcements (if still exist)
Remove message from waiting_peer_announcements
Did all other subscribers already answer valid too?
No
Yes
Forward message to all known peers (Type 504)

Figure 1: Flow chart (v4) for Gossip. Colored boxes represent modules/classes. Note that the boxes are a simplification and some tasks may be in box A but use functionality of class represented by box B.

which is used in multiple parts of the program. All current classes and modules will be shortly introduced in the following.

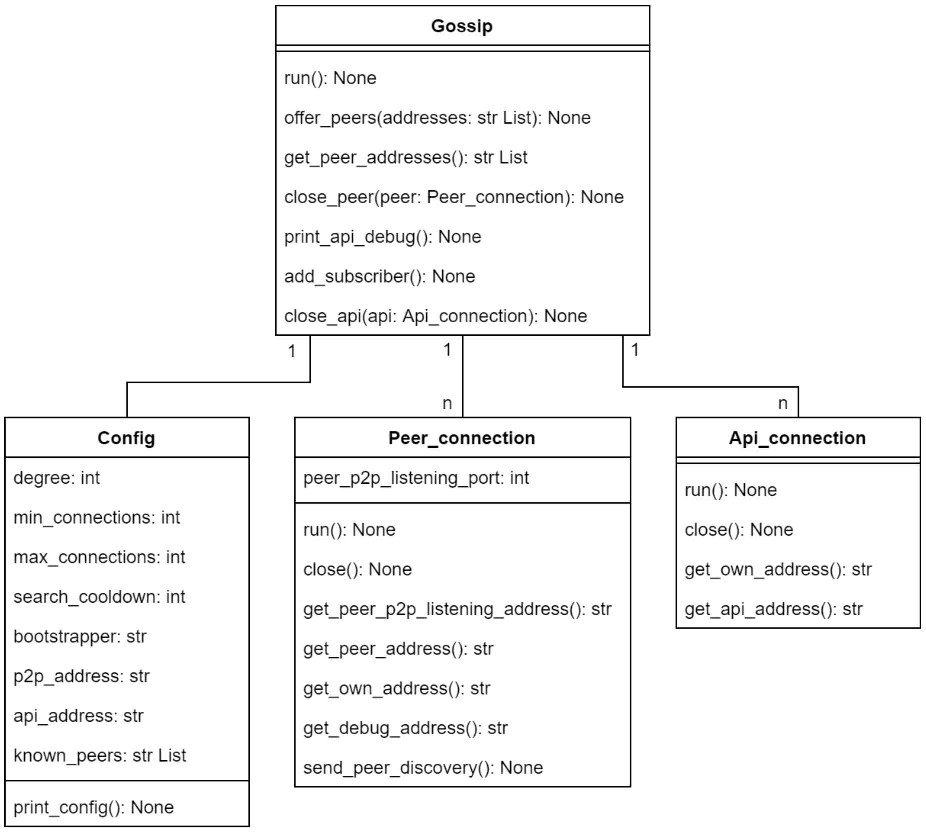

Figure 2: Gossip UML class diagram

2.1.1 Gossip class

An instance of the Gossip class represents an instance of the Gossip module. It manages the startup and searches for new peers if the amount of connected peers is bellow a specified number (see peer_control). Gossip knows all connected peers and API connections, which themselves use gossip to communicate with each other. It also knows about every datatype and subscriber. Logically it is at the very top of the hierarchy as it includes all the other classes.

2.1.2 Peer_connection class

An instance of Peer_connection represents a connection to another peer. The class provides the functions to communicate with the other peer and a run() function, which waits for incoming messages from the connected peer.

2.1.3 Api_connection class

Api_connection works similarly to its peer counterpart, representing an active connection to a single API.

2.1.4 Config class

The Config provides and abstraction for config files. It provides parsing and allows for quick access to config parameters. The Config module also contains a 'config_config' variable, which is used as a blueprint for parsing config files. It allows quick and easy addition of new parameters, setting default values and types.

Variables:

- `degree`: Number of peers this peer exchanges information with. Relevant for PEER ANNOUNCE.
- `min_connections`: Minimum amount of alive connections this peer should try to keep.
- `max_connections`: Maximum amount of alive connections this peer should try to keep.
- `search_cooldown`: In this interval it is checked whether we have `min_connections` peers. If not start search.
- `bootstrapper`: One trustworthy bootstrapping node, used as a fallback.
- `p2p_address`: Listening ip and port number for other Gossip peers.
- `api_address`: Listening ip and port number for API users.
- `known_peers`: List of peers to bootstrap through, if it is empty consult `bootstrapper`.

2.1.5 Connection handler module

The connection handler module is used to wait for and accept new incoming connections from other peers or API. For this, Gossip starts two separate tasks with the peer address or API address specified in the given config respectively.

2.1.6 Proof of work producer module

The proof of work producer module (pow_producer) provides the functionality required to solve proof of work challenges.

2.1.7 Packet parser module

The packet parser module is currently divided into two different modules, one for parsing and constructing API related messages and the other for parsing and constructing peer related messages. They will likely be merged in the final version.

2.2 Process architecture & Networking

As mentioned in 1.1 we changed our current implementation to exclusively use Pythons AsyncIO instead of threading. However, we may use threading or multiprocessing for generating proof of work in a future version.

2.3 Security measures

As soon as malformed messages from a peer are received we terminate the connection. We can exclude transmission errors as we use TCP.

3 Protocol/-s

Number	Name	Purpose
504	PEER ANNOUNCE	spreading of knowledge
505	PEER DISCOVERY	discovering new peers
506	PEER OFFER	spreading information about other peers
507	PEER INFO	tells peer which port is listening on my side

Table 1: Peer Message Overview

3.1 PEER ANNOUNCE

Figure 3 shows this message. Spreads the GOSSIP ANNOUNCE to peers. In case we receive a GOSSIP ANNOUNCE from an API user we transform it into this message and send it to our peers. In case we receive a PEER ANNOUNCE from a peer we keep spreading it in general. If we have a subscriber to this datatype, we transform the PEER ANNOUNCE into a GOSSIP NOTIFICATION. If we receive a negative PEER VALIDATION from the API that received the NOTIFICATION, we do not propagate it further.

The TTL field functions equally to the GOSSIP ANNOUNCE TTL field.

The id is a temporary addition from a previous sketch, we plan to rework it as soon as we further thought about how gossip is spreading its messages. It's whole purpose is to prevent message loops by using the id to remember that we already encountered this packet.

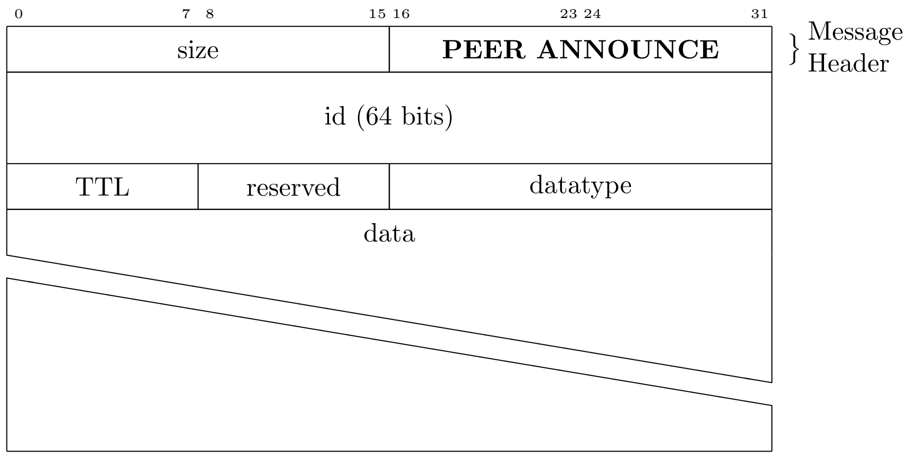

Figure 3: PEER ANNOUNCE message format

3.2 PEER DISCOVERY

Figure 4 shows this message. This and the next message (PEER OFFER) deal with the problem of keeping connection to degree amount of peers. We want to create a pool of peers, limited by max_connections in the config, from which we can choose degree many peers. Degree is also specified in the config.

To find degree many peers, do the following:

```
# n = amount of connected peers
while running
    if n < min_connections
        1. Send PEER DISCOVERY to all connected peers
        2. Peers respond (PEER OFFER) with a list
           containing all their known peers, add amount to n
        3. connect to peers in all given lists in a random order
           until we reach min_connections.
    sleep for search_cooldown seconds
```

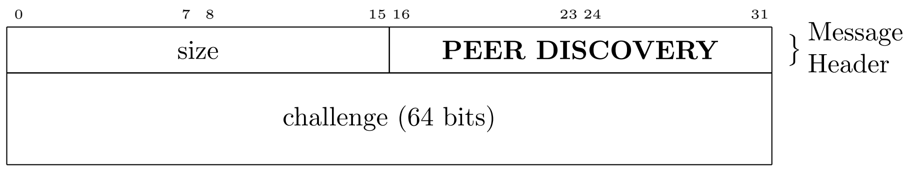

Figure 4: PEER DISCOVERY message format

3.3 PEER OFFER

Figure 4 shows this message. Answer to a PEER DISCOVERY message. Includes the senders peer list

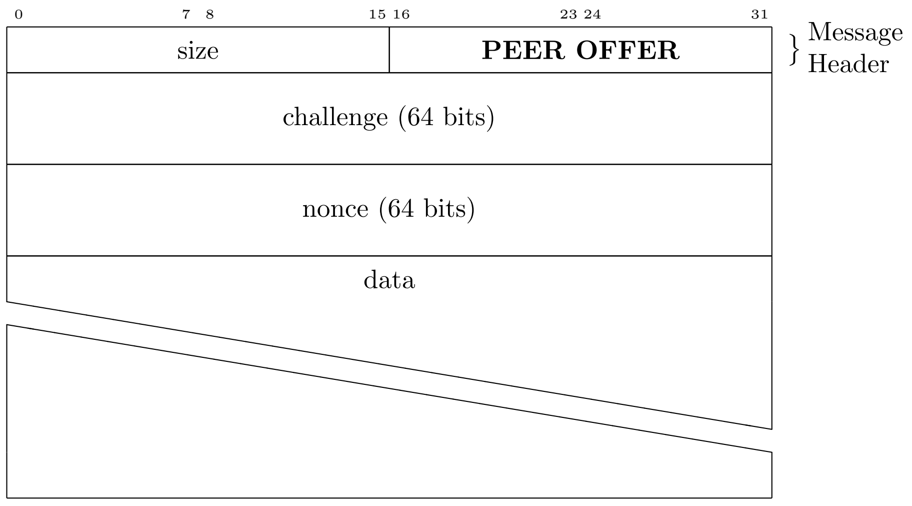

Figure 5: PEER OFFER message format

3.4 PEER INFO

Figure 4 shows this message. Informs the receiver of the senders p2p listening port for new incoming p2p connections. This is necessary since the port used to connect to the other peer is dynamic and not equal to the p2p address port. This port number is also spread further in the network through each peers list when answering with PEER OFFER to PEER DISCOVERY.

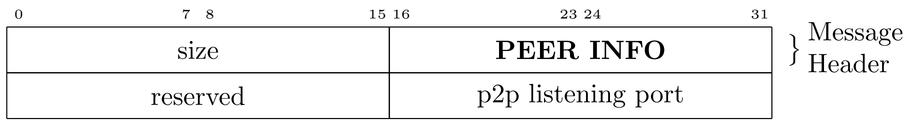

Figure 6: PEER INFO message format

4 Future Work

PoW based identities on parallel connection establishment when connecting to new peer (potencially given to us from one bad actor) to counteract Sybill and Eclipse attacks. Currently we have PoW in the request for new peers to neighbours, we plan to move it to the connection establishment.
Secure Channel between Peers: Pubkey exchange, Pubkeys as long-term keys

to this Identity, Session key establishment (DH), usage of AEAD for encryption and authentication of messages.
More tests in form of unittests for utility functions and whole system tests with mocks.

5 Workload Distribution

5.1 Liebald

- Peer_connection class implementation
- Packet parsing and generation for PEER DISCOVERY, PEER OFFER and PEER INFO
- Gossip class: basic structure and functionality related to Peer_connection
- Config class implementation
- Smaller modules: connection_handler and pow_producer

5.2 Ploch

- Api_connection class implementation
- Packet parsing and generation for GOSSIP ANNOUNCE, GOSSIP NOTIFY, GOSSIP NOTIFICATION and GOSSIP VALIDATION
- Gossip class: functionality related to Api_connection
- API Packet parser unit tests

6 Effort spent for the project

- Meetings/together: approximately 21 hours
- Liebald: approximately 30 hours
- Ploch: about 27-30 hours