Peer-to-Peer Networks: Gossip-5 Documentation

Lagemann, Thomas
Peuker, Petra

September 2022

1 Introduction

This documentation describes the python project Gossip-5 which implements the VoidPhone Project Specification in version 2022-0.1. The project and its documentation are part of the joint work of both authors.

2 Usage

In the following section we will introduce into the python application and how to run and test it. For direct execution please refer to the Quick Start Guide.

2.1 Requirements

To run Gossip-5, Python 3.10 is needed. To install Gossip-5 please clone the GitLab repository:

```
git clone https://gitlab.lrz.de/netintum/teaching/p2psec_projects_2022/Gossip-5.git
cd Gossip-5
```

2.1.1 Dependencies

You also need the `asyncio` python module. To execute the tests the modules `unittest` and `parameterized` are needed. To install all packages please run:

```
pip install asyncio unittest parameterized
```

2.2 Configuration

To configure the Gossip module, a .ini file should be provided with the following parameters:

- `cache_size`: Maximum number of data items to be held as part of the peer's knowledge base
- `degree`: Number of peers the current peer has to exchange information with
- `p2p_ttl` (optional): Specifies the maximum TTL for `GOSSIP_PUSH` updates. Default set to 5
- `bootstrapper` (optional): If this option is not provided, operate as a bootstrapper. Otherwise connect pee
- `p2p_address`:
- `api_address`:

An example is given below:

```
[gossip]
cache_size = 50
degree = 30
p2p_ttl = 10
bootstrapper = 127.0.0.1:6001
p2p_address = 127.0.0.1:6011
api_address = 127.0.0.1:7011
```

2.3 Execution

To execute Gossip-5 a path to an configuration file is allways required. It can be specified by the `-c PATH/TO/CONFIG.ini` parameter. The optional parameters for the validation timeout `-v INT` and the circular spread suppression time `-s INT` can be given in seconds. For detailed information, please refer to ??. Execution example:

```
cd src
python init.py -c ./config/bootstrap.ini -v 60 -s 300
```

2.4 Tests

To run the Tests please add the full path to Gossip-5/src to your PYTHONPATH:

```
TODO: Petra fuegt PYTHONPATH-Anpassung ein. Thomas kann kein Bash mehr, da er zu viel FISH gen
```

Then the tests can be executed by running:

```
cd tests/unit
python -m unittest
```

2.5 Quick start guide

To install the minimum requirements please run:

```
pip install asyncio
```

And to run the first client of a network:

```
cd src
```

And to run another client please execute:

```
python init.py -c ./config/peer1.ini
```

3 Project Structure

3.1 Project Files

```
Gossip-5
 ├─ docs
 │   ├─ documentation.pdf
 │   ├─ initial_report.md
 │   └─ midterm_report.md
 ├─ src
 │   ├─ config
 │   │   ├─ bootstrapper.ini
 │   │   ├─ peer2.ini
 │   │   ├─ peer3.ini
 │   │   └─ peer4.ini
 │   ├─ log
 │   │   └─ server.log
 │   ├─ connection.py
 │   ├─ goosip_logic.py
 │   ├─ init.py
 │   ├─ packing.py
 │   ├─ server.py
 │   └─ until.py
 ├─ tests
 │   └─ unit
 │       ├─ __init__.py
 │       ├─ helper.py
 │       ├─ test_connection.py
 │       ├─ test_gossip_logic.py
 │       ├─ test_packaging.py
 │       ├─ test_server.py
 │       └─ test_util.py
 └─ README.md
```

3.2 Classes

3.3 Architecture

We only support IPv4 addresses.

Incoming messages to the API server are directly passed to the Gossip Handler.

The P2P Server manages the peer's view and is responsible for spreading `PEER_NOTIFICATION` packages as well as forward them to the Gossip Handler. The peer view is represented by an `Connections` object. An entry of the view consists of an peer's IPv4 address and port as identifier. If there is an existing connection between the host and an peer, `Connections` also provides the current `asnycio StreamReader` and `StreamWriter` pair. According to the specification, 'A user is assumed to run a single peer at any time'. Thus, while a connection is alive, the reader-writer pair cannot be updated for a peer in the view.

Maintaining the Peer's View The peer's view should only contain peers with active connections to them. Thus, if we want to add a new peer, we establish a connection and introduce ourself with a `HELLO` message. Since the connection should be only active while there is a corresponding entry in the peer list, the peer to whom we want to connect (Peer2), should also add us to its view. To make DoS and Sybil Attacks more difficult, Peer2 sends us a nonce, we have to do a Proof of Work and send the result. If the result is verified, Peer1 is added to Peer2's view. In response to a correct challenge, we get Peer2's view. This protocol is also depictured in Figure 1.

Our view will be updated with peers from the `PEER_RESPONSE` as long as our view does not exceeds the limit, but at most with half of the limit. On a successful addition of a new peer, we spread a `PUSH` message containing the new peer. Peers may also try to establish a connection with `HELLO` with the spread peer.

If the number of items in the view is less than a specific threshold, we send a `PEER_REQUEST` message to a random neighbor. This neighbor answer us with `PEER_RESPONSE` and its view. If the P2PServer should spread messages, but the view is empty, the peer reconnects to the bootstapper.

If we have to remove a peer from the list, e.g. when the list size exceeds its limit, we choose a random peer and close the connection.

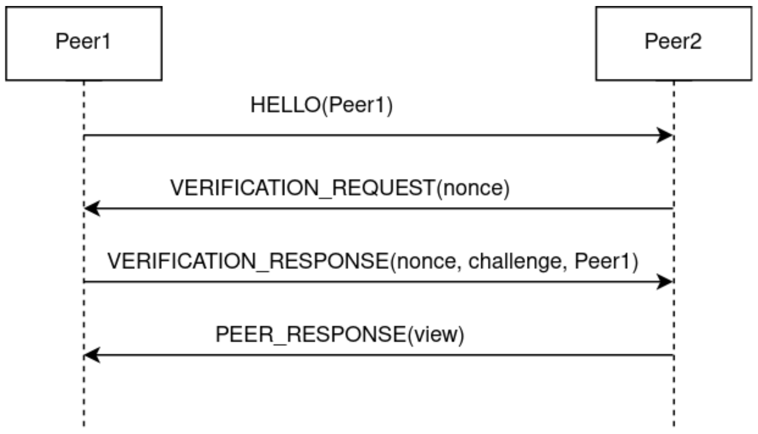
Peer1
Peer2
HELLO(Peer1)
VERIFICATION_REQUEST(nonce)
VERIFICATION_RESPONSE(nonce, challenge, Peer1)
PEER_RESPONSE(view)

Figure 1: HELLO handshake

3.4 Network Packages

3.4.1 GOSSIP ANNOUNCE

Message to spread data in the network. Is send from other modules to Gossip. For detailed information review the project specification.

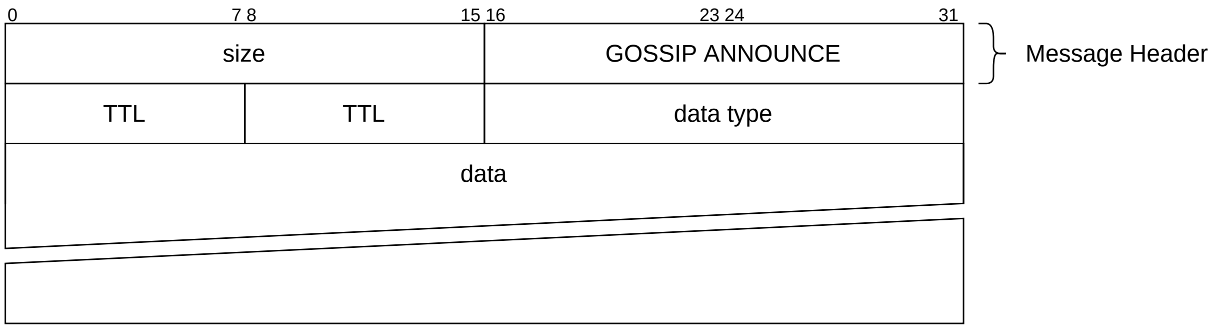

Figure 2: GOSSIP ANNOUNCE

3.4.2 GOSSIP NOTIFY

Message to subscribe for data that will be spread. Sent from other modules to Gossip. For detailed information review the project specification.

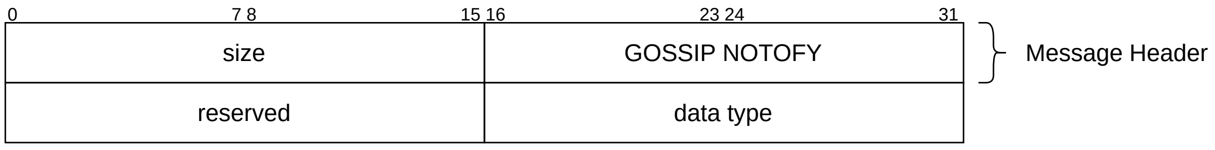

Figure 3: GOSSIP NOTIFY

3.4.3 GOSSIP NOTIFICATION

Message to hand knowledge over to modules. Sent from Gossip to Modules that subscribed previously via GOSSIP NOTIFY. The knowledge to forward can be received from modules via GOSSIP ANNOUNCE or from peers via GOSSIP PEER NOTIFICATION

3.4.4 GOSSIP VALIDATION

Message to inform Gossip about valid formatted messages. Sent from module to Gossip as response to GOSSIP ANNOUNCE and indicates whether to spread a GOSSIP PEER NOTIFICATION to the network.

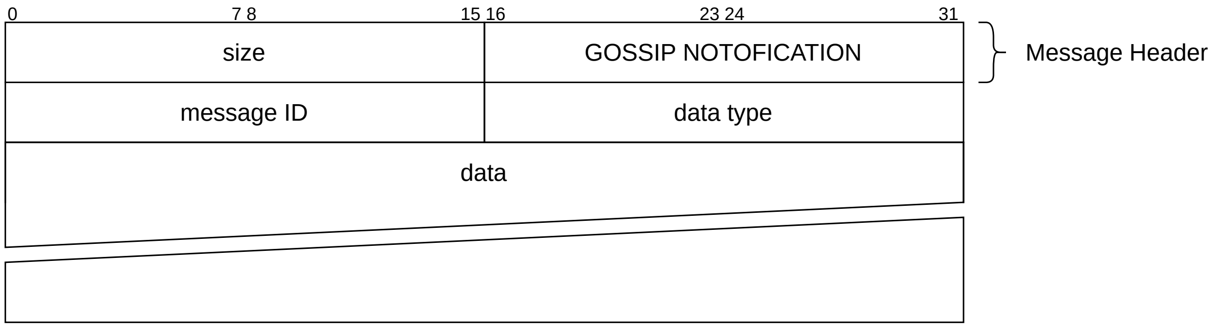

Figure 4: GOSSIP NOTIFICATION

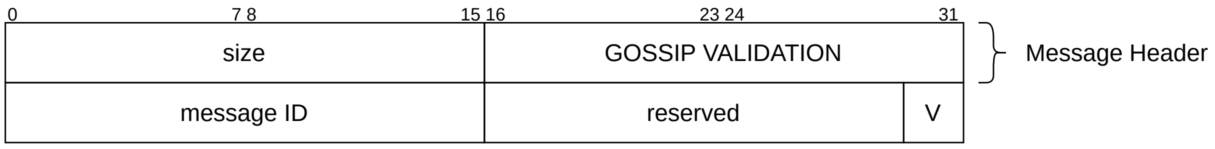

Figure 5: GOSSIP VALIDATION

3.4.5 GOSSIP PEER REQUEST

3.4.6 GOSSIP PEER RESPONSE

3.4.7 GOSSIP VERIFICATION REQUEST

3.4.8 GOSSIP VERIFICATION RESPONSE

3.4.9 GOSSIP HELLO

3.4.10 GOSSIP PUSH

3.4.11 PING

3.4.12 GOSSIP PEER NOTIFICATION

This message spreads knowledge between peers. It is send between Gossip modules. A received GOSSIP PEER NOTIFICATION will be spread to other modules via GOSSIP ANNOUNCE When the GOSSIP ANNOUNCE message has been verified by a module via GOSSIP VALIDATION and the TTL is higher than 0, the message will be spread in the network as new GOOSIP PEER NOTIFICATION message with reduced TTL. If the message ID of an incoming GOSSIP PEER NOTIFICATION was received and is already waiting for validation or was spread recently than the message will be dropped. The duration until those message IDs expire can be set with the validation time and the spread suppress time.

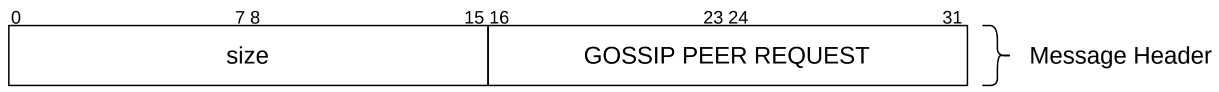

Figure 6: GOSSIP PEER REQUEST

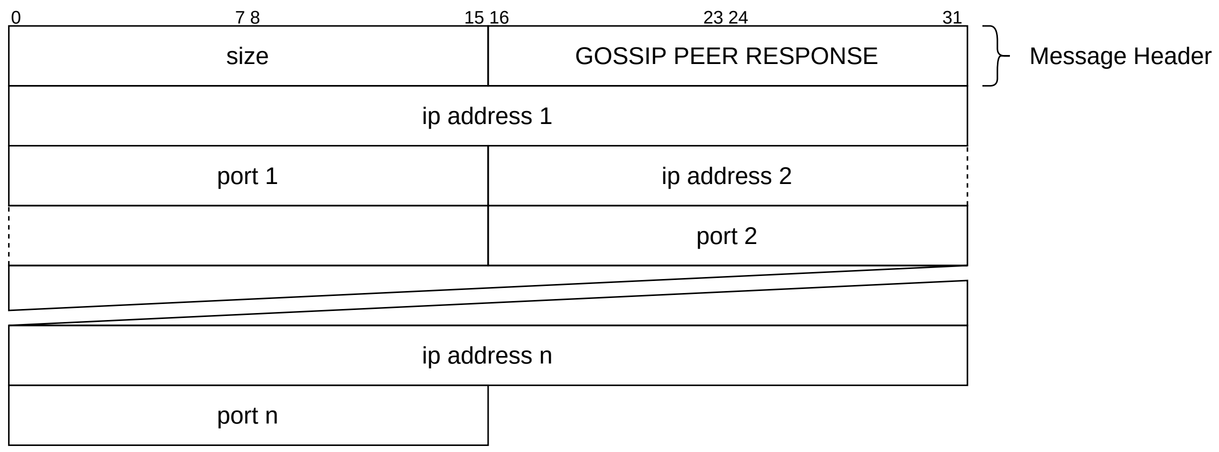

Figure 7: GOSSIP PEER RESPONSE

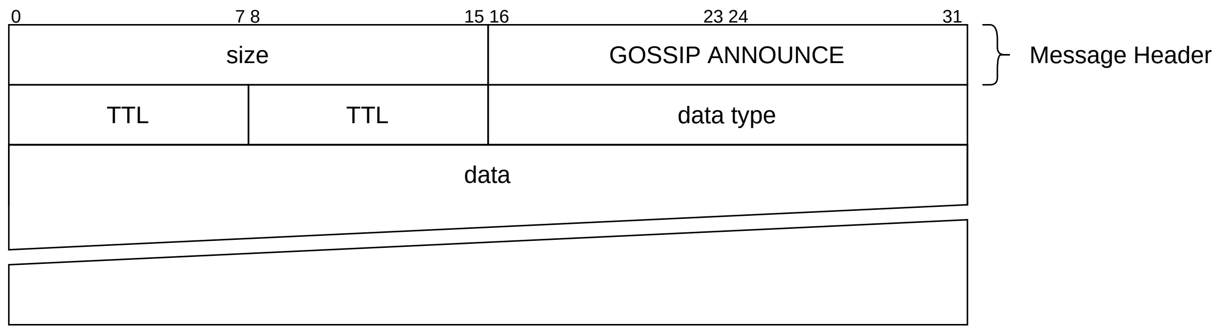

Figure 8: GOSSIP VERIFICATION REQUEST

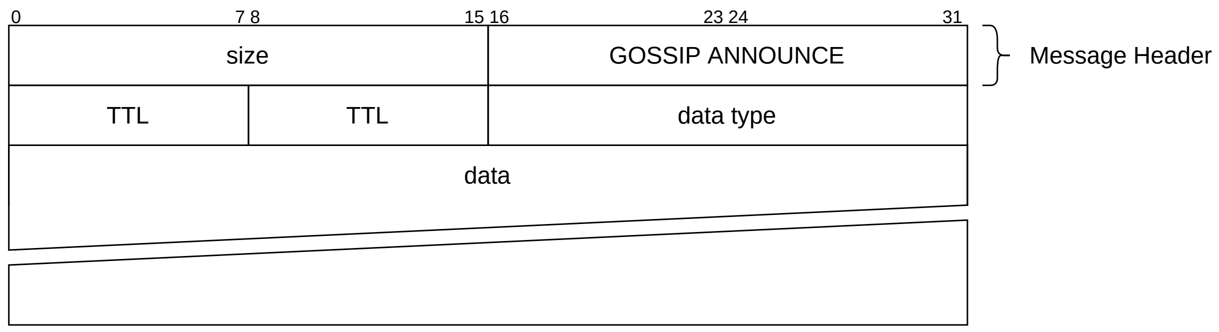

Figure 9: GOSSIP VERIFICATION RESPONSE

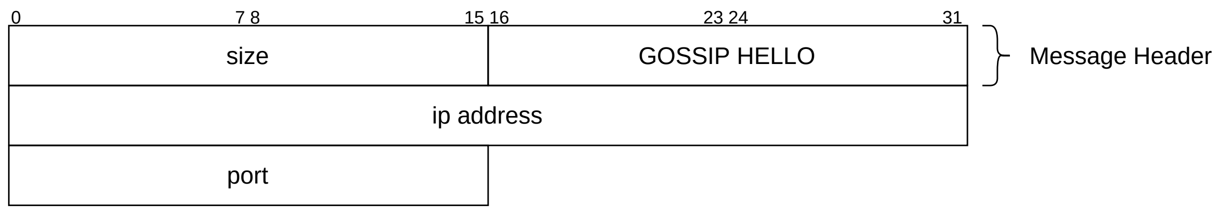

Figure 10: GOSSIP HELLO

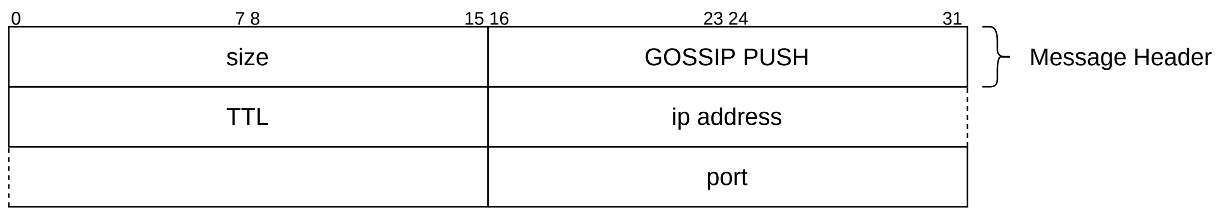

Figure 11: GOSSIP PUSH

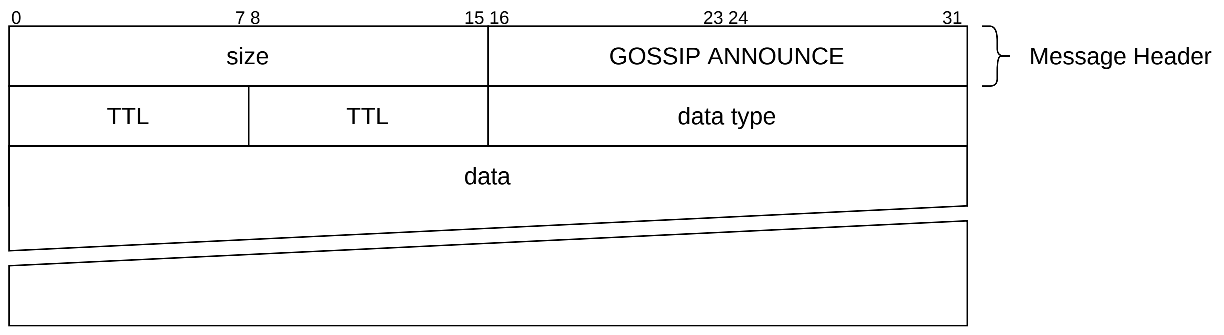

Figure 12: PING

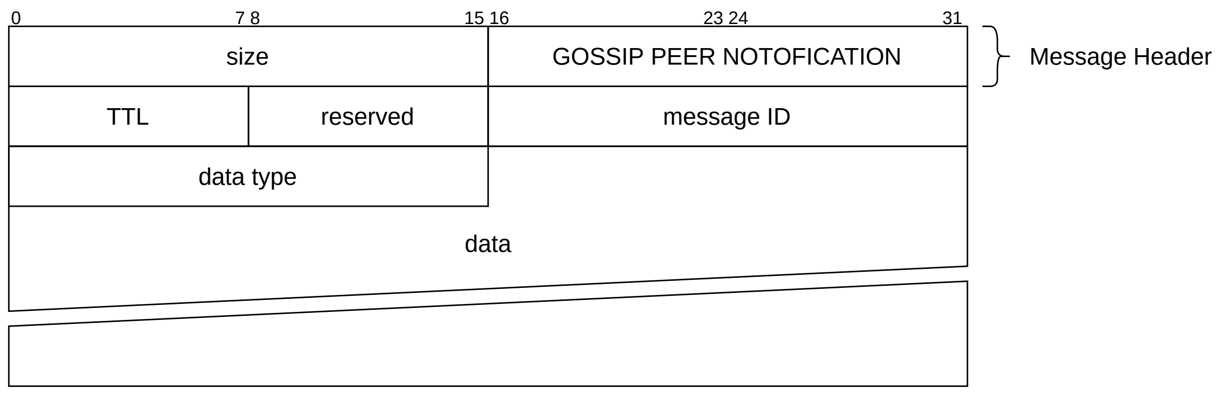

Figure 13: GOSSIP PEER NOTIFICATION

4 Security

5 Future Work

5.1 Security

5.2 Extension

5.3 Optimization

6 Workload Distribution

6.1 By Structure

6.2 By Time