Final Report

Peer-to-Peer Systems and Security

Network Size Estimation

Group 37
Dominik Winter
Stefan Armbruster

1 Program Documentation

1.1 Program Dependencies

This program was developed with *Python 3.6* and *Ubuntu 18.04 LTS*.
The only external dependent library that is used by the program is `pycrypto`.
All other libraries are standard libraries of *Python 3.6*.

In order to build a virtual environment with the dependent library a build script was added to the project which is further explained in section 1.2.

To install any additional library please execute:
`pip3 install --user <library>`.

1.2 Program Setup

The project directory is set up as follows:

```
voidphone
├── build.sh
└── implementation
    ├── api_message.py
    ├── api_server.py
    ├── asym_crypto.py
    ├── config.ini
    ├── gossip.py
    ├── hostkey.pem
    ├── nse.py
    ├── parser.py
    ├── pow.py
    ├── time_delay.py
    └── tests
        ├── config_error.ini
        ├── config.ini
        ├── hostkey_different.pem
        ├── hostkey.pem
        ├── nse_query.py
        └── tests.py
```

In order to run the program *Python 3.6* the additional library `pycrypto` must be installed locally.

In addition, a build script was written to simplify the build process which can be executed with: `source build.sh`.
The script activates a virtual python environment with the `virtualenv` library and installs the required `pycrypto` library. At the end of the script all unittests in `implementation/tests/tests.py` are executed.

The program needs a hostkey file and a config file, which can be optionally specified via command line options.
Moreover, before executing the program a gossip module must be set up on the specified port from `config.ini`.

After these steps the program can be executed with:
`python3 implementation/nse.py`

To run the unittests please execute:
`python3 implementation/tests/tests.py`

The usage of the program with all available command line options is shown as follows:

```
Usage: nse.py [-h] [-c CONFIG] [-k HOSTKEY]

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        Path to config file
  -k HOSTKEY, --hostkey HOSTKEY
                        Path to hostkey file
```

1.3 Issues

Our program has been tested with several numbers of peers up to 100 and always produced accurate estimates within an acceptable range. However, due to memory limits, we were not able to test the program with a network greater than 100 peers. There are no known issues of the program so far. We added debug messages to many functions to show the program flow and a lot of exception messages have been added in order to output more specific error messages.

2 NSE Protocol

For our protocol we chose GNUnet's implementation of the network size estimation [1] as it has several benefits compared to other implementations. GNUnet NSE requires only $O(1)$ state per peer and sends controlled floods of $O(|E|)$ messages per round. In addition, it is more secure than other implementations by requiring Proof-of-Work-based peer identities as well as signatures attached to every protocol message.

In short, all peers that use the GNUnet NSE algorithm compute a distance estimation at a frequency of f, by comparing the own identity with a random key T and comparisons of other peers received over the last few iterations. The first matching bits of the peer identity and the random key T are counted and compared to the proximity calculated by other peers. The network size is then derived by computing $2^{p-0.332747}$, where p is the highest proximity of matching bits in the current round.

The message format used in our implementation of the GNUnet NSE algorithm is explained in the following:

```
{
 "Hop-Count": Number,
 "Round": String,
 "Proximity": Number,
 "Pub-key": String,
 "PoW": {
         "Time": String,
         "Random-number": Number,
         "Hash": String
        },
 "Sign": String
}
```

Figure 1: Message Format

- **Hop-Count:** Contains information about the number of peers that forwarded the message.
- **Round:** Contains the time of the corresponding round of the NSE.
- **Proximity:** Contains the amount of equal bits of the random key and the peer identity.
- **Pub-key:** Contains the public-key of the peer which sends the message.
- **PoW:** Contains the proof of work which consists of a time stamp, a random number and a hash.
- **Sign:** Contains the signature, which is used to authenticate the sender.

We had to add the *Hop-Count* field during our developing, as we needed it to calculate an accurate process delay. This was the only change we made compared to our message format in the interims report.

3 Future Work

As described in the interims report, we used the GNUnet's implementation of the network size estimation [1]. In order to create a usable program, the main elements of the paper were implemented.
These include the generation of random keys and their respective proximities for the individual peers, the calculation of a peer estimate, the delay for broadcasting own messages, the delay for forwarding received messages as well as the proof of work approach. Therefore, our NSE-module is already usable.
However, in order to receive better results there are still further approaches that can be implemented.

3.1 Exchange of the History

If a peer joins the network, it lacks information about the previous rounds. Hence, it would be beneficial if an adjacent peer sent its history to the new joining peer.
Our approach includes that new joining peers would first send a message containing a request of the message history to their neighbours. Subsequently, the neighbours would answer by sending their history to the new peer. As a last step of the bootstrapping process the new peer would use

the received history as its own and would be able to calculate useful delays and standard deviations.
The request for history messages and their answer messages would need own identification numbers. Possible numbers would be 531 for the history request message type and 532 for the history answer message type. The answer message would additionally use the public-key of the request message to address the respective peer which requested the history.

History Request:

```
{
 "Pub-key": String,
 "PoW": {
         "Time": String,
         "Random-number": Number,
         "Hash": String
        },
 "Sign": String
}
```

Figure 2: History Message Request

- **Pub-key:** Contains the public-key of the peer which sends the message.
- **PoW:** Contains the proof of work which consists of a time stamp, a random number and a hash.
- **Sign:** Contains the signature, which is used to authenticate the sender.

History Answer:

```
{
 "Peer-key": String,
 "AmountOfMessages": Number,
 "History": String,
 "Pub-key": String,
 "PoW": {
         "Time": String,
         "Random-number": Number,
         "Hash": String
        },
 "Sign": String
}
```

Figure 3: History Message Answer

- **Peer-key:** Contains the public-key of the peer which sent the request and receives the answer.
- **AmountOfMessages:** Contains the amount of messages in the history.
- **History:** Contains a string represenation of the history.
- **Pub-key:** Contains the public-key of the peer which sends the message.
- **PoW:** Contains the proof of work which consists of a time stamp, a random number and a hash.
- **Sign:** Contains the signature, which is used to authenticate the sender.

3.2 Handle Messages with Wrong Time

An additional problem might be time differences between peers. Our current approach handles messages with a wrong round time in dependence to the magnitude of the time difference. If the message is from more than one round before or after, the message is ignored. If the message is from the next round, it is saved for that round, whereas if it is from the previous round, it is compared with the history message from the corresponding

round. Thereby, the history message is updated if the received message has the better proximity.
A further problem are small time differences within a round, as it leads to wrong timed announcements. For that purpose, a time message would be helpful as the time could be adjusted to other peers in the network.
If a peer receives a message with a wrong time, a wrong-time-message might be sent. A possible identification number would be 533.

Wrong-Time-Message:

```
{
 "Peer-key": String,
 "Current-time": String,
 "Receiving-time": String
 "Wrong-peer-time": String,
 "Pub-key": String,
 "PoW": {
         "Time": String,
         "Random-number": Number,
         "Hash": String
        },
 "Sign": String
}
```

Figure 4: Wrong-time-message

- **Peer-key:** Contains the public-key of the peer which sent a message with the wrong time.
- **Current-time:** Contains the current time of the own peer.
- **Time-delta:** Contains the time difference of the received and the own time.
- **Wrong-peer-time:** Contains the received wrong time of the other peer.
- **Pub-key:** Contains the public-key of the peer which sends the message.

- **PoW:** Contains the proof of work which consists of a time stamp, a random number and a hash.
- **Sign:** Contains the signature, which is used to authenticate the sender.

If a peer with a wrong time received such a message, it would wait for messages from other peers with a similar content. In order to avoid an attack, it is necessary to wait for more than one wrong-time-message. If the current-times of the other messages were almost equal, than the own time would be adjusted.
Thereby, a timedelta balancing out the time differences could be created. For that purpose, the minimum of the time-deltas from the wrong-time-messages could be used. This time-delta would correct the wrong time of the peer, resulting in less messages, as the time-delta and the delay would inhibit messages with small proximities to broadcast too early.

4 Work Distribution

4.1 Individual Effort

Description	**Dominik (hours)**	**Stefan (hours)**
Read and understand GNUnet's NSE protocol	10	10
Implement *config.ini* parser module	4	0
Understand and implement asyncio architecture	15	5
Understand and implement necessary asymmetric crypto module	12	10
Understand and implement gossip message handling (asyncio loops)	20	20
Understand and implement api server and query handling	10	2
Understand and implement proof of work module	8	6
Understand and implement nse handler module (validation and updating of messages)	22	20
Understand and implement time delay module	0	12
Implement unittests	10	8
Refactoring of codebase	12	5
Testing and debugging in gossip-testing branch	25	25

Table 1: Individual Effort

4.2 Team Effort

Description	**Effort in hours**
Initial Report	3
Interims Report	4
Final Report	8

Table 2: Team Effort

References

[1] N. Evans, B. Polot, C. Grothoff, L. Kencl, L. Li, J. Widmer and H. Yin. *Efficient and Secure Decentralized Network Size Estimation.* In: *Networking 2012.* (2012), pp. 304-317. ISBN: 978-3-642-30045-5.