Peer-to-Peer Systems and Security - Project Documentation

Kevin Burton

1 Licensing

The licensing of the implementation is stated in the LICENSE file. Contrary to the initial report, where it was planned to release using a CC0 license, the MIT license is employed. A decision was made against CC0 and in favour of MIT to preserve the original copyright with mention of NetInTum and the peer to peer security course, especially since the gossip dummy and its corresponding utility file is not code developed by me.

2 Architecture and Protocols

This section is very similar to that of the same name in the midterm report, with a few updates. No major changes to the architecture and protocol were made and the message formats are all identical.

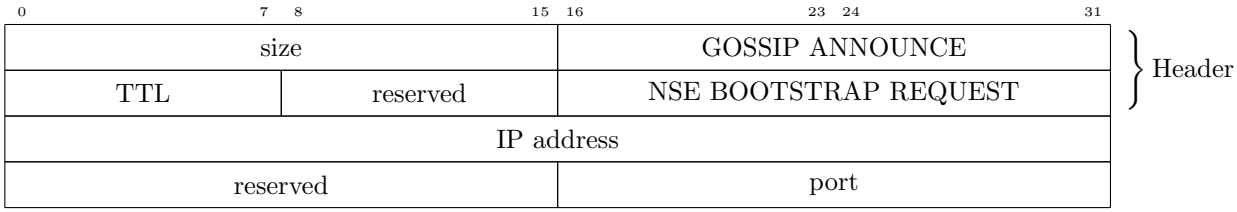
The module architecture is quite simple: the `nse` file is responsible for bootstrapping and everything protocol related, namely receiving and sending messages, while the `nse_util` file is used to outsource frequently used calculations such as getting an estimate from a leading bit proximity and calculating proof of work.
The NSE protocol itself will be explained using the messages sent throughout. It is an implementation of GNUnet's NSE which is described in Nathan Evans 2012, with a few compromises.

2.1 Bootstrapping

First of all every NSE module has to subscribe to GOSSIP ANNOUNCE messages of data type NSE BOOTSTRAP REQUEST by sending a GOSSIP NOTIFY to gossip. Therefore every peer in the network is able to respond to a new node requesting its first estimate. This request is sent out right after the subscription and is further specified in the following subsection.
Furthermore an event loop is set up at the beginning of module execution, on which a couple of tasks are created to run asynchronously: the server which listens for incoming messages, the round coroutine function, which loops indefinitely and is responsible for starting new rounds and getting an estimate, and a function that listens for new messages on a stream represented by a reader/writer pair with gossip. The latter therefore handles GOSSIP NOTIFICATION messages.

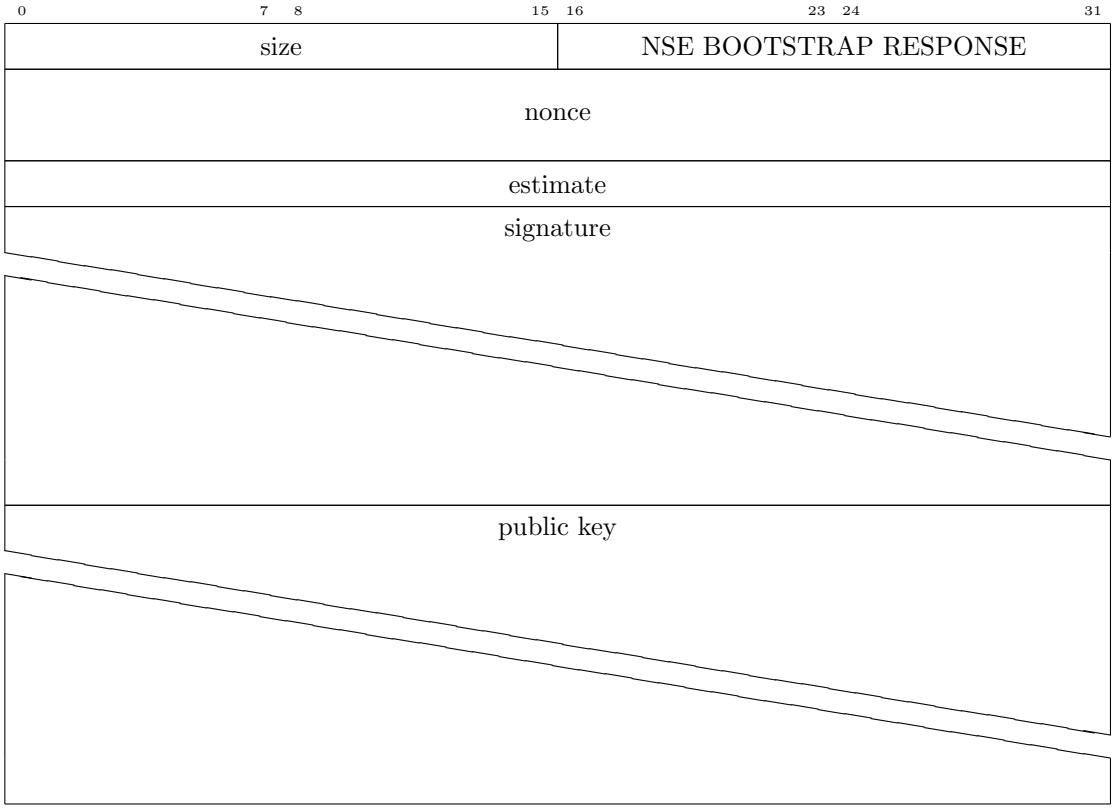
2.1.1 Bootstrap Request

The bootstrap request announcement is sent to the local gossip instance at the start of module execution so that the NSE module has a first estimate to work with. In addition to the regular GOSSIP ANNOUNCE header fields it also contains the senders IP address and the port on which the NSE module can be reached.



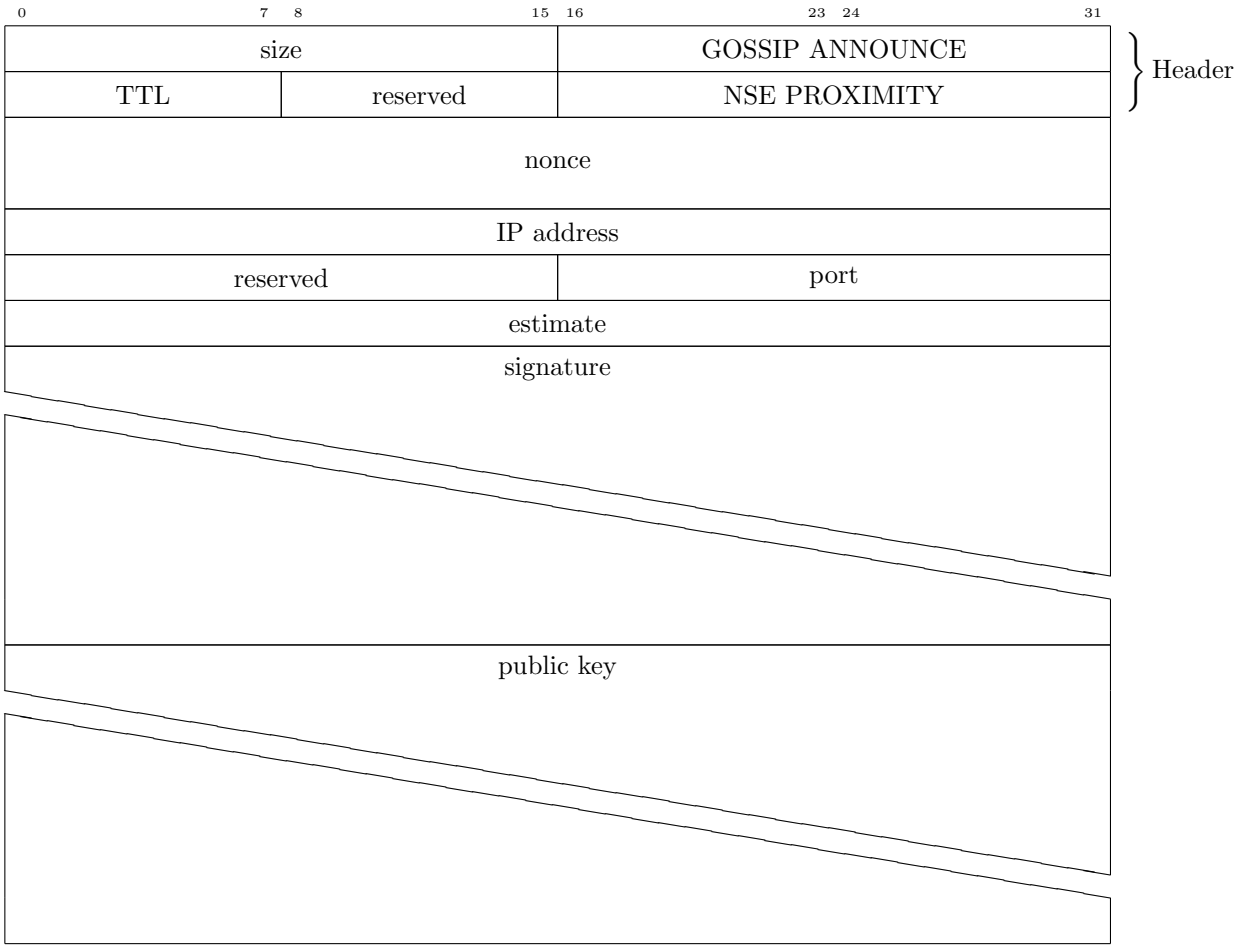
2.1.2 Bootstrap Response

Whenever an instance of an NSE module receives an announcement with NSE BOOTSTRAP REQUEST datatype, it answers with the following message mainly containing an estimate to bootstrap with. The random nonce provides proof of work, while the signature provides authenticity and integrity, which can be verified using the public key. The proof of work is achieved by a certain number of overlapping leading bytes between a hash of the data of the message and the node identifier, which is a hash of its public key.



2.2 Announcing and Relaying an Estimate

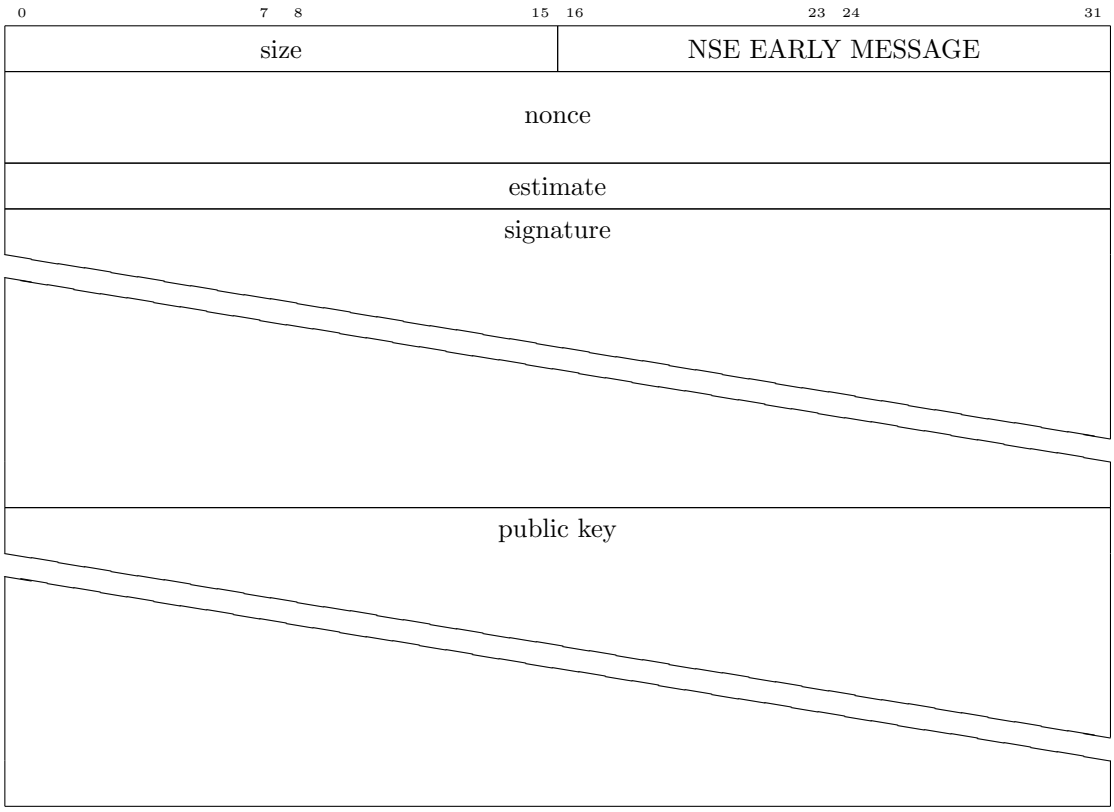
At the beginning of a round every node calculates a proximity by determining the number of overlapping leading bits between a random key and the node identifier. The maximum expected proximity in the network can then be translated to an estimate of how many peers are participating in the network. The calculation and proof is further explained in Nathan Evans 2012, section 3.
From the locally calculated estimate a node can then determine when to announce it to its peers. Lower estimates are announced later in the round. The exact method of calculating this waiting time is explained in Nathan Evans 2012, section 3.2. When this waiting time ends the NSE module announces its estimate using the following message:



IP address and port are necessary to be included in the announcement so if another node notices that the message was sent too early for the estimate it transported, it can answer with the message type presented in section 2.3. Nonce, signature and public key are included for the same reason as in the BOOTSTRAP RESPONSE message type.
If the module receives such a message it has to compare the received estimate with the highest estimate of the round so far. In case it is higher, the above message is not transmitted with the local module's calculated estimate and instead the received message is relayed. Contrary to the midterm report, where the parameters in the message to be relayed are replaced with our own, now the message is sent as it was received. This is because otherwise the verification described in section 3 would fail. The other case is that a lower estimate is received, which indicates clock skew in the sending node. This leads to a sending of the message type shown in the following subsection.

2.3 Answering Clock Skew

This message is received when another node has detected that a message was sent too early, indicated by an estimate that was too low to be sent that early in the round. It mainly contains a proper estimate which can be immediately used to update the current one within the receiving node. Again, a nonce is included for proof of work, as is a signature and the corresponding public key for verification purposes.



3 Countermeasures

The NSE also has the possibility of an attacker in the network trying to skew the overall estimate. This includes trying to lead the network into both significantly over- or underestimating the network size. This is explained more in depth in Nathan Evans 2012, section 4. One specific countermeasure to reduce the attack surface in addition to signing messages and proof of work is to verify that a received estimate was generated using the nodes identifier and the key from the current round. This is done for every received estimate excluding the bootstrap estimate, since when the NSE is being bootstrapped, it does not yet have a round key to verify with.
In addition to the round key of the currently running round being stored and used for verification, the last round's key is saved for the same purpose. In the case that every node in the network got an estimate that is significantly lower than the last round's estimate, every peer in the network will announce theirs very late into the round. This entails that the notifications from gossip containing those estimates could be received well into the next round, when a new round key was already generated. Then the old round's key will be used for verification to then be able to update the last round's estimate.

4 Testing Scenario

Alongside the NSE a testing scenario assessing all its functionalities is provided. It is comprised of a testing module, a bash script and a gossip dummy.

4.1 Testing Module

The testing module is the main part of the testing scenario. Its implementation is very similar to that of the NSE itself and it behaves in the network like an instance of NSE. The main difference between the two is that the testing module simulates a full network of peers behind it and that instead of participating in indefinite rounds of estimation, it has a regimented procedure of a set amount of testing rounds.
During bootstrapping it behaves similarly to the NSE by subscribing to the gossip notifications regarding new estimates and bootstrapping requests by sending gossip notifies. It however does

not send out an announcement to request a bootstrap estimate, as it already knows the exact number of peers in the network, since it has to simulate them. For this simulation it is necessary to generate a new key pair for each and every participant, which is fairly demanding on the host. To speed up this process, multiprocessing is employed. It is recommended to drop the amount of peers in the simulation if the scenario is being executed on a less powerful host, which is explained how to do in the repository's README. After that the module awaits the startup of the NSE and begins with the testing rounds afterwards. For every new round a new number of peers in the network is randomly generated according to a Gaussian distribution with the mean being the amount of peers that participated in the last round. It can therefore happen that new peers join the simulated network, which means that new keys have to be generated. The module then calculates the proximity of each peer and finds the maximum, which then represents the estimate that is to be gossiped. While waiting to gossip, a message of type NSE QUERY is sent to the NSE to query the estimate it stored in the last round (or after bootstrapping, if it is the first round). This estimate and the actual number of peers are stored. All of this is repeated for a number of rounds which is specified in the configuration file.
Afterwards the aforementioned estimates and number of peers are plotted against each other in a chart, which is further explained in section 5.3.3. Finally the testing module announces a BOOTSTRAP REQUEST to check if the bootstrapping of the NSE is working as intended.

4.2 Bash Script

To be able to execute the testing scenario, a bash script called `test.sh` is provided. It simply starts the python files in parallel in a given order. The gossip module is started first with the test module following shortly. After a delay of 20 seconds the NSE module is added to the execution. This delay is added to give the test module enough time to generate the keys for its simulated peers.

4.3 Gossip Dummy

The gossip dummy is implemented in the `gossip_mockup.py` and `gossip_util.py` files. These were provided by the team behind the peer to peer security course. Only a few small changes were made to the gossip dummy: the outputs were modified slightly to be able to better identify the module that is printing, and the name of the utility file was changed to its current name from `util.py` to be able to better discriminate between the two utility files in the project.

5 Software Documentation

5.1 Installation and Running

Instead of using Setuptools to build the project as described in the initial report, a bash script called `test.sh` is used to run the testing scenario. It is built and distributed using Docker. Information on how to run the NSE module and how to build and run the testing scenario with Docker is provided in the `README.md` document inside the repository.

5.2 Known Issues

The main known issue with the NSE itself is that it doesn't implement the peer specific random delay as proposed by Nathan Evans 2012 in section 3.3. This entails that any announcement of a new high estimate will incur a flood of messages as every peer relays that estimate. Another problem is that the NSE doesn't yet support IPv6.
Additionally there are a few known issues with the current implementation of the testing scenario. The first is that if the test is started close to the beginning of a round, it will start before the test module has finished generating the keys, resulting in it missing the first round. This is more likely to occur on less powerful computers. Announced estimates from the NSE will then be discarded by the testing module because they were determined using a round key that the testing module never calculated. The NSE will then write its likely low estimate to storage. It will then later be queried and received by the testing module, where normally the estimate that the NSE was bootstrapped with would be expected. Apart from creating this discrepancy in the first estimate, which is then

reflected in the chart, no further problems arise from this issue.
Similarly, between two rounds if the churn in the network is especially high and many new peers join, many new keys have to be generated. This can then lead to the test module missing a round entirely, with the similar effect that one rounds estimate is skewed into being much lower than expected. If this occurs it is recommended to lower the amount of peers or increase the time between rounds.
Furthermore, an issue exists with both the testing module and the NSE: when receiving a notification with a new estimate, a GOSSIP VALIDATION is sent on the writer connected to gossip. Shortly after, in case the estimate was higher than the locally calculated estimate, an announcement relaying it is sent on the same writer. If gossip receives the validation slowly, it is possible for the start of the announcement to be included in the validation, resulting in an unexpected message on gossip's side. Currently the modules sleep for a short amount of time before announcing to mitigate this issue, but this is no ultimate solution. Having a separate writer to send validations is unfortunately not possible, since gossip expects the validation to come from a reader/writer pair representing a connection that is subscribed to the type of notification that is being validated.

5.3 Understanding the Output

In this section the output of the testing scenario is explained. Since the NSE module prints the same outputs when it is run in isolation, this explanation covers this case as well. The module that is the originator of the output always precedes the line in square brackets. Messages received and sent over the network are annotated with the port and IP of the sender or target. Arrows pointing right indicate a message received by the printing module, whereas arrows pointing left mean that a message is being sent out.

5.3.1 Bootstrapping

At the very start of the execution the modules output on which ports they are communicating with whom and where they are listening for new messages. During this phase the bootstrapping of the testing module and the NSE is performed, which includes generating keys for a specified number of peers for the test and announcing a bootstrap request and receiving a bootstrap estimate for the NSE. In this case 32 peers are part of the network simulated by the testing module, resulting in a bootstrap estimate of 33 including the NSE.

```
[Gossip] GOSSIP mockup listening on 127.0.0.1:7001
[Gossip] 127.0.0.1:46846 >>> Incoming connection
[Test] Communicating with Gossip on ('127.0.0.1', 46846) and ('127.0.0.1', 46848)
[Gossip] 127.0.0.1:46846 >>> GOSSIP_NOTIFY registered: (524)
[Gossip] 127.0.0.1:46848 >>> Incoming connection
[Gossip] 127.0.0.1:46848 >>> GOSSIP_NOTIFY registered: (522)
[Test] Listening on 127.0.0.1:7202
[Test] Generating keys for 32 peers, stand by...
[Gossip] 127.0.0.1:46850 >>> Incoming connection
[Gossip] 127.0.0.1:46850 >>> GOSSIP_NOTIFY registered: (524)
[Gossip] 127.0.0.1:46852 >>> Incoming connection
[Gossip] 127.0.0.1:46852 >>> GOSSIP_NOTIFY registered: (522)
[NSE] Communicating with Gossip on ('127.0.0.1', 46850) and ('127.0.0.1', 46852)
[NSE] Listening on 127.0.0.1:7201
[NSE] 127.0.0.1:7001 <<< GOSSIP_ANNOUNCE | NSE_BOOTSTRAP_REQUEST
[Gossip] 127.0.0.1:46850 >>> GOSSIP_ANNOUNCE: (524)
[Gossip] 127.0.0.1:46846 <<< GOSSIP_NOTIFICATION(32895, 524)
[Test] Finished key generation, waiting for NSE...
[Test] 127.0.0.1:7001 >>> GOSSIP_NOTIFICATION | NSE_BOOTSTRAP_REQUEST
[Gossip] 127.0.0.1:46846 >>> GOSSIP_VALIDATION: (32895:True)
[Test] 127.0.0.1:7201 <<< NSE_BOOTSTRAP_RESPONSE
[Test] NSE online, beginning with test
[NSE] 127.0.0.1:45536 >>> NSE_BOOTSTRAP_RESPONSE
[NSE] Got a bootstrap estimate of 33
```

At the very end of the execution the test module announces a bootstrap request to test the functionality in the NSE. In this case the NSE answers with an estimate of 101. Afterwards the testing protocol is finished and the scenario has to be terminated by pressing CTRL-C, otherwise the gossip and NSE modules will continue to run indefinitely. The user is reminded of the chart,

where the estimates and the actual amount of peers are plotted against each other, which is further elaborated on in section 5.3.3.

```
[Test] Announcing testing bootstrap request
[Gossip] 127.0.0.1:46846 >>> GOSSIP_ANNOUNCE: (524)
[Gossip] 127.0.0.1:46850 <<< GOSSIP_NOTIFICATION(33365, 524)
[NSE] 127.0.0.1:7001 >>> GOSSIP_NOTIFICATION | NSE_BOOTSTRAP_REQUEST
[Gossip] 127.0.0.1:46850 >>> GOSSIP_VALIDATION: (33365:True)
[NSE] 127.0.0.1:7202 <<< NSE_BOOTSTRAP_RESPONSE
[Test] 127.0.0.1:36970 >>> NSE_BOOTSTRAP_RESPONSE
[Test] Bootstrap response with estimate 101 received
[Test] Testing protocol finished, estimations compared to the actual values can be
    seen in chart.png, hit CTRL-C to terminate
```

5.3.2 Testing Rounds

During a round the output could look like the following:

```
[Test] Starting round 2 of 5 with 32 peers
[Test] Estimate for round 2: 25
[Test] 127.0.0.1:7201 <<< NSE_QUERY
[NSE] Time to next round: 50 seconds
[NSE] 127.0.0.1:45538 >>> NSE_QUERY
[NSE] 127.0.0.1:45538 <<< NSE_ESTIMATE(Estimate: 25, Std deviation: 4)
[Test] 127.0.0.1:7201 >>> NSE_ESTIMATE
[Test] Time to next round: 37 seconds
[Test] 127.0.0.1:7001 <<< GOSSIP_ANNOUNCE | NSE_PROXIMITY
[Gossip] 127.0.0.1:46848 >>> GOSSIP_ANNOUNCE: (522)
[Gossip] 127.0.0.1:46852 <<< GOSSIP_NOTIFICATION(62685, 522)
[NSE] 127.0.0.1:7001 >>> GOSSIP_NOTIFICATION | NSE_PROXIMITY
[NSE] Writing estimate of 25 to storage
[NSE] 127.0.0.1:7001 <<< GOSSIP_ANNOUNCE | NSE_PROXIMITY
[Gossip] 127.0.0.1:46852 >>> GOSSIP_VALIDATION: (62685:True)
[Gossip] 127.0.0.1:46852 >>> GOSSIP_ANNOUNCE: (522)
[Gossip] 127.0.0.1:46848 <<< GOSSIP_NOTIFICATION(64207, 522)
[Test] 127.0.0.1:7001 >>> GOSSIP_NOTIFICATION | NSE_PROXIMITY
[Gossip] 127.0.0.1:46848 >>> GOSSIP_VALIDATION: (64207:True)
```

At the beginning the test prints the number of the current round relative to the number of total rounds, and how many peers are participating in it. The testing module then outputs the max estimate it got from its simulated participants. Afterwards an NSE QUERY is sent to the NSE. It then replies with the last round's estimate, which in this case is also 25. Furthermore, the time from now until the start of the next round is printed. The test then announces its estimate after waiting for a calculated time. Gossip sends a notification to the NSE, which writes the new estimate to its own storage. It then relays the new estimate by announcing a NSE PROXIMITY.

5.3.3 Chart

After the testing scenario terminates a chart is available inside the root of the repository. This chart plots the estimates that the NSE wrote into its queue against the actual number of peers that were simulated by the testing module over the testing rounds. A sample chart is given in figure 1.

6 Future Work

The most obvious improvements that can be made to the NSE directly is to relinquish the issues described in the first part of section 5.2, namely including IPv6 support and peer specific random delay into the implementation, and fixing the issue regarding sending validations and announcements from the same writer. The peer specific random delay however requires a specifically tailored implementation of gossip, since relaying estimate is done via an announcement. Gossip would then have add this delay to its sending of notifications whenever an estimate is being relayed.
More future work that would enhance the accuracy of estimates delivered by the NSE is include a mode for structured networks, where the amount of peers is directly calculated from the overlying structure.

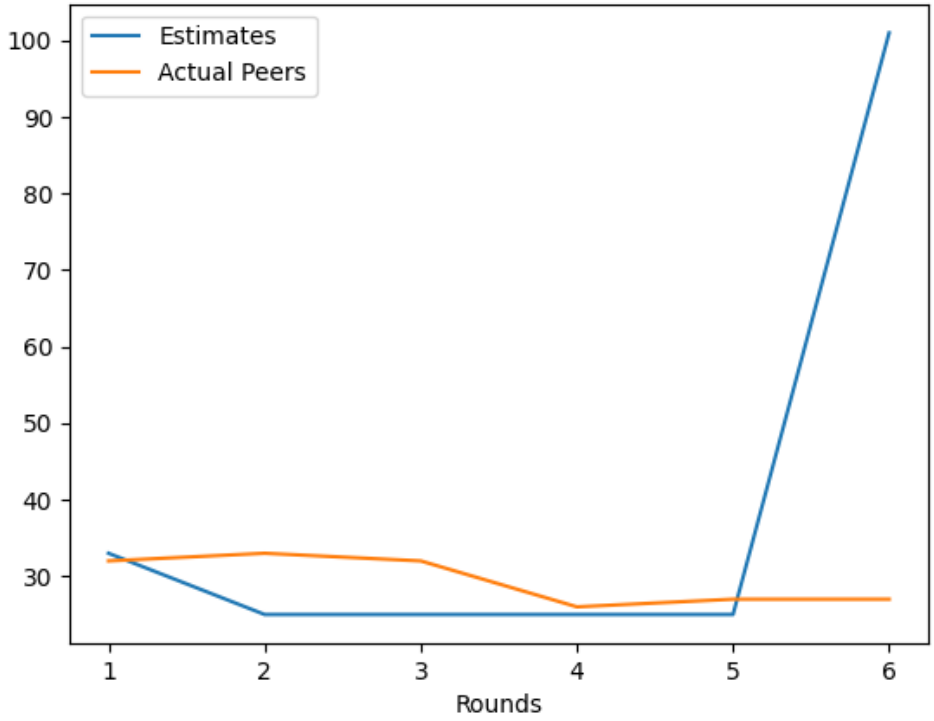


Figure 1: Sample chart of the testing scenario

Alongside improvements to the NSE, the testing scenario could also be updated: first of all the inclusion of a real gossip instead of a dummy would improve the validity of the test in simulating a network. In addition to that a complete revision of the testing scenario is also conceivable: instead of having a module that simulates a set amount of peers, a real local network with proper NSE instances would make the test even more authentic.

7 Workload

All of the work was done by myself. Additionally to the workload described in the midterm report around 14 full work days were put into the development of the NSE.

References

Nathan Evans Bartlomiej Polot, Christian Grothoff (2012). *Efficient and Secure Decentralized Network Size Estimation*. DOI: 10.1007/978-3-642-30045-5_23.