Peer-to-Peer Systems and Security - Initial Report Team 4

Kevin Burton

For the project making up the main part of the Peer to Peer Security lecture, I have chosen to implement the network size estimation module alone. The assigned team number after having done the registration is 4.
I plan to implement the module using Python on my GNU/Linux Ubuntu 22.04 LTS distribution. The reason for choosing Python is mainly that I already have some experience in using it and that I feel like there won't be any scenario down the road where I will be limited by the possibilities the language provides. Furthermore, interfacing with the testing module should be no issue, as this is developed in Python, too. Ubuntu is chosen as the OS for similar reasons: I have the most experience with distributions based on Debian, and it is usually the easiest platform to get an application or library to work properly.
The build system to be used is Setuptools[1], which enhances the basic Python `distutils` build system.
The implementation itself is planned to heavily rely on GNUnet's NSE described in "Efficient and Secure Decentralized Network Size Estimation" by Evans, Polot and Grothoff[2], as it both provides a solution for an unstructured overlay, such as the one we are dealing with, and prevents an attacker from skewing the estimate.
The quality of the code itself will be scrutinised using the Flake8[3] combined tool, consisting of the logical quality control PyFlakes, the stylistic control pycodestyle, and the analytical tool Mccabe.
API conformity will be tested using the NSE client that is part of the `voidphone_pytest` module.
The functionality of the NSE itself will be evaluated by simulating a peer-to-peer network with churn locally, with the nodes partaking in the network each providing a NSE service. Communication between these nodes is handled by the `gossip_mockup` module, which is also part of the `voidphone_pytest` repository. The result of the NSE will then be compared to the actual number of peers in the network over multiple rounds.
Important libraries are going to be `socket` and `asyncio` for inter-module communication, `hashlib` for proof of work and `pycryptodome` for cryptography. The latter is chosen as my main cryptographic library because of the many enhancements it brings with respect to the more standard `pycrypto`.
The module will most likely be licensed under CC0 and therefore be part of the public domain. Reasoning for this is that I have no financial interest in the software and in the unlikely event that someone desires to use the code for one of their own projects, I have no reason to withhold the free usage of it.
Previous experience relevant to this project include small Python homework challenges as part of the Network Security lecture and the implementation of a centralised checkpoint management system. The latter receives and remotely stores application checkpoints as part of a network of electronic control units, to later in the case of failure resume the application on a different node in the network. This was implemented in C++ as an application running on the Genode Operating System Framework as the main part of my Bachelor's Thesis.

[1] `https://setuptools.pypa.io/en/latest/index.html`
[2] `https://dx.doi.org/10.1007/978-3-642-30045-5_23`
[3] `https://flake8.pycqa.org/en/latest/`