1 Motivation

QNetwork aims to provide a simple mechanism that allows for secure communication using quantum key distribution. The system can be configured to use different QKD protocols. The protocols currently implemented are BB84 and DIQKD. Users of the framework are provided with a function to realize secure communication based on their configured protocol.

Listing 1: Sending message

```
from QNetwork.q_network import open_channel
with open_channel('Alice', 'Bob') as channel:
    channel.write("Hello, Bob!")
```

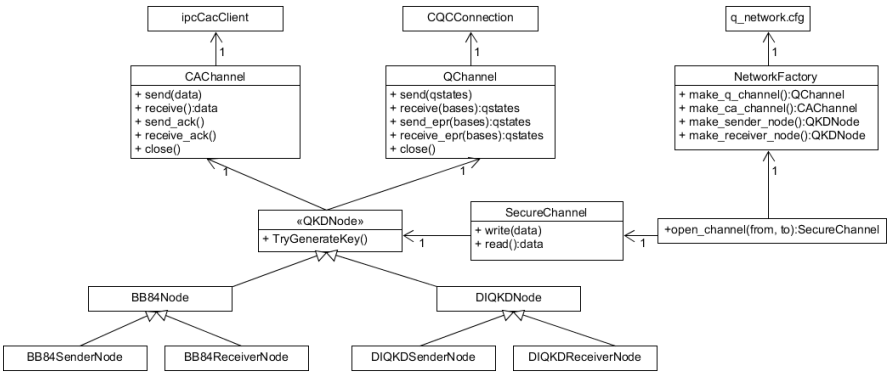
Listing 2: Receiving message

```
from QNetwork.q_network import open_channel
with open_channel('Bob', 'Alice') as channel:
    msg = channel.read()
    print("Bob received message:", msg)
```

This implementation can also be found in the example folder of the project. The function open_channel returns a SecureChannel context manager which handles the quantum key distribution and encryption of the message sent via the channel. The protocol used for QKD can be configured in q_network.cfg.

2 Code Architecture

The code has been designed to abstract away implementation details of quantum key distribution to facilitate easy extension of the system. The user should just see a very simple interface for communication.



One of the major refactoring opportunities that have been noticed during development is that a procedural approach for some of the basic QKD functions

would result in a better and more testable design. Additionally, the data flow of the QKD would be more clear.

2.1 QKDNode

This is the base of all the nodes implementing quantum key distribution and contains common functions and interfaces that are used by all protocols.

2.2 BB84Node

Implements the BB84 protocol for quantum key distribution. It is the base for the BB84SenderNode and BB84ReceiverNode with each implementing the sender and receiver version of the protocol.

2.3 DIQKDNode

Implements the DIQKD protocol for quantum key distribution. It is the base for the DIQKDSenderNode and DIQKDReceiverNode with each implementing the sender and receiver version of the protocol.

2.4 QChannel

Provides an interface to send and receive qubits or shared EPR pairs. It abstracts away the SimulaQron interface, hence it would be easy to use the framework with a different quantum communication device or a different version of SimulaQron.

2.5 CAChannel

Provides an interface to send and receive classical communication. It abstracts away the tinyIpcLib interface so it would be easy to use the framework with a different classical communication implementation.

3 Areas of Improvement

The implementation is currently lacking an error correction step. However, this could be easily added by extending the QKDNode with the appropriate functions.

The key extractor could be improved. The only possible choices currently are a single bit extractor algorithm or an algorithm that simply splits the raw key in $n/(n-k)$ substrings, before performing single bit extraction on each substring (where n is the length of the raw key and k the number of mismatching bits).

The DIQKD protocol is still in an early stage and couldn't be tested thoroughly. It has been implemented mostly to showcase the extensibility of the framework.