SimulaQron Ripe Hackathon

Welcome to the SimulaQron Ripe Hackathon! The coming weekend you will be discussing, programming, debugging (probably) and enjoying (hopefully) whatever creative and innovative use, extension or application you have in mind for SimulaQron, a simulation tool for software development for the quantum internet. We are really looking forward to the next two days, and we hope that so are you.

This document provides a very short introduction to what SimulaQron is and what it does, and where to find more information about its inner workings. You will also find *challenges*, proposed projects on which you may work during the weekend (but if you have something different in mind, that is also completely fine!). The weekend will focus on collaboration rather than on competition; we thus hope to create an atmosphere in which learning from and contributing to each other's ideas are as important as realizing your most creative an innovative ideas that take SimulaQron to new heights.

Good luck & enjoy!

Useful links

- `https://github.com/SoftwareQuTech/SimulaQron`. The SimulaQron code.

- `www.simulaqron.org`. The SimulaQron website.

- `https://arxiv.org/abs/1712.08032`. The publication on SimulaQron.

- `https://softwarequtech.github.io/SimulaQron/html/GettingStarted.html`. Documentation.

- `https://github.com/mdskrzypczyk/QChat`. The code for *QChat*, an application running on SimulaQron.

1 In a nutshell: quantum bits and why they are useful for communication

SimulaQron is a simulator for a quantum internet. Such an internet consists of nodes, which, on top of sending around bits (classical information), also send *quantum bits*, or *qubits*, which are the basic units of quantum information. Qubits have properties that are highly different from classical bits; for example, depending on the way we choose to read-out a quantum bit (referred to as a *measurement basis*), we may read a completely random bit and also change the qubit's state. We will be sure, however, which value we will read if the measurement basis is 'compatible' with the qubit's state. The latter fact is useful for cryptographic applications, since a sender and receiver can encrypt information and read out using a secret measurement basis, only known to the two parties involved, whereas an eavesdropper would read out in a basis which returns random information (see also *Quantum Key Distribution* in section 6).

Two or more qubits can be *entangled*, which means that their physical properties cannot be described separately. A "strongest" form of entanglement between two qubits (called *maximal entanglement*) has two favorable properties:

- maximal correlation: if the two qubits are read out in the same basis, they are bound to yield two maximally correlated[1] read out bits, one for each qubit;

- maximal privacy: no-one else than the parties with access to the two qubits at the time of measurement can even have the slightest knowledge of these two read-out bits.

---

[1] In case you want to be really precise: the two bits are either maximally correlated or maximally *anti*correlated.

In the context of a quantum internet, entanglement can be used for many things: for encrypting data, for transferring qubits by a process called *quantum teleportation*[2] and for protecting quantum data against decoherence of the state of the qubits (a form of quantum error correction), among many other things.

Finally, it is useful to know that manipulating a quantum bit can be done through a *quantum gate*, whose use is similar to the use of a logic gate in a classical circuit (although the underlying math is highly different).

Dealing with quantum data is highly different from dealing with classical information, not in the least because of the fact that qubits can be entangled. Although researchers all over the world are developing increasingly more accurate quantum processers, these devices are not yet sufficiently reliable for deployment in a real network. But why wait with developing software for the quantum internet in the meantime? SimulaQron has been developed for precisely this goal: it is a simulator with software development as main purpose. What follows in the next section is an overview of the context in which SimulaQron was developed: as a stand-in for part of the quantum network stack.

2   The bigger picture

2.1   Software can be independent from low-level hardware

To understand where in the overall system your programs will be running, it is useful to have a brief look at the bigger picture sketched in Figure 1. At the very bottom of this lies the actual quantum hardware. In Delft, we have two networked quantum hardware systems available: One very sophisticated system based on so-called NV centers in diamond. This is a small quantum computer, capable of executing quantum gates and measurements. Depending on the exact chip, these currently have between 1 and 10 qubits at QuTech. These have presently been entangled over a distance of 1.3kms at QuTech, which currently remains the world record in producing long lived entanglement. Such processors with a smaller number of qubits (2-5) are also planned to be deployed in the Dutch demonstration network at the end of 2020, with one such quantum computer in different cities. In addition, we also have a much simpler - but faster - system capable only of preparing and measuring one qubit at a time, for example for running quantum key distribution.

Within the European Quantum Internet Alliance, we also work with another quantum computing system, namely Ion Traps. For the purpose of this Hackathon, however, this system can be considered equivalent as a small quantum computer on which you can execute gates and make measurements! For networking, you want quantum computers that can (1) have an optical interface to talk to remote quantum computers (2) have good quantum memories, that is, the qubits live for a long amount of time allowing you to wait for communication (less than a second at present, but this is very long in the quantum world). Both NV in diamond as well as Ion Traps satisfy both of these demands. Other platforms that you may have read about online (superconducting qubits) presently do not.

At the quantum physical layer, such quantum hardware is connected via fiber. Using frequency conversion to the telecom band, standard telecom fibers can be used. In the demonstration network, we will for simplicity use a dedicated fiber for the actual quantum data, and send classical (control) information using a separate fiber. It is possible however in principle to transmit quantum and classical data over the same fiber, using technology at QuTech.

Above the quantum hardware sits a low level classical control system. This low level classical control is platform dependent, that is, it depends on what kind of quantum hardware (NV in diamond, Ion Traps, Photonic,...) sits underneath. It is responsible for controlling this exact physical system. Entanglement generating between neighbouring nodes is also dealt with on the low level controller, implementing our link and physical layer protocols.

Above, sits a high level control system (dubbed QNodeOS within the Quantum Internet Alliance). This is where applications are run, as well as higher level functionality such as network layer protocols sit. This higher level control system is platform independent, meaning that it can be programmed independently of what the underlying platform actually is. That is, for example applications you write there should in principle run on all platforms, and only be limited by the set of functionality they offer (such as the number of qubits, supported quantum gates, etc). The platform independent system can talk to the platform dependent one over a common interface called CQC (classical-quantum-combiner) which we are presently working on to implement as an interface to our hardware platforms. This is an extended API/instruction set that supports the executing of quantum gates and measuements on quantum processors, but also the creation of entanglement.
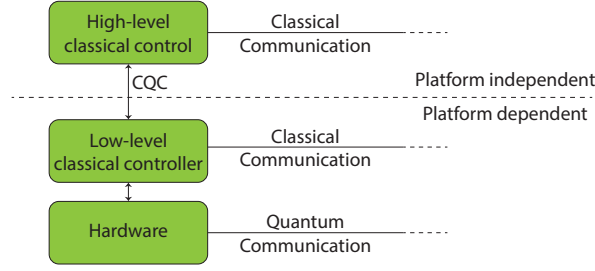
---

[2]See also sec. 6.1.

Figure 1: Very high level overview of the relation between the quantum hardware system, the low level control, and the high level classical control system on which applications are run. The clasical communication lines drawn here are logical ones and may be done over the same medium.

2.2 Relation to SimulaQron

SimulaQron provides a stand-in for the platform dependent part in a distributed simulation. You can talk to it over the same interface, namely CQC. Since this is quite cumbersome, a higher level python library allows you to program quite abstractly, and will then talk to the platform dependent system over CQC in order to execute quantum commands. Evidently, the simulation works differently than the real system in order to realize such behaviour: it is a distributed simulation giving you the illusion of talking to real quantum processors and being able to entangle them at a distance. This sounds very complicated, but the real system is also extremely complex, requiring a lot of nitty gritty which we will not need to consider in this hackathon!

2.3 Relation to our quantum network stack

While this is not really relevant for this hackathon, as we will focus mostly on application programming, and testing, also quantum networks require a network stack to function. Importantly, to scale quantum networks one wants to have a stack that enables fast and reactive control, since qubit lifetimes are still very short.

For completeness we briefly sketch a short overview in Figure 2. The physical layer is where the real quantum happens. In our system, it is resonsible for the physical production of entanglement requiring timing synchronization between neighbouring nodes. We call our physical layer protocol a Midpoint Heralding Protocol (MHP), which fits many different hardware systems. All low level control (such as timing synchronization) happens here, but no higher level logic is available. On top, we have a link layer protocol which we call Entanglement Generation Protocol (EGP). The EGP can be smart and allow higher level logic, for example to decide when to produce entanglement or which entanglement is produced first, or allocate quantum memory for entanglement production. Both the MHP and the EGP reside on the low level controller.

To the higher layers the EGP provides an abstract interface in which one can request to create and keep entanglement (CREATE+KEEP), or to create and immediately measure entanglement (CREATE+MEASURE) with directly connection nodes on the same link. Evidently, CREATE+KEEP also allows you to measure later, but goes back to the higher layer to wait for such instructions. There are very good reasons for supporting both options, and we are happy to explain this to you in person. Along with the CREATE the higher layer may supply a minimum desired fidelity, a maximum waiting time, and the number of entangled pairs to produce. If successful, EGP will return to the higher layer an OK message, that - amongst other things - includes an entanglement identifier that is unique in the network. In the simulation, SimulaQron provides an interface for you to the EGP via CQC, and returns such an identifier. Since SimulaQron is a simulation, it achieves the same functionality in a different way than our actual link layer protocol EGP does.

Higher layers of the network stack are responsible for producing entanglement between nodes not on the same network reside on the platform independent high level control. Here, we are presently exploring good options of what this stack might look like exactly for us to allow scalability.

3 More details on SimulaQron

More information about the inner workings of SimulaQron can be found in the paper `https://arxiv.org/abs/1712.08032` and the website `www.simulaqron.org`.

A few things have changed since the paper was published. One of the main differences in the current version of SimulaQron with the version described in the article (`https://arxiv.org/abs/1712.08032`) has to do with the fact that quantum information that is sent through the network is *noisy*. Just like classical
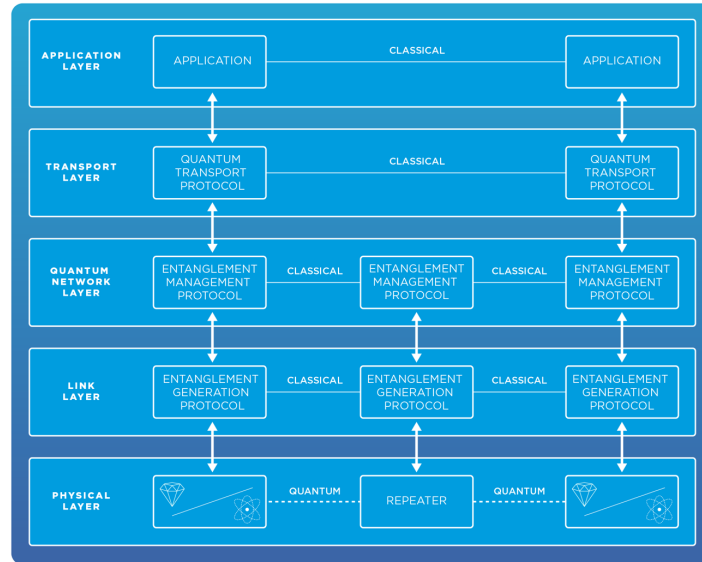
Figure 2: Present version of the network stack of the Quantum Internet Alliance.

bits, sent as light pulses through a glass fiber, might flip from 0 to 1 or vice versa, quantum bits also suffer from noise when they are sent. For quantum bits, life is even worse: the information a quantum state holds goes lost over time, a process known as 'decoherence'.

In the version of SimulaQron you will be working with during this weekend, it is possible to turn on decoherence over time (see `SimulaQron/config/settings.ini`). We challenge you to use your creativity to use and improve SimulaQron to build applications or other uses that deal with such noise. It might sound that in order to tackle problems related to noise, you should have studied quantum information for a few years, but that is definitely not true: using a bit of knowledge (we are around to explain) and most of your own expertise and creativity, only your imagination (and the amount of hours in a single weekend) is the limit!

It is now also possible to setup a network with a restricted topology, i.e. a network where not every node is directly connected to all other nodes. For more details on how to do this, see `https://softwarequtech.github.io/SimulaQron/html/ConfNodes.html#configuring-the-network-topology`.

4   Installation & getting started

For installing SimulaQron and how to set up the virtual servers, we refer to the documentation: `https://softwarequtech.github.io/SimulaQron/html/GettingStarted.html`.

During the hackathon we will setup a few computers running multiple simulated networks which you can connect to by sending CQC messages to the nodes, see section 2.1. For more information on how to connect to a remote simulated network using the Python library, see the docs of SimulaQron and in particular: `https://softwarequtech.github.io/SimulaQron/html/ConfNodes.html#connecting-to-a-remote-simulated-network`. In a git branch called "Hackathon2018" you will be able to find the CQC config files you need to access the remote simulated networks in the folder `network_configs`. There will be simulated networks that can be used by the different teams and also two common networks that can simultaneously be used by everyone (like a real network :)). When using the common networks we ask you to set the application IDs (appID) of the CQCConnection (or in the CQC message if you're sending this manually) to your team number.

For example, if you are in team 1 and wish to create a qubit at node *Node0* in the network *network_complete_team01* you can simply execute the following code:

```
cqc_file_path = "/path/to/SimulaQron/network_configs/network_complete_team01"
with CQCConnection("Node0", cqcFile=cqc_file_path) as node0:
    q = qubit(node0)
    m = q.measure()
    print(m)
```

or let's say that you are in team 3 and wish to do the same but in the common network with restricted topology:

```
cqc_file_path = "/path/to/SimulaQron/network_configs/network_topology_common"
with CQCConnection("Node0", cqcFile=cqc_file_path, appID=3) as node0:
    q = qubit(node0)
    m = q.measure()
    print(m)
```

5   Challenges

Below, we describe a few possible challenges that you could work on during the weekend. For you to get an idea where you could best use your expertise, we added an indicator of how much background knowledge could help you. The level is indicated by a pictogram of a diamond with an electron (⬦): these refer to nitrogen-vacancy centers in diamond, one of the promising physical systems for realising quantum networks.

For more examples on how some of the quantum information concepts are implemented in code, please see the documentation on https://softwarequtech.github.io/SimulaQron/html/index.html.

Challenge A: create network exploration tools (a test suite)

Difficulty

Networking:	⬦⬦◇◇◇
Quantum:	⬦⬦⬦⬦◇

This challenge focuses on adding tools to SimulaQron that enables a node to "explore" the network it is part of, in terms of its physical properties (how noisy and long are the links?) and its topology (to which other nodes is a node connected?). In particular, you could think of

- adding a network ping: a classical message that a node sends to the address of another node to find out if it's part of the network and still alive;

- adding a "quantum ping". This could, for example, be *teleportation*[3] of a qubit to a remote node and simultaneously send additional classical instructions stating that the qubit should be teleported back. Subsequently find out how much noise crept in in the meantime by comparing the original state with the received state by computing their *fidelity*.

- measure the *qubit error rate* (QBER, see also sec. 6)

- find out the state of a quantum bit that was sent to you by using *tomography* (see also sec. 6)

- find and implement a neat way to visualize the network properties you measured

Let us explain why understanding the properties of a quantum network as the ones described above can be quite crucial. As an example, we argue that knowing the noisiness of the network a node is part of can have a vast impact on the scheduling choices a node makes. To be precise: if the quality of entanglement is low, then consuming additional entangled pairs of qubits in order to increase the quality of the first link is favorable (a process known as *purification*). Whereas high-quality entangled links can straightforwardly be used for applications. Note that such scheduling is not something to be though of lightly: since quantum bit information decoheres over time, processing of quantum information is always a race against the clock!

Challenge B: take SimulaQron's network tools to a next level

Difficulty

Networking:	⬦⬦⬦⬦◇
Quantum:	⬦◇◇◇◇

Currently, networks in SimulaQron are static: after having set up the network, it is not possible to add or remove nodes to network. To real-life networks, however, nodes are added or removed all the time: when someone connects their laptop to a WiFi-network, when a server breaks down, and so on. In order to make SimulaQron's network resemble real-life (dynamic) networks more, you could think of:

---

[3]See also sec. 6.1.

- letting other nodes hook on to an existing network

- having a network deal with nodes that suddenly break down or disconnect from the network

Note that, since multiple computers can take part in simulating the network, one important question in this challenge is how a computer should leave the simulation without breaking it. For this challenge it is therefore helpful to understand the inner workings of SimulaQron and in particular the notion of virtual and simulated qubits, see `https://softwarequtech.github.io/SimulaQron/html/Overview.html#how-simulaqron-works-internally`.

Challenge C: take *QChat* to new heights

Difficulty	
Networking:	🔹⬡⬡⬡⬡
Quantum:	🔹🔹⬡⬡⬡
Classical error correction:	🔹🔹🔹🔹🔹

The application *QChat* was written by Matthew Skrzypczyk[4] and can be found on `https://github.com/mdskrzypczyk/QChat`. It is a chat application that uses *Quantum Key Distribution* to securely send messages to remote nodes. In this challenge, you could extend and improve it, for example in the following directions:

- make the application more optimal in the noisy version of SimulaQron, where quantum information decoheres over time

- add improvements using (classical) information reconciliation and privacy amplification techniques to extract as much key as possible (see below for references).

- add a graphical user interface

- add lightweight public cryptographic tools for verifying dignital signatures

A few references for post-processing of the generated secret key:

- By *information reconciliation*, we mean a process that ensures that the two parties involved end up with the same secret key. For implementation, you could use any classical scheme, such as Low-density parity-check codes, Turbo codes or Polar codes. In case you want to implement a protocol specifically tailored to Quantum Key Distribution, then you could have a look at *Cascade* by Brassard and Savail: `https://link.springer.com/content/pdf/10.1007%2F3-540-48285-7_35.pdf`. You could also go a bit further and implement an improved version: `https://arxiv.org/pdf/1007.1616`.

- *Privacy amplification* is about manipulations of the shared key such that an eavesdropper has minimal information about the final (shorter) key. The standard approach is to have to the two parties publicly decide on a universal hash function, which takes bitstrings of the size as the initial key as input, and outputs a bitstring of t fewer bits, where t is the number of bits that the eavesdropper is suspected to have learned.

Challenge D: whatever you come up with yourself

In case you have ideas of your own on how to use or extend SimulaQron for new applications, you are free to work on this, of course!

---

[4]Matt will also be around during the weekend!

Below, we add highly brief descriptions of some concepts that might help you in working on your challenge. If you require more information: just ask us!

6 Dictionary of some quantum concepts

Concept	Meaning
Bell pair	Pair of entangled qubits.
Entanglement between two qubits	Joint property of two qubits: the quantum state of these qubits cannot be described separatedly. It thus only makes sense to talk about the "joint state" of the two qubits.
EPR-pair	Pair of entangled qubits. Name comes from one of the initial papers on the topic, by Einstein, Podolsky and Rosen. Ironically, this paper argued against the existence of such entangled pairs.
Measurement/read-out of a qubit	Yields a single-bit. In general, this single bit is the outcome of a random biased coin flip, where the bias depends on the degree to which the chosen measurement basis and the quantum state of the qubit 'align'.
Measurement basis (Read-out basis)	"Measurement settings" using which one decides to measure/read-out the information that a quantum bit holds. Choice of measurement basis might strongly influence the outcome.
Qubit (quantum bit)	Equivalent of a classical bit. Can be in the 0-state or 1-state, or in a superposition of these.
Quantum bit error rate (QBER) in a given measurement basis	Probability that measuring two (entangled) qubits in the given measurement basis gives fully (anti)correlated outcomes. See also 'QBER' in the documentation `https://softwarequtech.github.io/SimulaQron/html/ExamplespythonLibQBER.html?highlight=qber`
Quantum gate	Operation that manipulates a quantum state. Equivalent of a logical gate in classical circuits.
Quantum Key Distribution (QKD)	Class of protocols for generating secure shared classical key using quantum bits. Famous QKD protocols include BB84 (Bennett-Brassard-1984) and Ekert91 (Ekert, 1991).
Quantum teleportation from node A to node B	Transferring a single qubit from node A to node B by consuming a shared entangled pair of qubits between them. See also sec. 6.1
Tomography	Process of measuring many copies of the same qubit or set of qubits to find out its quantum state. See also 'tomography' in the documentation `https://softwarequtech.github.io/SimulaQron/html/SimulaQron.cqc.pythonLib.html?highlight=tomography#SimulaQron.cqc.pythonLib.cqc.CQCConnection.tomography`

6.1 Quantum teleportation

Quantum teleportation is the process in which we transfer a quantum bit by consuming a single entangled pair. In the following, we show the code of two nodes Alice and Bob, where Alice wishes to teleport a qubit to Bob.

Alice's code is:

```python
# Initialize the connection
with CQCConnection("Alice") as Alice:

    # Make an pair of entangled qubits, one of which
    # one Alice's side and one with Bob
    # (such a pair is also called an EPR-pair)
    qA=Alice.createEPR("Bob")

    # Create a qubit to teleport
    q=qubit(Alice)

    # Prepare the qubit to teleport in |+>
```

```
q.H()

# Apply the local teleportation operations
q.cnot(qA)
q.H()

# Measure the qubits
a=q.measure()
b=qA.measure()

# Send measurement outcomes to Bob.
# Bob needs these outcomes to correct
# the state of the quantum bit
Alice.sendClassical("Bob",[a,b])
```

Bob's code:

```
# Initialize the connection
with CQCConnection("Bob") as Bob:

    # Make an entangled pair with Alice
    qB=Bob.recvEPR()

    # Receive Alice's measurement outcomes
    data=Bob.recvClassical()
    message=list(data)
    a=message[0]
    b=message[1]

    # Apply a gate that corrects the
    # state of the quantum bit, depending
    # on Alice's measurement outcomes
    if b==1:
            qB.X()
    if a==1:
            qB.Z()

    # From this point onward, 'qB' holds
    # the quantum state that was called 'q' in the code
    # for Alice. Now you could start to use it for your
    # favorite application.
```

(Code can be found in SimulaQron/examples/cqc/pythonLib/teleport.)