Delft University of Technology
Faculty of Applied Sciences

Entanglement tracking and EPL entanglement generation in quantum networks using the discrete event simulator QNetSquid

Thesis for the degree of

MASTER OF SCIENCE
in
Applied Physics

by
Roeland ter Hoeven

Delft, The Netherlands
August 2018

MSc Thesis Applied Physics

"Entanglement tracking and EPL entanglement generation in quantum networks using the discrete event simulator QNetSquid"

Roeland ter Hoeven
4312082

Delft University of Technology

Supervisors

Prof. dr. S.D.C. Wehner
MPhys. F.D. Rozpędek

Other committee members

Dr. D. Elkouss
Dr. M.T. Wimmer

August, 2018 Delft

Abstract

The realisation of a quantum network in the near future has been made possible by the developments in qubit systems. Links between quantum nodes in major cities in The Netherlands are scheduled to be demonstrated by 2020. There are many things that need to be considered in the development of a quantum network, in this thesis a few of these will be examined. The first topic is entanglement generation using the EPL (Extreme Photon Loss) protocol in the presence of dephasing noise, which is an existing protocol. Novel contributions include the maximum achievable fidelities and entanglement generation rates, which are obtained using theoretical analysis and numerical simulations. Also for elementary states generated by the EPL protocol it is shown that doing distillation first and then an entanglement swap results in a roughly two times higher rate than the other way around.

A new protocol called entanglement tracking is proposed to keep track of the classical information about entanglement in a quantum network. The entanglement tracker is a protocol that runs locally on every node in a quantum network and can communicate with the entanglement trackers of other nodes. The goal of the entanglement tracker is to keep a database with entanglement identifiers that is continuously updated after entanglement is modified. Performance metrics of an entanglement tracking protocol and a command interface with higher layer protocols are defined. We propose a format for entanglement identifiers and show how to update entanglement identifiers after an arbitrary number of entanglement swaps. The entanglement tracker is implemented in the discrete event simulator QNetSquid, which is under development by QuTech. It is applied by simulating a repeater chain using the entanglement tracking protocol, resulting in a completion time as a function of the number of repeaters. Future applications of the entanglement tracker are to assist routing of entanglement in a (large) quantum network, in which the tracker can take care of the classical communication.

Contents

0.1 Acknowledgements

I would like to thank everyone who helped me complete this thesis, both privately and professionally. It has been an interesting journey leading to many new challenges and struggles, which pushed me beyond what I had experienced before.

I want to especially thank my supervisors, Filip Rozpędek for his continuous support and cooperation throughout my project and Stephanie Wehner for keeping me on track and always giving me new insights. I really appreciate the discussions and cooperation with Tim Coopmans on various QNetSquid-related topics. The brainstorming sessions with Kenneth Goodenough and Axel Dahlberg about entanglement identifiers and the entanglement tracker were very useful. Rob Knegjens and Julio de Oliveira Filho were always willing to teach me about QNetSquid, support me with related issues and discuss applications. Finally, I would like to thank David Elkouss for helping me at different stages of my project and joining the thesis committee together with Michael Wimmer.

Chapter 1

Introduction to quantum networks

In this chapter some of the main tools that can be used to realise a quantum network will be discussed. A quantum network has not been realised yet, but there are many proposals and ideas. The main challenges are to construct quantum nodes and be able to run quantum protocols between the nodes in the network. In section 1.2 we will look at possible realisations of a quantum node. The quantum nodes should have access to at least a few quantum bits (qubits) that they have control over. Operations on these qubits and storage of the qubits should be good enough, which means that the quantum properties are preserved.

For a quantum computer to outperform a classical computer it needs to have a lot of qubits. The time it takes to do certain computations may scale better with the number of qubits for a quantum computer, but if the classical computer has millions of bits, we still need a lot of qubits to overcome that. The advantage of a quantum network is that there are protocols that can outperform any classical protocol, even when using just one or two qubits. This is due to the inherent quantum properties, which allows us to make cryptographic guarantees that would not be possible classically. Examples of these protocols are quantum key distribution [1], anonymous transfer [2] and position verification [3].

There are different ways the quantum nodes can communicate with each other. The easiest way is to simply have a cable running from one quantum node to the other and send photons carrying the quantum information. There are also proposals to send photons to satellites, which can then orbit the earth and emit photons again to establish a quantum connection. Here the focus will be on a network of quantum nodes connected by fibres.

One of the most important tasks of a quantum network is to establish quantum entanglement between two nodes in the network. This can be seen as a

goal in itself, as there are many quantum protocols already designed to run between two quantum nodes that need entanglement. Establishing entanglement between two quantum nodes connected with fibres has already been done over more than a kilometer [4] and there are plans to extend this to longer distances.

The generation of elementary entanglement can only be done between two nodes that are directly connected. Sometimes nodes do not have a direct connection, but instead are connected only via a middle node. It is then possible to make entangled links from both end nodes to the middle node and actually make entanglement using an entanglement swap [11]. The entanglement swap protocol, which is directly linked to quantum teleportation will be covered in section 1.5. In general this can be used to establish entanglement between any two nodes in a network, using a number of entanglement swaps.

Entanglement distillation consumes multiple weakly entangled pairs between two nodes to generate a better entangled pair. Weakly entangled means that there is noise in the system that destroys some of the entanglement. To look at a realistic model of a quantum network, basic noise models can be used to model the noise on the system. One of the main challenges of designing a quantum network is to limit this noise and enable multiple parties to run quantum protocols. More detail will be given in section 1.6 about entanglement distillation and section 1.4 about noise models and decoherence.

Ultimately the goal of designing a quantum internet can be compared to the internet as it is now. Any computer on earth that is connected to the internet is be able to communicate with any other computer that is connected to the internet. For a quantum network any quantum computer could then establish entanglement with any other quantum computer. There may of course be more requirements than just to establish entanglement, but this would be an important underlying protocol that will enable many applications.

1.1 Thesis outline

The thesis is divided into four chapters, of which this one is the first and serves the purpose of introducing the necessary background. No new results are introduced, but the material will be used throughout the rest of the thesis. The remaining three chapters are mostly separately readable, with some references between them. Each chapter has a separate conclusion and recommendations for future research.

The second chapter is about EPL [24] (Extreme Photon Loss) entanglement generation, which is an existing protocol to establish entanglement on an elementary link in the network. This protocol generates two noisy states that do not contain any useful entanglement and performs entanglement distillation to

obtain useful entanglement. The EPL protocol has also been realised experimentally [20]. The following contributions are made within this thesis: The EPL protocol is first analysed in a noisy scenario in section 2.1. After that our goal is to find the maximum rate at which entangled states can be generated given a specific number of available quantum memories in section 2.2. Then we analyse the difference between first doing entanglement swaps and then entanglement distillation, or the other way around, using the noisy states in section 2.3. The chapter concludes with numerical experiments in MATLAB in section 2.4, illustrating the results obtained in the previous sections.

The third chapter starts by introducing the discrete-event simulator QNetSquid, which is a Python package for simulating quantum networks, currently under development by QuTech. The rest of the chapter is about writing snippets, that each have a specific purpose of simulating a (small) part of a quantum network and doing simulations using these snippets. First an entanglement generation protocol in QNetSquid is obtained in section 3.2 by sampling from distributions from a previous result [17]. Using this the EPL entanglement generation can be simulated in a more realistic manner and the relation between swaps and distillation is again analysed in section 3.3. Then we develop an entanglement generation protocol that models every physical component in QNetSquid instead of sampling from a distribution in section 3.4. For this we look at photon detectors and how to model these in QNetSquid. The total model then consists of quantum memories, fibres and photon detectors that are combined into an entanglement generation protocol. Finally section 3.6 is about comparing the entanglement generation protocols and running simulations to obtain the average rate and fidelity of these protocols.

The final chapter is about the entanglement tracker, which is a new protocol that is in charge of handling the classical information about entanglement in a quantum network. The development of the entanglement tracker was inspired by the routing models discussed in [18] for a grid network and in [19] for a ring or sphere network. First we discuss the concept of entanglement identifiers that are given to every entangled link in the network in section 4.1. These identifiers can be used by nodes to run protocols and should be consistent such that both parties involved in the entanglement hold the same identifier. Then the entanglement tracker itself is discussed, which is a local protocol that runs on every node in the network in section 4.2. The entanglement tracker is in charge of updating the local databases containing entanglement identifiers. A higher layer protocol should give commands to the tracker whenever entanglement is generated, consumed, swapped or distilled. The tracker can then update its database and send messages to other nodes to update their databases accordingly. We then define some performance metrics that can be used to investigate the performance of an entanglement tracking protocol. In section 4.3 and section 4.4 we look at how to update entanglement identifiers after entanglement swaps, including the goodness parameter which gives a heuristic guess of the fidelity of the entangled pair. Finally section 4.5 is about numerical testing of

the entanglement tracker using an implementation of the entanglement tracker in QNetSquid. The test case is a repeater chain, which is a series of nodes that can be used to generate long distance entanglement using many entanglement swaps. The goal of this simulation is to compare the time it takes to establish long distance entanglement for two different entanglement tracking protocols.

1.2 Quantum hardware and entanglement generation

There are many experimental groups working on building a quantum computer or quantum node. A lot of different implementations use solid state materials as the basis, for example qubits in a superconductor [5]. There are also groups that trap ions using electric fields [6], or use a superconductor coupled to a cavity filled with photons [7] to realise qubits. The most important property of all these realisations is that they can make an effective two level quantum system.

All these different realisations can be used to make a quantum node in a quantum network, as long as they can establish entanglement between so-called memory qubits and flying qubits. They have to be able to store the memory qubits for a reasonable amount of time and transfer the flying qubits, mostly photons, onto a fibre to reach other nodes. The coupling from the quantum system onto the fibre is an experimental challenge. If the qubit system is well-isolated from environmental noise, it is often also hard for photons to leave the system. In general, there is a trade-off between fast operations and coherence time of quantum memories. This can be understood by considering that operations on these qubits are nothing more than an external interaction. The same principles that allow this interaction to effect the qubits also makes it more exposed to decoherence. This can be prevented by making sure that only a very controlled external field is able to reach the qubit and it is shielded from other interactions, but experimentally this is one of the big challenges. In section 1.4 noise and decoherence will be examined in more detail.

In this thesis nitrogen-vacancy centers (NV-centres) [8] will mostly be discussed. These will be used to attempt to establish the first quantum network in the Netherlands between multiple cities. These NV-centres exist in a diamond lattice, as a single nitrogen atom and a free electron. The free electron and an electron from the nitrogen atom form a boson with three spin levels, of which two can be used as an effective qubit. This qubit can be used as an interface with the NV-centre, for example to emit photons. The surrounding carbon atoms and the nitrogen atom can be used for long-term storage of qubits. In these systems multiple promising results have been obtained in the last years, such as a loophole-free Bell-test [4] and entanglement distillation [9].

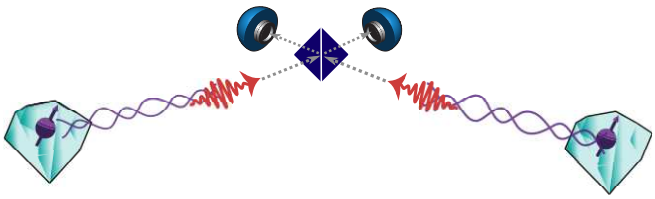


Figure 1.1: Physical set-up for entanglement generation. Both sides have access to a single NV-centre in diamond and can output photons towards the midpoint. By coordinating their timings, the photons from each side reach the detectors at the same time. The photons then interfere at the beamsplitter resulting in an entangled state between the NV-centres. Image courtesy: the authors of [10].

The ability to generate and share entanglement is one of the things that makes a quantum network special. We will briefly explain how entanglement can be generated between NV-centres.

The protocol requires two nodes to be connected with fibres, either directly or using a midpoint station. The most intuitive entanglement generation protocol is for one of the nodes to make an entangled state locally between a memory qubit and a photon. The photon then gets put on the fibre and gets picked up by the other side. The receiving node then transfers the state of the photon onto one of its memory qubits. There are a few problems with this scheme, for example catching a photon and transferring it to a memory state is very hard.

The entanglement generation protocol that will be looked at during this thesis requires both nodes to make local entanglement between a memory qubit and a photon. The photons then get put on a fibre towards the midpoint station where the photons are detected after a beamsplitter. Effectively this beamsplitter carries out an entanglement swap, resulting in an entangled state between the memory qubits of the nodes. In the chapter about the discrete event simulator QNetSquid, the whole entanglement generation protocol will be examined in more detail starting from section 3.2. This includes the source that can make locally entangled pairs and modelling of the beamsplitter detection, this is the QDetector as explained there.

1.3 Basic quantum mechanics

It is important to establish a mathematical framework for the rest of this thesis. For that we need density matrices, a general way to represent the state of a quantum system. In the following sections we will use these density matrices to examine quantum protocols.

Pure quantum states can be defined by a wave function, which is a 'ket' $|\phi\rangle$ in Dirac notation. There is also a 'bra' $\langle\phi|$, which is the complex conjugate of the ket. Every state should be normalized, which is similar to how a classical probability distribution should add up to one. Mathematically it can can be formulated using the criterion: $\langle\phi|\phi\rangle = 1$. Transformations from pure states to other pure states can then be done using a unitary U:

$$|\psi\rangle = U|\phi\rangle \tag{1.1}$$

Because U is unitary, applying it to the normalised state $|\phi\rangle$, results in a new normalised state $|\psi\rangle$. In section 1.5 specific unitary operations like the Hadamard and CNOT gate will be considered.

A density matrix ρ can in general be a classical mixture of pure quantum states. Mixed states occur due to a lack of knowledge about the state of the system. For example, the system could be in a few different pure states, but we do not know in which one it actually is. We can then assign classical probabilities to each pure state and write the total state as a mixture of these pure states. This can be made formal by defining ρ as:

$$\rho = \sum_i p_i |\phi_i\rangle\langle\phi_i|, \tag{1.2}$$

where $0 \leq p_i \leq 1$ and $\sum_i p_i = 1$. The way to operate on density matrices is similar to wave vectors. Here we restrict ourself to unitary operations on the density matrices. In section 1.4 a few specific non-unitary noise operations will be discussed, like depolarising and dephasing operations. To perform a unitary operation on a density matrix we have to apply the unitary on both sides of the density matrix:

$$\sigma = U\rho U^\dagger = \sum_i p_i U|\phi_i\rangle\langle\phi_i|U^\dagger = \sum_i p_i |\psi_i\rangle\langle\psi_i|, \tag{1.3}$$

where $U^\dagger$ is the hermitian conjugate of U and $|\psi_i\rangle = U|\phi_i\rangle$. This can be seen using the identity $\langle\phi_i|U^\dagger = (U|\phi_i\rangle)^\dagger$.

In general performing unitary operations on density matrices is simple, we can analyse the way the unitary acts on the separate parts of the mixture $U|\phi_i\rangle$ for all i. After that the contributions can be summed up to construct the resulting density matrix, σ in this case.

There are four maximally entangled states that are commonly used, called Bell-states. They form a basis for the two qubit space and are equivalent to each other up to local operations on either qubit. They are pure states and can be defined as follows:

$$|\phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$
$$|\phi^-\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$
$$|\psi^+\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}}$$
$$|\psi^-\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$$

An important property of entanglement is that it is non-local. This means that even when particles are moved very far apart, they can still be entangled and maintain those correlations.

Consider the fidelity of a density matrix ρ to a pure state $|\psi\rangle$. It gives a measure of how close the two states are and is defined as follows:

$$F = \langle\psi|\rho|\psi\rangle \tag{1.4}$$

In general this is a number between 0 and 1, where 1 means that the states are equivalent. In this thesis the fidelity to a maximally entangled state will often be used. We will sometimes talk about the fidelity of a state, meaning the fidelity to a maximally entangled state. When establishing entanglement the goal is to obtain a fidelity that is as high as possible.

The most simple measurements of a quantum state collapse the state to one of the eigenstates of a basis $\{|\phi_i\rangle\}$. The probability p_i of collapsing to the eigenstate $|\phi_i\rangle$ for a quantum state described by a density matrix ρ is given by $p_i = \langle\phi_i|\rho|\phi_i\rangle$. One of the most often used measurements is in the standard basis, which uses the two eigenstates $|0\rangle$ and $|1\rangle$. Another important measurement uses the four Bell-states as the eigenstates and is called a Bell-state measurement or measurement in the Bell-basis. Measuring in the standard basis is a single qubit measurement, resulting in two possible outcomes. Measuring in the Bell-basis is a two qubit measurement, resulting in four possible outcomes. Bell-state measurements are the basis for the quantum teleportation protocol and for the entanglement swap.

The most general quantum measurements can be defined using a set of Kraus operators $\{A_x\}$, resulting in a positive-operator valued measure (POVM). The POVM-elements M_x are then given by $M_x = A_x^\dagger A_x$ and are required to sum up to identity $\sum_x M_x = I$. The POVM-elements can be used to calculate the probability to obtain a certain measurement outcome when measuring a state ρ, using the trace:

$$p_x = Tr(M_x \rho). \tag{1.5}$$

The post measurement state can be found using the Kraus operators and is given by:

$$\rho_x = \frac{A_x \rho A_x^\dagger}{Tr(M_x \rho)}. \tag{1.6}$$

General quantum measurements will be used in the construction of the QDetector, as will be discussed in section 3.4.

1.4 Noise and decoherence

Understanding exactly which kind of noise has an influence on a quantum system can help to find a solution and get rid of the noise. In this section dephasing and depolarising noise will be discussed, as well as amplitude damping, which are general ways to represent noise in a quantum system. We will also look at the T_1 and T_2 times that are often mentioned in experiments to indicate the coherence time of a quantum system.

Depolarising noise is given by the following noise map on a density matrix ρ:

$$N_p(\rho) = p\rho + (1-p)\frac{I}{d}, \tag{1.7}$$

where d is the dimension of the system and $\frac{I}{d}$ is the maximally mixed state, given by the normalized identity matrix. The noise coefficient $0 \leq p \leq 1$ determines the amount of noise. In the extreme case of $p = 0$, all the information about the initial density matrix is lost.

If the original state ρ is a Bell-state, this means that the noisy version of the state loses entanglement. The parameter p can also be a function of time, for example when this noise channel is applied to a qubit in a quantum memory. In this case there would be an interaction with the environment that slowly degrades the state to a maximally mixed state.

Depolarising noise is often used as a worst case scenario in theory, but in experiments noise is often indicated by T_1 and T_2 times. These times indicate the coherence time of a quantum system. In general when doing experiments on quantum systems, the experimenters try to make these coherence times as long as possible. There are different mathematical definitions and interpretations of the T_1 and T_2 times. Here we will assume that T_1 is the time it takes to lose all information about the state of the system, whereas T_2 is about the time it takes to lose its quantum phase. To understand the idea of these T_1 and T_2 times, it is useful to look at a qubit.

A qubit is a quantum system with two eigenstates, which are often separated in energy. We can then talk about the ground state, which is the state with the lowest energy and the excited state, which has (slightly) more energy. The difference in energy between the excited state and the ground state, is often called the (energy) gap. The qubit can in general also be in a superposition of the ground state and the excited state, which means it can be represented as:

$$|\psi\rangle = \alpha|g\rangle + \beta|e\rangle, \tag{1.8}$$

where $|g\rangle$ is the ground state and $|e\rangle$ is the excited state, α and β are complex numbers such that $|\alpha|^2 + |\beta^2| = 1$.

We know that when left alone, any system decays back to the ground state. For example by releasing a photon with the energy of the energy gap between ground and excited state. The time constant with which the excited state naturally decays to the ground state is the T_1 time.

The T_1 decay process can also be seen as amplitude damping, which is defined using the Kraus operators $E_0 = |g\rangle\langle g| + \sqrt{1-\gamma}|e\rangle\langle e|$ and $E_1 = \sqrt{\gamma}|g\rangle\langle e|$. Here γ is the amplitude damping coefficient, indicating the strength of the amplitude damping channel acting on a density matrix ρ:

$$N_\gamma(\rho) = E_0\rho E_0^\dagger + E_1\rho E_1^\dagger. \tag{1.9}$$

This is also the process that happens in fibres when photons get lost, as will be discussed in section 3.4.

The decay process indicated by T_2 does not change the energy of the qubit; the amplitudes $|\alpha|^2$ and $|\beta|^2$ are constant over time. It is rather the phase between ground and excited state that changes. In general the T_1 time is longer than the T_2 time, which means that the phase of the qubit drifts faster than the decay process to the ground state. As a result, in some cases the T_1 process can be disregarded, because by the time it decays to the ground state it will already have lost all coherence due to dephasing. In that case we can model the noise on the qubit purely by dephasing:

$$N_p(\rho) = p\rho + (1-p)Z\rho Z, \tag{1.10}$$

where Z is the pauli matrix given by $Z = |g\rangle\langle g| - |e\rangle\langle e|$.

To see that this noise map implements dephasing we can apply it to the pure state $|\psi\rangle$ from 1.8, which means that $\rho = |\psi\rangle\langle\psi|$. Then the effect of applying Z to the state ψ is:

$$\begin{aligned} Z|\psi\rangle &= (|g\rangle\langle g| - |e\rangle\langle e|)(\alpha|g\rangle + \beta|e\rangle) && (1.11)\\ &= \alpha|g\rangle\langle g||g\rangle - \beta|e\rangle\langle e||e\rangle && (1.12)\\ &= \alpha|g\rangle - \beta|e\rangle, && (1.13) \end{aligned}$$

where the orthogonality of $|g\rangle$ and $|e\rangle$ is used. This means that the noise map for dephasing noise leaves the state as it is with propability p, and applies a phase flip with probability $(1-p)$.

The noise maps for depolarising noise (1.7) and dephasing noise (1.10) will be used throughout the thesis as standard noise models.

1.5 Entanglement swaps

Entanglement swaps are one of the fundamental building blocks of a quantum network. They are used in a quantum repeater [11] and make it possible to generate entanglement over distances, which could not be reached using direct transmission. Essentially it is a Bell-state measurement on one qubit of two entangled pairs. This generates a new entangled state between the qubits that were not measured. In this section the noiseless scenario is mathematically derived for entanglement swaps on Bell-states. In general we can define the Bell-states as:

$$|\psi_{a,b}\rangle = (X_1)^a (Z_1)^b \frac{|00\rangle + |11\rangle}{\sqrt{2}}, \tag{1.14}$$

where $a, b \in \{0, 1\}$ are two bits, and X_1 and Z_1 are the standard Pauli operators, where the index 1 indicates that the operations are executed on the first qubit:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad , Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{1.15}$$

Note that it does not matter whether X and Z are applied to the first qubit, or to the second qubit. The resulting state will be the same (up to global phase). This can easily be verified by applying X or Z to either the first or the second qubit of the four Bell-states and checking that the resulting state is always the same up to global phase.

$$\begin{aligned} |\psi_{0,0}\rangle &= \frac{|00\rangle + |11\rangle}{\sqrt{2}} \\ |\psi_{0,1}\rangle &= \frac{|00\rangle - |11\rangle}{\sqrt{2}} \\ |\psi_{1,0}\rangle &= \frac{|01\rangle + |10\rangle}{\sqrt{2}} \\ |\psi_{1,1}\rangle &= \frac{-|01\rangle + |10\rangle}{\sqrt{2}} \end{aligned}$$

If now the nodes A and B share one of these states as well as B and C, then B can make an entanglement swap to make entanglement between A and C. After B makes this measurement, there has to be communication from B to

either A or C to communicate the outcome of the measurement.

Lemma 1. *Let $|\psi_{a,b}\rangle_{AB_1}$ and $|\psi_{c,d}\rangle_{CB_2}$ be two entangled states of the form (1.14) and let B perform an entanglement swap on its qubits B_1 and B_2. Then the resulting state on AC is $|\psi_{a\oplus c,b\oplus d}\rangle_{AC}$ after correction for the outcome of the measurement. The correction for the outcome applied by A to the state $|\psi_{a\oplus c,b\oplus d}\rangle_{AC}$ is the following:*

$$\begin{aligned}
&B \text{ measures } |\psi_{0,0}\rangle \text{ then no correction}\\
&B \text{ measures } |\psi_{0,1}\rangle \text{ then apply } Z\\
&B \text{ measures } |\psi_{1,0}\rangle \text{ then apply } X\\
&B \text{ measures } |\psi_{1,1}\rangle \text{ then apply } XZ
\end{aligned}$$

Proof. Start from the definition of the initial states:

$$|\psi_{a,b}\rangle_{AB_1} \otimes |\psi_{c,d}\rangle_{CB_2} = \frac{1}{2}(X_A)^a(Z_A)^b(|00\rangle + |11\rangle) \otimes (X_C)^c(Z_C)^d(|00\rangle + |11\rangle). \tag{1.16}$$

Here the operators X^a and Z^b act on A, whereas X^c and Z^d act on C. Re-order the terms such that the qubits of B are in the start:

$$\begin{aligned}
|\psi\rangle_{B_1B_2AC} = \frac{1}{2}\Big[&|00\rangle_{B_1B_2} \otimes (X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|00\rangle_{AC} +\\
&|01\rangle_{B_1B_2} \otimes (X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|01\rangle_{AC} +\\
&|10\rangle_{B_1B_2} \otimes (X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|10\rangle_{AC} +\\
&|11\rangle_{B_1B_2} \otimes (X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|11\rangle_{AC}\Big].
\end{aligned} \tag{1.17}$$

Now we can rewrite the qubits of B in the Bell-basis.

$$\begin{aligned}
|\psi\rangle_{B_1B_2AC} &= \frac{1}{2}\Big[(|\psi_{0,0}\rangle + |\psi_{0,1}\rangle)_{B_1B_2} \otimes (X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|00\rangle_{AC} +\\
&\quad (|\psi_{1,0}\rangle + |\psi_{1,1}\rangle)_{B_1B_2} \otimes (X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|01\rangle_{AC} +\\
&\quad (|\psi_{1,0}\rangle - |\psi_{1,1}\rangle)_{B_1B_2} \otimes (X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|10\rangle_{AC} +\\
&\quad (|\psi_{0,0}\rangle - |\psi_{0,1}\rangle)_{B_1B_2} \otimes (X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|11\rangle_{AC}\Big]\\
&= \frac{1}{2}\Big[|\psi_{0,0}\rangle_{B_1B_2} \otimes (X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|\psi_{0,0}\rangle_{AC} +\\
&\quad |\psi_{0,1}\rangle_{B_1B_2} \otimes (X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|\psi_{0,1}\rangle_{AC} +\\
&\quad |\psi_{1,0}\rangle_{B_1B_2} \otimes (X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|\psi_{1,0}\rangle_{AC} +\\
&\quad |\psi_{1,1}\rangle_{B_1B_2} \otimes (X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|\psi_{1,1}\rangle_{AC}\Big] +
\end{aligned} \tag{1.18}$$

Now we can see what happens when B makes the measurement. It will obtain one of the four Bell-states with equal probability. The resulting state between A and C is then also a Bell-state and after the correction of the required form (up to global phase):

$$\begin{aligned}
&(X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|\psi_{0,0}\rangle_{AC} = \pm|\psi_{a\oplus c,b\oplus d}\rangle_{AC} \\
&Z_A(X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|\psi_{0,1}\rangle_{AC} = \pm|\psi_{a\oplus c,b\oplus d}\rangle_{AC} \\
&X_A(X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|\psi_{1,0}\rangle_{AC} = \pm|\psi_{a\oplus c,b\oplus d}\rangle_{AC} \\
&X_AZ_A(X_A)^a(Z_A)^b(X_C)^c(Z_C)^d|\psi_{1,1}\rangle_{AC} = \pm|\psi_{a\oplus c,b\oplus d}\rangle_{AC}
\end{aligned} \tag{1.19}$$

□

This lemma has important implications for entanglement swaps in a quantum network. It means that when doing an entanglement swap on two Bell-states, the resulting state will also always be a Bell-state and we can predict which Bell-state it will be. This means that if there are multiple quantum repeaters that are doing entanglement swaps at the same time, the correction can be applied at any point. For this the correction information of all the entanglement swaps is needed to find the correct final Bell-state. Repeaters do not have to wait for each other to make these entanglement swaps. This result will be used in the chapter about the entanglement tracker, for example to efficiently simulate a repeater chain in section 4.5.

1.6 Entanglement distillation

Entanglement distillation or purification is a protocol that takes a number of weakly entangled states and converts it into a smaller number of strongly entangled states. There are many ways to achieve this, but in this section we will look at distilling two states into one. We will look at a very specific state, of which A and B share two copies:

$$\rho = p|\psi^+\rangle\langle\psi^+| + (1-p)|11\rangle\langle 11|, \tag{1.20}$$

with $|\psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$ being the maximally entangled state. This state occurs in the EPL generation protocol as well, as will be discussed in the next chapter in the case of more noise. For more information about the EPL-protocol see [12].

A $CNOT$-operation can be defined using the following maps of the elements of the standard basis:

Algorithm 1 EPL distillation protocol

Let ρ_1 and ρ_2 from (1.20) be two entangled states shared between A and B.

1. Do a $CNOT$-operation with the qubits of ρ_1 as control and the qubits of ρ_2 as target
2. Measure the qubits of ρ_2 in the standard basis
3. If both A and B measure $|1\rangle$ on their qubits, call success, otherwise call fail.

$$
\begin{aligned}
CNOT_{1\to 2}|00\rangle &\to |00\rangle \\
CNOT_{1\to 2}|01\rangle &\to |01\rangle \\
CNOT_{1\to 2}|10\rangle &\to |11\rangle \\
CNOT_{1\to 2}|11\rangle &\to |10\rangle
\end{aligned}
$$

The first qubit is called the control qubit and the second one the target. The operation can be understood by seeing that when the control qubit is in the $|1\rangle$ state, the operation makes the target bit flip and if the control bit is in the $|0\rangle$ state it does nothing. The notation $CNOT_{1\to 2}$ then means doing a $CNOT$-operation with the first qubit as control and the second qubit as target.

Lemma 2. *The EPL-distillation protocol succeeds with probability $p^2/2$ and the final state will always be $|\psi^+\rangle$.*

Proof. Analyse the effect of step 1 by looking at the different components of the mixture separately. The first component is when both states are in the $|\psi^+\rangle$ state, this has a coefficient in the mixture of p^2.

$$
\begin{aligned}
&\frac{1}{2} CNOT_{1\to 3} CNOT_{2\to 4} (|01\rangle + |10\rangle)(|01\rangle + |10\rangle) \\
&= \frac{1}{2}\big[|01\rangle(|00\rangle + |11\rangle) + |10\rangle(|11\rangle + 00\rangle)\big] \qquad (1.21) \\
&= \frac{1}{2}\big[(|01\rangle + |10\rangle)|00\rangle + (|01\rangle + |10\rangle)|11\rangle\big]
\end{aligned}
$$

The $|11\rangle|\psi^+\rangle$ component has coefficient $p(1-p)$:

$$
\begin{aligned}
&\frac{1}{\sqrt{2}} CNOT_{1\to 3} CNOT_{2\to 4} |11\rangle(|01\rangle + |10\rangle) \qquad (1.22) \\
&= \frac{1}{\sqrt{2}}(|11\rangle|10\rangle + |11\rangle|01\rangle) \qquad (1.23)
\end{aligned}
$$

The $|\psi^+\rangle|11\rangle$ component also has coefficient $p(1-p)$:

$$\frac{1}{\sqrt{2}} CNOT_{1\to 3} CNOT_{2\to 4}(|01\rangle + |10\rangle)|11\rangle \tag{1.24}$$

$$= \frac{1}{\sqrt{2}}(|01\rangle|10\rangle + |10\rangle|01\rangle) \tag{1.25}$$

Finally the $|11\rangle|11\rangle$ has a coefficient of $(1-p)^2$:

$$CNOT_1 CNOT_2 |11\rangle|11\rangle = |11\rangle|00\rangle \tag{1.26}$$

It can be seen that only the first component of the mixture allows for both the third and the fourth qubit to be in the $|1\rangle$ state. This means that if we measure $|11\rangle$, we know that the resulting state is $\frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) = |\psi^+\rangle$. This happens with the probability of the according component of the mixture times the probability to measure $|11\rangle$ if we are in that component. The probability of measuring $|11\rangle$ is $1/2$ and the component in the mixture is p^2, so the total probability of measuring $|11\rangle$ is $p^2/2$. □

This lemma shows that if the noise in the state is of a very specific form, in this case orthogonal to the Bell-state, distillation can reproduce a maximally entangled state. In general this distillation protocol can also work for noise not orthogonal to the Bell-state. In that case the EPL-distillation protocol will not recover a maximally entangled state, but still a noisy state. Two copies of this noisy state can then be distilled to obtain a state with more entanglement using the DEJMPS protocol [13].

For more examples about distillation, including numerical values see section 4 of [15]. It is then possible to generate more noisy entangled states and keep doing distillation, this is also referred to as entanglement pumping [16]. Doing multiple rounds of distillation will also be analysed in section 2.4.

In the previously discussed protocols for entanglement swaps and distillation no noise was taken into account in the operations. However, this is one of the main challenges of making a quantum network. Distillation can be used to increase the entanglement, however performing this protocol introduces new noise. If the noise the protocol adds is more than we gain from entanglement distillation, this protocol is essentially useless. This also means that instead of having perfect Bell-states, in reality we always have to deal with noisy entangled states. Instead of performing perfect measurements and quantum operations, we always have noisy measurements and operations.

Algorithm 2 DEJMPS distillation protocol

Let ρ_1 and ρ_2 be two entangled states shared between A and B, with fidelity to a Bell-state of higher than 0.5 .

1. Perform local operations on both ρ_1 and ρ_2 so that the two states are of the form:

$$\rho_i = p_{1,i}|\phi^+\rangle\langle\phi^+| + p_{2,i}|\psi^+\rangle\langle\psi^+| + p_{3,i}|\phi^-\rangle\langle\phi^-| + p_{4,i}|\psi^-\rangle\langle\psi^-|,$$

   with $p_{1,i} > 0.5 > p_{2,i} \geq p_{3,i} \geq p_{4,i}$ and $p_{1,i} + p_{2,i} + p_{3,i} + p_{4,i} = 1$ for $i = 1, 2$. Using this order of the Bell-states, the highest boost in fidelity to $|\phi^+\rangle$ can be achieved [14].

2. Rotate the two qubits of A around the X axis by $\pi/2$ and the two qubits of B around the X axis by $-\pi/2$

3. Do a $CNOT$-operation with the qubits of ρ_1 as control and the qubits of ρ_2 as target.

4. Measure the qubits of ρ_2 in the standard basis and communicate the outcomes.

5. If both A and B measure $|1\rangle$ on their qubits or both measure $|0\rangle$, call success, otherwise call fail.

Chapter 2

EPL entanglement generation

This chapter is about the EPL-scheme, which is a protocol to distill entanglement from an imperfect state [12]. The most basic form of the states that are generated before distillation is:

$$\rho = p|\theta_\phi\rangle\langle\theta_\phi| + (1-p)|11\rangle\langle 11|, \tag{2.1}$$

with $|\theta_\phi\rangle = (|01\rangle + e^{i\phi}|10\rangle)/\sqrt{2}$.

The phase ϕ is unknown, which means that the entanglement in the state ρ is useless. However, if we have two copies of the state ρ and they have the same phase ϕ, the EPL-scheme can successfully get rid of the unknown phase.

For more background on how these states are produced experimentally, see for example [20], where the whole process is realised in NV-centres. This includes generation of states ρ and distilling them to get rid of the unwanted phase ϕ.

The new additions to the previously established theory start in the next section by analysing the EPL-scheme in the case of dephasing noise. After that, the maximum obtainable rates are examined as a function of the available memories to store qubits. Finally the interaction between entanglement swaps and performing this EPL-distillation will be investigated, for different memory models.

2.1 EPL with dephasing noise

In this section we add an extra term indicating dephasing noise. We start by defining the following state:

$$\rho = p\Big[p_d|\theta_\phi^+\rangle\langle\theta_\phi^+| + (1-p_d)|\theta_\phi^-\rangle\langle\theta_\phi^-|\Big] + (1-p)|11\rangle\langle 11| \tag{2.2}$$

with $|\theta_\phi^\pm\rangle = (|01\rangle \pm e^{i\phi}|10\rangle)/\sqrt{2}$. The value of ϕ is introduced by the physical channel during the generation process and unknown, but assumed the same for every use. The values of p and p_d are related to the amount of noise.

The goal is to use two of these states to make a state as close as possible to the Bell-state $|\psi^+\rangle = (|01\rangle + |10\rangle)/\sqrt{2}$.

Lemma 3. *Take two states ρ_1 and ρ_2 from (2.2) and do CNOT gates using the qubits of ρ_1 as control qubits and the qubits of ρ_2 as target qubits and measure both of the qubits of ρ_2 in the standard basis. The resulting state after getting measurement outcome $|11\rangle$ is $\rho_{11} = (p_d^2 + (1-p_d)^2)|\psi^+\rangle\langle\psi^+| + 2p_d(1-p_d)|\psi^-\rangle\langle\psi^-|$ and the probability of getting measurement outcome $|11\rangle$ is $p^2/2$.*

Proof. Look at the effect of the CNOT operations by considering each pure state component of $\rho \otimes \rho$. Working with unnormalized states, the result for the state $|\theta_\phi^-\rangle|\theta_\phi^-\rangle$ is:

$$\begin{aligned} &CNOT_1CNOT_2(|01\rangle - e^{i\phi}|10\rangle)(|01\rangle - e^{i\phi}|10\rangle) \\ &= |01\rangle(|00\rangle - e^{ß\phi}|11\rangle) - e^{i\phi}|10\rangle(|11\rangle - e^{i\phi}|00\rangle) \\ &= (|01\rangle + e^{2i\phi}|10\rangle)|00\rangle - e^{i\phi}(|01\rangle + |10\rangle)|11\rangle \end{aligned} \tag{2.3}$$

for $|\theta_\phi^+\rangle|\theta_\phi^+\rangle$:

$$\begin{aligned} &CNOT_1CNOT_2(|01\rangle + e^{i\phi}|10\rangle)(|01\rangle + e^{i\phi}|10\rangle) \\ &= (|01\rangle + e^{2i\phi}|10\rangle)|00\rangle + e^{i\phi}(|01\rangle + |10\rangle)|11\rangle \end{aligned} \tag{2.4}$$

for $|\theta_\phi^+\rangle|\theta_\phi^-\rangle$:

$$\begin{aligned} &CNOT_1CNOT_2(|01\rangle + e^{i\phi}|10\rangle)(|01\rangle - e^{i\phi}|10\rangle) \\ &= |01\rangle(|00\rangle - e^{ß\phi}|11\rangle) + e^{i\phi}|10\rangle(|11\rangle - e^{i\phi}|00\rangle) \\ &= (|01\rangle - e^{2i\phi}|10\rangle)|00\rangle - e^{i\phi}(|01\rangle - |10\rangle)|11\rangle \end{aligned} \tag{2.5}$$

for $|\theta_\phi^-\rangle|\theta_\phi^+\rangle$:

$$\begin{aligned} &CNOT_1CNOT_2(|01\rangle - e^{i\phi}|10\rangle)(|01\rangle + e^{i\phi}|10\rangle) \\ &= |01\rangle(|00\rangle + e^{ß\phi}|11\rangle) - e^{i\phi}|10\rangle(|11\rangle + e^{i\phi}|00\rangle) \\ &= (|01\rangle - e^{2i\phi}|10\rangle)|00\rangle + e^{i\phi}(|01\rangle - |10\rangle)|11\rangle \end{aligned} \tag{2.6}$$

for $|11\rangle|11\rangle$:

$$CNOT_1CNOT_2|11\rangle|11\rangle = |11\rangle|00\rangle \tag{2.7}$$

for $|\theta_\phi^\pm\rangle|11\rangle$:

$$CNOT_1CNOT_2(|01\rangle \pm e^{i\phi}|10\rangle)|11\rangle = |01\rangle|10\rangle \pm e^{i\phi}|10\rangle|01\rangle \tag{2.8}$$

and for $|11\rangle|\theta_\phi^\pm\rangle$:

$$CNOT_1CNOT_2|11\rangle(|01\rangle \pm e^{i\phi}|10\rangle) = |11\rangle|10\rangle \pm e^{i\phi}|11\rangle|01\rangle \tag{2.9}$$

Colecting all the terms for measurement outcome $|11\rangle$ on ρ_2:

$$\begin{aligned}\rho_{11} &= p^2e^{i\phi}e^{-i\phi}\Big[(p_d^2+(1-p_d)^2)(|01\rangle+|10\rangle)(\langle 01|+\langle 10|) \\ &+ 2p_d(1-p_d)(|01\rangle-|10\rangle)(\langle 01|-\langle 10|)\Big] \\ &= (p_d^2+(1-p_d)^2)|\psi^+\rangle\langle\psi^+| + 2p_d(1-p_d)|\psi^-\rangle\langle\psi^-|\end{aligned} \tag{2.10}$$

The only states with $|11\rangle$ on the last two qubits originate from $|\theta_\phi^\pm\rangle|\theta_\phi^\pm\rangle$ terms. We get these kind of products with probability p^2 as can be seen in (2.2). The probablity of measuring $|00\rangle$ is exactly the same as measuring $|11\rangle$; this gives a probability of $p^2/2$ of measuring $|11\rangle$. □

We can also take a look at what happens when measuring $|00\rangle$ on the right-side qubits. Then the remaining unnormalized state on the left qubits is:

$$p^2/2\Big[(p_d^2+(1-p_d)^2)|\theta_{2\phi}^+\rangle\langle\theta_{2\phi}^+| + 2p_d(1-p_d)|\theta_{2\phi}^-\rangle\langle\theta_{2\phi}^-|\Big] + (1-p)^2|11\rangle\langle 11| \tag{2.11}$$

with $|\theta_{2\phi}^\pm\rangle = (|01\rangle \pm e^{2i\phi}|10\rangle)/\sqrt{2}$
and normalized:

$$\rho_{00} = \frac{p^2/2\Big[(p_d^2+(1-p_d)^2)|\theta_{2\phi}^+\rangle\langle\theta_{2\phi}^+| + 2p_d(1-p_d)|\theta_{2\phi}^-\rangle\langle\theta_{2\phi}^-|\Big] + (1-p)^2|11\rangle\langle 11|}{p^2/2+(1-p)^2} \tag{2.12}$$

2.2 EPL-scheme: maximum achievable rates

We now set $p_d = 1$,which means leaving out the dephasing term in (2.2). The state then becomes:

$$\rho = p|\theta_\phi^+\rangle\langle\theta_\phi^+| + (1-p)|11\rangle\langle 11| \tag{2.13}$$

with with $|\theta_\phi^+\rangle = (|01\rangle + e^{i\phi}|10\rangle)/\sqrt{2}$ and ϕ an unknown phase.

If we revisit the density matrix ρ_{00} in (2.12) and enter $p_d = 1$, the density matrix becomes

$$\rho_{00} = \frac{p^2/2|\theta_{2\phi}^+\rangle\langle\theta_{2\phi}^+| + (1-p)^2|11\rangle\langle 11|}{p^2/2+(1-p)^2} \tag{2.14}$$

with fidelity to the state $|\psi^+\rangle$:

$$F_{00} = \frac{p^2/2}{p^2/2+(1-p)^2}(\frac{1}{2}+\frac{1}{2}\cos 2\phi). \tag{2.15}$$

The goal is now to use these states ρ_{00} to generate more $|\psi^+\rangle$ states. Note that the fidelity F_{00} is below 0.5 if we do not know the value of ϕ and we average uniformly over it. The idea is now to use two of the states with double the phase to get $|\psi^+\rangle$.

Introduce the new notation $c_\theta^1 = p$ and $c_{11}^1 = 1 - p$. Whenever a quantity is squared in the rest of this section, it is always displayed between brackets to distinguish from the superscripts. In this new notation:

$$\rho = c_\theta^1 |\theta_\phi^+\rangle\langle\theta_\phi^+| + c_{11}^1 |11\rangle\langle 11| \tag{2.16}$$

and ρ_{00} becomes:

$$\rho_{00} = \frac{(c_\theta^1)^2/2 |\theta_{2\phi}^+\rangle\langle\theta_{2\phi}^+| + (c_{11}^1)^2 |11\rangle\langle 11|}{(c_\theta^1)^2/2 + (c_{11}^1)^2} \tag{2.17}$$

$$= c_\theta^2 |\theta_{2\phi}^+\rangle\langle\theta_{2\phi}^+| + c_{11}^2 |11\rangle\langle 11| \tag{2.18}$$

with the coefficients:

$$c_\theta^2 := \frac{(c_\theta^1)^2/2}{(c_\theta^1)^2/2 + (c_{11}^1)^2} \qquad c_{11}^2 := \frac{(c_{11}^1)^2}{(c_\theta^1)^2/2 + (c_{11}^1)^2} \tag{2.19}$$

Using the result of the previous section, the probability of measuring $|11\rangle$ using two states with twice the phase is $p_{11}^2 := (c_\theta^2)^2/2$. Similarly the probability of measuring $|00\rangle$ is $p_{00}^2 := (c_\theta^2)^2/2 + (c_{11}^2)^2$. If we get $|00\rangle$ we can continue in the same way with a state with 4 times the phase. In general the coefficients c_θ^n and c_{11}^n are:

$$c_\theta^{n+1} := \frac{(c_\theta^n)^2/2}{(c_\theta^n)^2/2 + (c_{11}^n)^2} \qquad c_{11}^{n+1} := \frac{(c_{11}^n)^2}{(c_\theta^n)^2/2 + (c_{11}^n)^2} \tag{2.20}$$

The state after measuring $|00\rangle$, n times is then:

$$\rho_{00}^n = c_\theta^{n+1} |\theta_{2^n\phi}^+\rangle\langle\theta_{2^n\phi}^+| + c_{11}^{n+1} |11\rangle\langle 11| \tag{2.21}$$

with the probabilities of obtaining the states:

$$p_{11}^n = (c_\theta^n)^2/2 \qquad p_{00}^n = (c_\theta^n)^2/2 + (c_{11}^n)^2 \tag{2.22}$$

Now we can take a look at the rate at which maximally entangled states can be generated. For this we need to define the number of available memories. In any case we need two memories to store two copies of ρ, to be able to perform the EPL scheme. In addition we may have memories to store the resulting states after measuring $|00\rangle$, an overview of the possibilities given an amount of memories can be found in Figure 2.1.

For example, if we have four memories in total, we can use two to perform the basic step of the EPL scheme. The other two memories can then be used

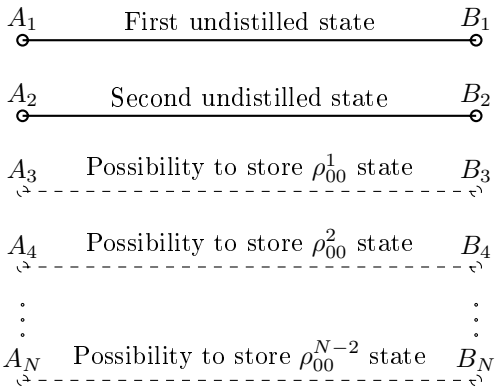


Figure 2.1: Overview of the quantum memories for A and B, numbered by $A_1, .., A_N$ and $B_1, .., B_4$. The minimum amount of memories required, to be able to generate two new states and perform distillation, is two. If there are N memories available, the states up to ρ_{00}^{N-2} can be stored and still have enough memories to generate another ρ_{00}^{N-2} state for distillation. Whenever an entangled pair is successfully generated, it is stored in one of the memories, in which case the protocol continues with one less memory. This means that now the states up to ρ_{00}^{N-3} can be stored. It is also possible to directly consume the distilled state, for example by using it to run quantum key distribution. In that case the protocol runs with a fixed amount of memories even after successful distillation.

to store the ρ_{00}^1 and ρ_{00}^2 states in (2.21).

The memories also have to be used to store the maximally entangled states we obtain after measuring $|11\rangle$ at any point. This means that each time we have a success in the protocol, one less memory is available to store ρ_{00}^n states.

In order to find the maximum achievable rates using the EPL scheme, we define a rate as follows, within this section:

$$R = \frac{\#|\psi^+\rangle}{\#\rho}. \tag{2.23}$$

The rate is thus given by the number of Bell-pairs that are obtained divided by the amount of initial states ρ from (2.13) that were consumed in the process.

As a first step we look at the rate R_M^* that can be obtained using M available memories and whenever success is obtained the resulting state is consumed. For example, when we have $M = 2$, we can not store anything and we have to measure $|11\rangle$ on the initial states. The rate then becomes $R_2^* = p_{11}^1/2 = p^2/4$. The factor $1/2$ indicates that we used two ρ states to get one $|\psi^+\rangle$ state.

What happens for $M = 3$ can be seen when we consider N rounds of performing the EPL scheme. On average we get Np_{00}^1 times the state ρ_{00}^1. Using two ρ_{00}^1 states we get $|\psi^+\rangle$ with probability p_{11}^2. The rate R_3^* becomes:

$$R_3^* = R_2^* + \frac{Np_{00}^1 p_{11}^2/2}{2N} = R_2^* + \frac{p_{00}^1 p_{11}^2}{4} \tag{2.24}$$

$$= p^2/4 + \Big(p^2/2 + (1-p)^2\Big)\Big(\frac{p^4/8}{(p^2/2 + (1-p)^2)^2}\Big)/4 \tag{2.25}$$

The factor $2N$ indicates that $2N$ initial copies from (2.13) were used and the factor $1/2$ that two ρ_{00}^1 states are needed to get one success.

Continuing for $M = 4$ we have the same terms as for $M = 3$ but this time we can also store ρ_{00}^2. Then we need four times the ρ_{00}^1 state to make two ρ_{00}^2 states and measure $|11\rangle$.

$$R_4^* = R_3^* + \frac{Np_{00}^1 p_{00}^2 p_{11}^3/4}{2N} = R_3^* + \frac{p_{00}^1 p_{00}^2 p_{11}^3}{8} \tag{2.26}$$

This result can be generalised, as for R_M^* the extra term is measuring $|00\rangle$ the first $M-2$ times and then measuring $|11\rangle$.

$$R_{M+1}^* = R_M^* + \frac{p_{11}^M}{2^M} \prod_{i=1}^{M-1} p_{00}^i \tag{2.27}$$

If we start with $R_1^* = 0$ then the previous results follow from (2.27).

We can now look at the case where we start with M memories, but now each time we generate a $|\psi^+\rangle$ state, we store it and so we lose one available memory. Then in the start the rate is R^*_M, but once we get one success, the rate drops to R^*_{M-1}. We can continue this way until there are two available memories left and the rate has dropped to R^*_2. Then when we measure $|11\rangle$ our memory is filled up with $M-1$ states $|\psi^+\rangle$. On average the number of elementary states needed to get one success using M memories is $1/R^*_M$, so the total amount of elementary states N_M needed to fill up $M-1$ memories is:

$$N_M = 1/R^*_M + 1/R^*_{M-1} + ... + 1/R^*_2 = \sum_{m=2}^{M} 1/R^*_m \tag{2.28}$$

Using those N_M elementary states, the number of successful output states was $M-1$ which means that the rate R_M becomes:

$$R_M = \frac{M-1}{N_M} = \frac{M-1}{\sum_{m=2}^{M} \frac{1}{R^*_m}} \tag{2.29}$$

2.3 Entanglement swapping and distillation

We now look at repeater schemes, where we want to use multiple repeaters to create long distance entanglement. For example A and C_1 have an entangled state and B and C_2 also have an entangled state, where C_1 and C_2 are in the same location. It is now possible to use these two entangled states to create entanglement between A and B. We assume that both the entangled states are created by the EPL scheme, discussed in previous sections:

$$\rho_{AC_1} = p|\theta^+_{\phi_1}\rangle\langle\theta^+_{\phi_1}| + (1-p)|11\rangle\langle 11| \tag{2.30}$$

and

$$\rho_{C_2B} = p|\theta^+_{\phi_2}\rangle\langle\theta^+_{\phi_2}| + (1-p)|11\rangle\langle 11|, \tag{2.31}$$

where $|\theta^\pm_{\phi_j}\rangle = (|01\rangle + e^{i\phi_j}|10\rangle)/\sqrt{2}$. Note that in general $\phi_1 \neq \phi_2$, because these phases are determined by the fibre. We assume for simplicity that the probability p is the same for both connections.

Lemma 4. *Measuring $|\psi^+\rangle$ or $|\psi^-\rangle$ in a Bell-state measurement on C_1 and C_2 as defined in* (2.30) *and* (2.31) *happens with probability $\frac{1}{2}(2p-p^2)$ and will result in $\rho_{AB} = p'|\theta^+_{\phi_1+\phi_2}\rangle\langle\theta^+_{\phi_1+\phi_2}| + (1-p')|11\rangle\langle 11|$ after a correction for the measurement outcome and where $p' = \frac{p^2}{2p-p^2}$.*

Proof. The total state of the system is $\rho_{AC_1} \otimes \rho_{C_2B}$. We will look at this mixed state term by term, starting with the part $|\theta^+_{\phi_1}\rangle\langle\theta^+_{\phi_1}| \otimes |\theta^+_{\phi_2}\rangle\langle\theta^+_{\phi_2}|$. This unnormalized state is:

$$\begin{aligned}\phi_{AC_1C_2B} &= (|01\rangle + e^{i\phi_1}|10\rangle) \otimes (|01\rangle + e^{i\phi_2}|10\rangle) \\ &= |01\rangle|01\rangle + e^{i\phi_1}|10\rangle|01\rangle + e^{i\phi_2}|01\rangle|10\rangle) + e^{i(\phi_1+\phi_2)}|10\rangle|10\rangle\end{aligned} \tag{2.32}$$

Reorder the qubits as C_1C_2AB and rewrite in the Bell-basis for C_1C_2.

$$\begin{aligned}\phi_{C_1C_2AB} &= |10\rangle|01\rangle + e^{i\phi_1}|00\rangle|11\rangle + e^{i\phi_2}|11\rangle|00\rangle) + e^{i(\phi_1+\phi_2)}|01\rangle|10\rangle \\ &= (|\psi^+\rangle - |\psi^-\rangle)|01\rangle + e^{i\phi_1}(|\phi^+\rangle + |\phi^-\rangle)11\rangle \\ &+ e^{i\phi_2}(|\phi^+\rangle - |\phi^-\rangle)|00\rangle) + e^{i(\phi_1+\phi_2)}(|\psi^+\rangle + |\psi^-\rangle)|10\rangle \\ &= |\psi^+\rangle(|01\rangle + e^{i(\phi_1+\phi_2)}|10\rangle) + |\psi^-\rangle(-|01\rangle + e^{i(\phi_1+\phi_2)}|10\rangle) \\ &+ |\phi^+\rangle(e^{i\phi_1}|11\rangle + e^{i\phi_2}|00\rangle) + |\phi^-\rangle(e^{i\phi_1}|11\rangle - e^{i\phi_2}|00\rangle)\end{aligned} \tag{2.33}$$

With $|\phi^\pm\rangle = |00\rangle \pm |11\rangle$ and $|\psi^\pm\rangle = |01\rangle \pm |10\rangle$.

The next term is $|\theta_1^+\rangle\langle\theta_1^+| \otimes |11\rangle\langle 11|$, for which the state is:

$$\phi_{C_1C_2AB} = |11\rangle|01\rangle + e^{i\phi_1}|01\rangle|11\rangle = (|\phi^+\rangle - |\phi^-\rangle)|01\rangle + e^{i\phi_1}(|\psi^+\rangle + |\psi^-\rangle)|11\rangle. \tag{2.34}$$

Similarly for $|11\rangle\langle 11| \otimes |\theta_1^+\rangle\langle\theta_1^+|$, the state becomes:

$$\phi_{C_1C_2AB} = |10\rangle|11\rangle + e^{i\phi_2}|11\rangle|10\rangle = (|\psi^+\rangle - |\psi^-\rangle)|11\rangle + e^{i\phi_2}(|\phi^+\rangle - |\phi^-\rangle)|10\rangle. \tag{2.35}$$

And finally for $|11\rangle\langle 11| \otimes ||11\rangle\langle 11|$:

$$\phi_{C_1C_2AB} = |11\rangle|11\rangle = (|\phi^+\rangle - |\phi^-\rangle)|11\rangle \tag{2.36}$$

Now collect all the terms and order them based on the outcome of the Bell-measurement. The final states between A and B are then:

$$\rho_{AB,\psi^+} = \frac{1}{4}\frac{p^2|\theta^+_{\phi_1+\phi_2}\rangle\langle\theta^+_{\phi_1+\phi_2}| + 2p(1-p)|11\rangle\langle 11|}{\frac{1}{4}(2p-p^2)}, \tag{2.37}$$

$$\rho_{AB,\psi^-} = \frac{1}{4}\frac{p^2|\theta^-_{\phi_1+\phi_2}\rangle\langle\theta^-_{\phi_1+\phi_2}| + 2p(1-p)|11\rangle\langle 11|}{\frac{1}{4}(2p-p^2)}, \tag{2.38}$$

$$\begin{aligned}\rho_{AB,\phi^+} = \frac{1}{4}\Big[&p^2/2(e^{i\phi_1}|11\rangle + e^{i\phi_2}|00\rangle)(e^{-i\phi_1}\langle 11| + e^{-i\phi_2}\langle 00|) \\ &+ p(1-p)|01\rangle\langle 01| + p(1-p)|10\rangle\langle 10| + 2(1-p)^2|11\rangle\langle 11|\Big]/\Big[\frac{1}{4}((p-1)^2+1)\Big],\end{aligned} \tag{2.39}$$

$$\rho_{AB,\phi^-} = \frac{1}{4}\Big[p^2/2(e^{i\phi_1}|11\rangle - e^{i\phi_2}|00\rangle)(e^{-i\phi_1}\langle 11| - e^{-i\phi_2}\langle 00|) + p(1-p)|01\rangle\langle 01| + p(1-p)|10\rangle\langle 10| + 2(1-p)^2|11\rangle\langle 11|\Big] / \Big[\frac{1}{4}((p-1)^2+1)\Big]. \tag{2.40}$$

Here $\rho_{AB,x}$ is the state A and B share after a measurement outcome x on C. Applying a Z-gate to either of the qubits of ρ_{ψ^+} transforms ρ_{ψ^-} to ρ_{ψ^+}. The probability of obtaining ρ_{ψ^-} or ρ_{ψ^+} is given by the denominator in the density matrix: $p_{succ} = \frac{1}{4}(2p-p^2) + \frac{1}{4}(2p-p^2) = \frac{1}{2}(2p-p^2)$

□

Afterwards we still have to get rid of the unknown phase by performing distillation on two entangled pairs between A and B. We can not do this if one of the pairs comes from measuring $\rho_{\psi^\pm}$ and the other one $\rho_{\phi^\pm}$. The reason for this is that for $\rho_{\psi^\pm}$ the phases ϕ_1 and ϕ_2 are added, whereas for $\rho_{\phi^\pm}$ the phases are subtracted. This difference between $e^{\phi_1+\phi_2}$ and $e^{\phi_1-\phi_2}$ can not be solved by distillation any more. This means that we have to pick either two pairs of $\rho_{\psi^\pm}$ or two pairs of $\rho_{\phi^\pm}$.

Note that in this case a Z-gate also transforms the state ρ_{ϕ^-} to ρ_{ϕ^+}. Then, however, the noise is not orthogonal to the Bell-state and therefore we will not be able to get rid of it by performing the EPL-scheme. For high values of p it is still possible to get a good entangled state between A and B. In general the states $\rho_{\psi^\pm}$ are less noisy so would be preferred.

We can now analyse the difference between doing distillation first and doing the entanglement swapping first. The two possible protocols to establish entanglement between A and B are displayed in Figure 2.2. For this we assume a memory which is only coherent for one time step. This models a network in which nodes do not have time to communicate the results of their operations. This reduces the probability of successfully obtaining entanglement over large distances, but increases the fidelity if creating this entanglement succeeds.

In the most simple case the time step starts with two states ρ_{AC_1} and two states ρ_{C_2B} in (2.30) and (2.31). Here we assume that these states are already generated, in section section 3.3 we will look at how to include entanglement generation in this process. Then either we perform distillation first or we start by doing the entanglement swap. After that either entanglement swapping is done on a Bell-state, or distillation is attempted on the two states that were swapped.

Lemma 5. *Two states ρ_{AC_1} and two states ρ_{C_2B} in* (2.30) *and* (2.31) *can be used to probabilistically obtain a perfect Bell-state between A and B. However doing distillation first has twice the success probability of doing the swap first, $p_{dist,swap} = 2p_{swap,dist}$.*

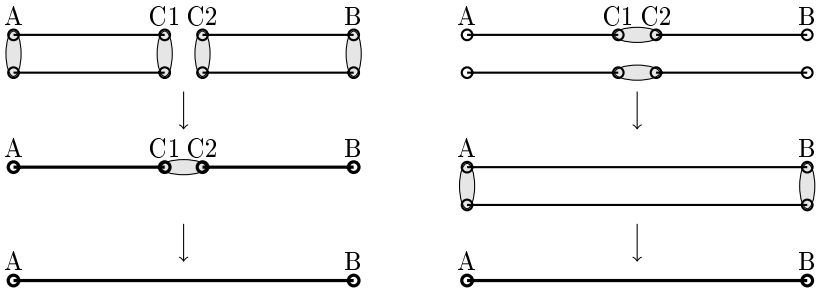


Figure 2.2: Schematic overview of the two different options to obtain one distilled link between A and B. Shaded areas indicate the next step of the protocol, horizontal being an entanglement swap and vertical entanglement distillation. The first option is to do distillation first between A and $C1$ as well as between B and $C2$ and then a single entanglement swap at C, which is displayed on the left. The second option is to do two entanglement swaps at C and then perform distillation between A and B, as displayed on the right.

Proof. Doing distillation first results in a Bell-state with success probability $p^2/2$ as proven in lemma (3). We need a successful outcome at the link A to C_1 as well as C_2 to B, after that the entanglement swap can be done deterministically. Thus the total probability of success when doing distillation first is $p_{dist,swap} = p^4/4$.

In the case of doing the swap first, we can use lemma (4) to get the probability of measuring ψ_+ or ψ_- at C, which is $\frac{1}{2}(2p - p^2)$. We need the entanglement swap to succeed on both the initial pairs, which happens with probability $\frac{1}{4}(2p - p^2)^2$. After that the distillation is successful with probability $p'^2/2$, where $p' = \frac{p^2}{2p-p^2}$ as in equation (2.37). This means that the total probability is $p_{swap,dist} = \frac{1}{8}(2p - p^2)^2 \frac{p^4}{(2p-p^2)^2} = p^4/8$.

We do not use the states $\rho_{AB,\phi^\pm}$ from lemma 4 because they will never result in a Bell-state after distillation. We see that doing distillation first is always optimal and will have twice the success probability, $p_{dist,swap} = 2p_{swap,dist}$. □

2.4 Numerical experiments

In this section some of the previously discussed cases are simulated using MATLAB. For the first few results the simulation data can be compared to the analytical results from section 2.2. Both the simulation and the analytical result assume perfect memories and operations. The rest of the simulations include some noise and looks at the possibility to pump the entanglement to a high fidelity state. Entanglement pumping means doing an entanglement distillation process repeatedly, by keeping one entangled state in memory and generating

new entangled state to distil. Every time the newly generated entanglement is distilled into the stored state, another entangled pair can be generated. This process can be repeated, but competes against decoherence. Based on the noise parameters there is a maximum fidelity that can be achieved using entanglement pumping, after which additional pumping will not increase the entanglement.

As a first result we will look at the rate of successful distillation as a function of the number of available memories for different values of p in the state:

$$\rho = p|\theta^+\rangle\langle\theta^+| + (1-p)|11\rangle\langle 11|, \tag{2.41}$$

with $|\theta^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + e^{i\phi}|10\rangle$ as defined in section 2.2. Each time there is a success, we put the resulting maximally entangled state in a memory and therefore continue the protocol with one less memory. The protocol ends once there is only one memory left that is not filled, because then there can be no distillation any more. The rate goes up as a function of the amount of memories, because if we have more free memories we can use the measurement result $|00\rangle$ and not just $|11\rangle$.

In Figure 2.3 there is a slight difference between the analytical and simulation result. The analytical result calculates the rate that you would get if for every number of memories you would generate a large number of Bell-pairs. The first Bell-pair takes longer to generate than the following ones, because for the later ones we can use the $|00\rangle$ outcomes that were left over from generating the previous copies. In the simulation there is only one Bell-pair that is generated for each number of memories, which is the first and slowest one.

For example, in the analytical calculation when we have four memories we assume that we will keep generating a large number of new copies and removing the distilled states after success. Then we do the same for three memories and for two memories, in the end we add all the contributions up to get the rate for filling four memories. Effectively this means that for each number of memories we generate a large number of distilled states and not just one, as would be the case in the simulation.

Another example, in the case of running the simulation: we start with three memories and after one round we get a $|00\rangle$ outcome, resulting in a state with twice the phase. This state can then be stored in one of the memories and we generate two new states for distillation. If the result of this distillation is then a Bell-state, this should be stored in a memory. Then if we want to continue the protocol we have to throw out the state resulting from $|00\rangle$ to generate two new states for distillation. In the analytical case we would have been able to use this $|00\rangle$ outcome, because we remove the Bell-state from our memory after success.

This also explains why the rates using the simulation and analytical result

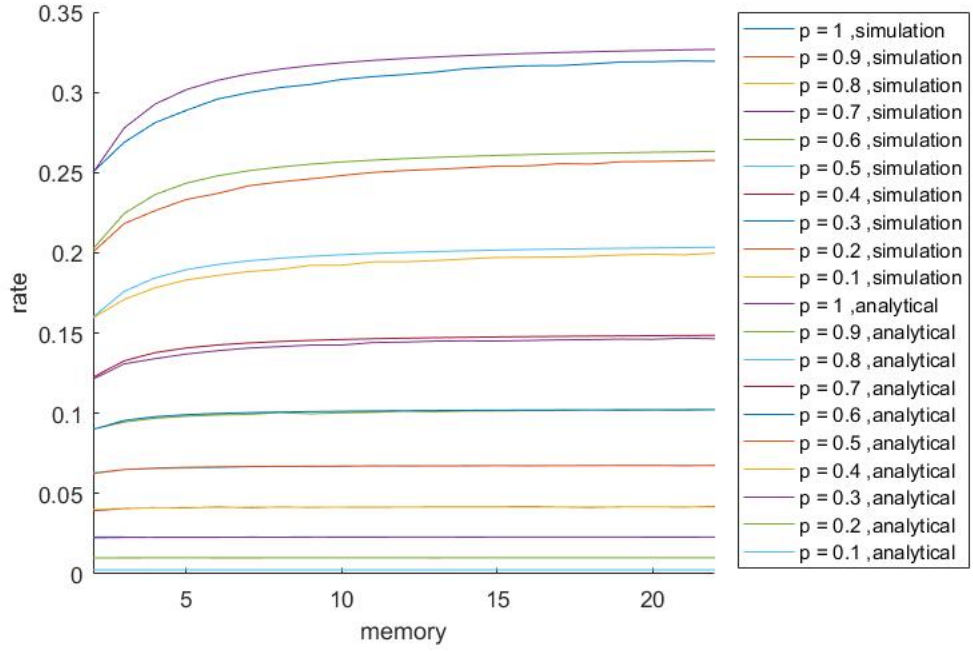


Figure 2.3: Simulation of the EPL scheme. The x-axis is the amount of memories that are available, where 2 is the minimum to perform distillation. The y-axis shows the rate at which maximally entangled states can be generated and is calculated by dividing the number of successfully distilled states by the amount of initial states used. Comparison of simulation (top 10 lines in legend) with analytical solution (bottom 10 lines in legend) of the rate in the model where the memories start filling up.

are equal when using two memories, because in that case we can never use the $|00\rangle$ outcome states. When using two memories, the rates are given by lemma (3), the rate of success is then $p^2/2$. Each attempt consumes two copies, so that means that when $p = 1$ the rate becomes $1/4$.

The above analysis considered the case without noise and decoherence. Now we can include some noisy operations and storage. The goal of the simulation is to achieve an entangled state with at least a specified fidelity to Bell-state. Decoherence then happens in every round, where a round can be defined by two phases:

1. **Generating**: Generate two new noisy entangled states of the form (2.41) and put them in two free memories. If there are not enough free memories, throw away a $|00\rangle$ outcome state with the maximum phase.

2. **Distilling**: Distil the two noisy states. Depending on the outcome of the distillation, we may throw away the resulting state, store the state or do further distillation. Further EPL-distillation happens when we measure $|00\rangle$ and there is a state of the same phase in one of the other memories. DEJMPS-distillation happens when we measure $|11\rangle$ and there is already an EPL-distilled state in one of the memories. The new EPL-distilled state can then be used to pump up the fidelity of this stored state using DEJMPS.

This definition of a round makes sense when generation of entanglement takes a lot longer than performing distillation. This may be the case when the two nodes performing EPL are far apart, resulting in slow entanglement generation compared to local operations and classical communication.

The first comparison that can be made is about the fidelity of the distilled state, which is done in Figure 2.4. The depolarising parameter in equation (1.7) for doing a distillation step is 0.99. Decoherence in the quantum memories happens with a dephasing parameter in equation (1.10) of 0.99 per round. The initial state is dephased with $p_d = 0.95$, as in equation (2.2). In the case of getting a success in the first round of EPL-distillation, the fidelity will always be highest. In that case the state has not been in memory and only minimal operations have been done. In the case of getting a $|00\rangle$ outcome and eventually distilling it to get a success, the distillation steps have been done twice and one of the states has also been in memory for a while.

It is clear that for these parameters storing the state ρ_{00}^2 from equation (2.21) would not be useful. This would introduce a lot more noise and the fidelity would be below a usable level. Therefore we will only look at storing the state ρ_{00}^1 in the remainder of this section.

Now let's look at entanglement pumping, which is the protocol where we keep using the EPL-distilled states to get a higher fidelity. The goal of repeating DEJPMS-distillation is to end up with a higher fidelity than 0.9, which is

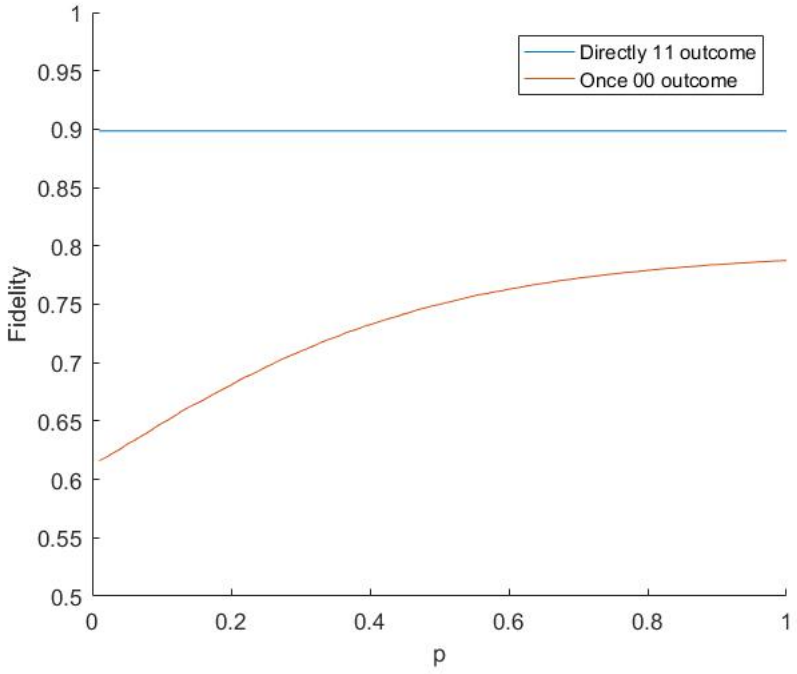


Figure 2.4: Fidelity of the state after performing distillation as a function of the initial p parameter. The depolarising parameter in equation (1.7) for doing a distillation step is 0.99. Decoherence of the quantum memories happens with a dephasing parameter in equation (1.10) of 0.99 per round. The initial state is dephased with $p_d = 0.95$, as in equation (2.2). The blue line means immediately getting a success, whereas the orange line means getting a $|00\rangle$ outcome and successfully performing distillation of two of those states. The orange line increases as a function of p due to having to store states only for a shorter time on average, before getting the second $|00\rangle$ outcome. The blue line is constant, because no storage is necessary and p only effects the success rate, but not the resulting fidelity.

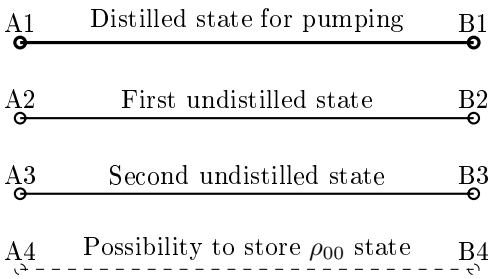


Figure 2.5: Overview of the quantum memories for A and B, numbered by $A1, .., A4$ and $B1, .., B4$. The minimum amount of memories required is three, to store an EPL-distilled state and to be able to generate two new states. The idea is to continuously generate entanglement between $A2$ and $B2$ as well as between $A3$ and $B3$. When both of these links are established, EPL-distillation is performed between them. Successfully EPL-distilled states can then be stored in memories $A1$ and $B1$, or if there is already an EPL-distilled state there, DEJMPS-distillation is performed again between them and the resulting state stored in $A1$ and $B1$. If $A4$ and $B4$ are also available, the ρ_{00} state can also be stored and used for EPL-distillation. Whenever an EPL-distillation between a new ρ_{00} state and one stored between $A4$ and $B4$ succeeds, it can be used to DEJMPS-distil with the state stored between $A1$ and $B1$.

what we end up with after doing just EPL-distillation on the initial states in Figure 2.4. We can compare the situations of having three or four memories available. Three memories indicates that we can store the resulting state after measuring $|11\rangle$ in the EPL scheme and try to pump that state using the same post-measurement states. With four memories we can also store the state after measuring $|00\rangle$ in the EPL scheme and use two of these states to decrease the number of rounds required. A schematic overview of the available memories is given in Figure 2.5 We can now analyse if having this additional memory actually boosts the rate, the result is displayed in Figure 2.6.

In general doing DEJMPS-distillation is not always better than throwing away a state. Further investigation could be done to see in which cases keeping the $|00\rangle$ outcome states is better than simply throwing them away. In the previous simulations we only threw states away from the memories if the fidelity became lower than 0.5. This threshold should in general depend on the target fidelity of pumping and can be optimized to let distillation always have a positive effect. This would prevent a scenario like in Figure 2.6 where having more memories results in having to do more rounds.

Summarizing the results of this chapter, we looked at various rates and resulting fidelities for the EPL entanglement generation protocol. Using not just the success case, where both parties measure $|1\rangle$, but also the case where both

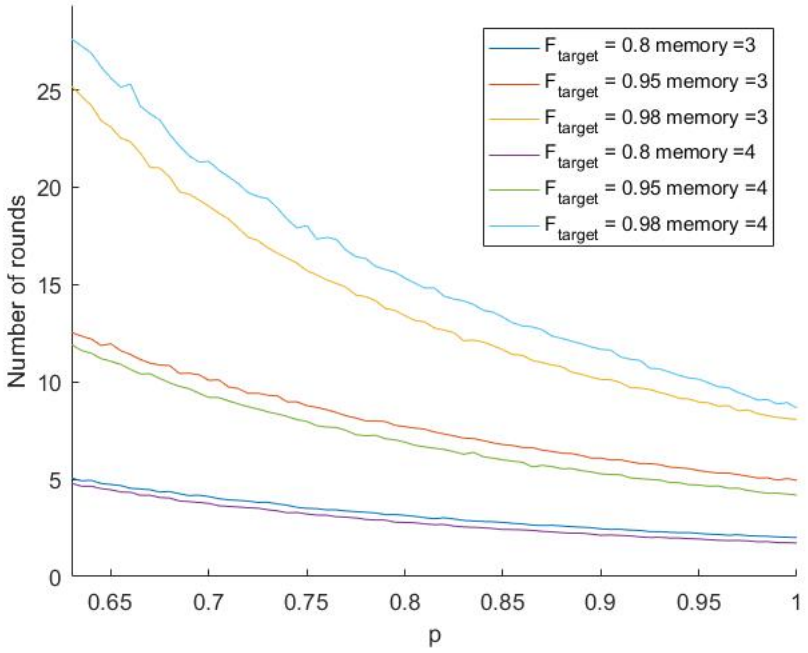


Figure 2.6: Number of rounds required to make a state with different target fidelities, using three or four memories. A round indicates the use of the EPL scheme on a new pair of states, as defined before. The DEJMPS protocol is used to pump the entanglement to the required fidelity. The same noise parameters are used as in Figure 2.4. Note that keeping the ρ_{00} states actually increases the number of rounds that is necessary, when trying to pump the fidelity to 0.98. This is because the states resulting from that EPL-distillation have lower fidelity, which together with the noise have a negative effect on the pumped state. The distillation of the pumped state and a state of lower fidelity does not increase the fidelity of the pumped state as much and also has a chance of failure, losing the pumped state completely. For the lower target fidelities it is beneficial to keep the ρ_{00} states as it gives a slight decrease in number of rounds.

parties measure $|0\rangle$, the rates can be boosted. This requires extra quantum memories and gives a rise in the rate, mainly for high values of p in the EPL starting state (2.41). The analysis of entanglement swaps and distillation resulted in the conclusion that doing distillation first has a higher success probability. In the next chapter there will be some more discussion about this protocol, comparing the distillation first and swap first protocols, also in the case of noise. This will be done using the discrete-event simulator QNetSquid.

In the case of noisy operations and storage, the usefulness of the ρ_{00} states decreases. This is due to the extra time the resulting states have to be in storage, before they can be distilled and the higher number of operations that have to be done to end up with a distilled state. As a result of this, in a realistic scenario it seems only useful to store the state with twice the phase, which is the state ρ_{00} in (2.12). The noise parameters that were taken to show the increase in rates by using the ρ_{00} states, have not been demonstrated in experiments. This means that before it could be useful to actually run this improved scheme, the noise in physical implementations will have to decrease. In the noisy scenario an entanglement pumping protocol has been analysed, in which you keep generating new EPL-states and distilling them. At some point the decoherence in quantum memories and noisy operations in the distillation process balance the increase in fidelity due to distillation, which means that there is a limit to which the fidelity can be pumped.

Chapter 3

Applications of QNetSquid

3.1 Introduction

In this chapter we look at applications of QNetSquid (Quantum Network Simulator for Quantum Information using Discrete-events), which is a Python package currently under development by QuTech. First we will shortly discuss what a discrete event simulator is and how it can be used for quantum information simulations. Then in the next sections some concrete applications that I have worked on for my project will be discussed. We model well known entanglement generation protocols using previously established distributions [17] in section 3.2 and using a full simulation in QNetSquid starting from section 3.4. These entanglement generation protocols are then used to obtain rates and resulting fidelities for specific noise models. We extend the analysis of doing distillation first or entanglement swap first in the EPL protocol in section section 3.3, which was first analysed in section 2.3.

The main idea of a discrete event simulation is that there is a timeline of events that happen within the simulation. These events happen at specific points in time and the assumption is that there are no changes in the system between these events. Every event has the ability to schedule future events at any future point in the timeline. The simulation ends when there are no further events scheduled.

As a small example we can take a table tennis game, which is also the example in QNetSquid for introducing the discrete event simulator. We now want to simulate this game between two parties, Ping and Pong. To start the game one of the parties has to serve, so we schedule an event in which Ping serves at time $t = 0$. Now Ping makes the serve and schedules an event at $t = 10$ that Pong will receive the ball. Pong is listening to this specific event and reacts exactly at $t = 10$ by hitting the ball back. Then Pong schedules an event at $t = 20$, at which it is Ping's turn to hit the ball again. The game could continue

like this forever if every time they hit the ball they schedule a new event. This means that we could either force the conclusion of the simulation at a fixed time or add a probability that the players will not return the ball. In that case the simulation will end after one of the players will not schedule a new event, which would be after some probabilistic amount of time.

In general the units of time of the simulation are not fixed, which means it could be seconds, hours, days, etc. We are not limited to integer points in time, but we can also schedule events at $t = 0.1$ or $t = 0.0001$. However, it is important that the units are consistent within the simulation, for example to avoid one component scheduling a delay in seconds and the other one in microseconds. There could also be multiple events scheduled at any point in time, in which case they would be executed in a random order. To formalise the example, we can say that Ping and Pong are two Entities. Entities are able to schedule events on the timeline and listen to specific events on the timeline. For example when there are two table tennis games going on at the same time, we do not want players to react to the wrong game.

In a lot of cases the events may be scheduled at probabilistic times, e.g. the time it takes to generate entanglement. When there is a lot of those probabilistic events that have to react to each other, it is very hard to obtain analytic results. This is the strength of the discrete event simulator; we can just run the whole simulation a few times and obtain information about the average time for the protocol to complete.

Now we can have a look at the quantum side of the simulator. Using the discrete event simulator we can simulate all kinds of quantum protocols. One example of a protocol is the entanglement generation protocol EntangleNodes. To understand this protocol we must first have a look at physical components. The first relevant component is the QSource, which is able to generate qubits with specified fidelity locally. After some delay in which the source prepares the qubits, the qubits can be retrieved from the source and used in a protocol. The second component is a QMemory, where we can put qubits and after some time get them from the memory again. During this time decoherence happens based on the specified noise parameter of the memory. The third component is a QChannel, which can be used to transmit qubits with noise specified in the noise model and a delay indicating the time it takes for qubits to travel over the fibre. The EntangleNodes protocol uses these components to generate entanglement between two nodes A and B, which is displayed schematically in Figure 3.1.

The goal of QNetSquid itself is to support a wide variety of protocols, for which it should stay as general as possible. For snippets we model a specific component or protocol to see how QNetSquid performs and to obtain new results. One example is the quantum four-node network that is currently under development in the Netherlands. In this network there would be four main nodes and likely intermediate stations which can be used as quantum repeaters. QNet-

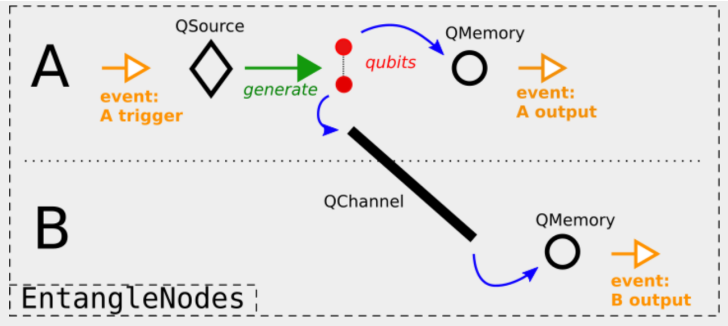


Figure 3.1: The EntangleNodes protocol starts to generate entanglement between A and B after being triggered by an external protocol. First the QSource generates a pair of maximally entangled qubits locally in A, after which one of the qubits is put on the QMemory of A. The other qubit is put on the QChannel towards B. Once the qubit reaches the other side of the fibre, B will pick it up and put it in its memory. When node A puts the qubit in his memory, the output event signalling that the qubit of A is ready will be given and when the photon reaches B the same will happen for B. Image courtesy: Rob Knegjens.

Squid can be used to simulate this network in as much detail as required. Every snippet has a very specific functionality and could be part of a bigger protocol. One snippet could be the QSource that generates qubits locally, then a snippet that simulates the optical system and fibres, a snippet for the detector that is used at the middle station, etc. It should be modular, such that the QSource could easily be replaced by a different QSource that has other properties, but the same in- and outputs. Then there is a snippet for the overlying entanglement generation protocol that uses all these snippets to simulate the generation of entanglement. Finally, there could be a snippet that uses this entanglement to run quantum key distribution, or some other application using entanglement.

During my project I have also written some snippets that are not mentioned in the rest of this thesis, a few of those are briefly discussed in Appendix B.

3.2 QSourceMid for abstract entanglement generation

One example of a snippet is the QSourceMid, that refers to the use of a mid-point to generate entanglement. In general the QSourceMid takes a timing distribution function, instead of the fixed preparation delay of the standard QSource. Next to this you can specify noise operations that should be applied to a Bell-state. This noise model can consist of any operation on the qubits. This is a very general model of a source that outputs entangled qubit pairs. In this section we will look at a few concrete applications in entanglement generation protocols.

The QSourceMid can be used to model some of the most used entanglement generation protocols like the Barrett and Kok [21], single click [22, 23] and EPL [24, 12] protocols. The general idea is that two nodes A and B both create entanglement between their stationary memory and a photon locally. Both A and B then put the photon on a fibre towards a mid-point C where both photons are detected. This detection event, referred to as a click, is essentially a Bell-state measurement and provides entanglement between A and B. For the Barrett and Kok, which is also called the double click protocol, both photons need to reach the detector in the same round. For the single click and EPL protocol only one photon should be detected at one time. Then if the channels from A to C and B to C have transmissivity η, for the single click and EPL protocol the probability of successful entanglement generation is proportional to η. The probability of success for the Barrett and Kok protocol, is proportional to η^2. The drawback of the single click protocol is that it is more sensitive to photon loss in the fibre and phase drift in the fibre. This can be solved by performing distillation, using two pairs generated by the single click protocol. Doing single click twice and then distillation is called the EPL protocol. The single click protocol can therefore only work by itself if we can stabilise the phase in the fibre.

The time it takes to generate entanglement for the different entanglement generation protocols is calculated in [17], for NV-centres. This time includes the entire entanglement generation process, starting from photon-memory entanglement to communication back from the mid-point signalling success. A simple way to model this in QNetSquid is to take the underlying probability distribution and use it effectively to model both the QChannel and the QSource in the EntangleNodes protocol. This means that after a random time sampled from the distribution we directly put the qubits on the QMemory of both A and B. They then both output the ready event of the entanglement generation protocol at the same time; this protocol in QNetSquid is called MidEntangleNodes.

The QSourceBK is an instance of this QSourceMid and is specific for the Barrett and Kok protocol. It takes as input the distance between A and B and some hardware properties like the NV-outcoupling efficiency, the frequency conversion frequency and the expected fidelity of the output state. It then constructs the time distribution and noise operations for this protocol. This QSourceBK can then directly be used in the MidEntangleNodes protocol.

The QSourceSC for the single click protocol is similar to the QSourceBK, but uses slightly different hardware parameters and generates a different timing distribution and noise operation model. The most important source of noise is the emission of a photon by both A and B, but due to photon loss only one of them getting to the detector. The detector then registers a single click and A and B think they have entanglement, but in fact they just have a classical state. This is also the process leading to the $|11\rangle$ component in the EPL protocol, as

discussed in the previous chapter. This noise can be reduced by emitting less photons from the NV-centres. The probability of two photons being released in the same round then becomes smaller. Both A and B can generate entanglement between the electron spin and a photon locally:

$$|NV, photon\rangle = \sin(\theta)|\downarrow 0\rangle + \cos(\theta)|\uparrow 1\rangle. \tag{3.1}$$

$|\downarrow\rangle$ and $|\uparrow\rangle$ refer to the dark and bright state of the electron spin, whereas $|0\rangle$ and $|1\rangle$ to the absence and presence of a photon. The angle θ is a parameter that can be chosen. In the limit of low transmissivity $\eta \to 0$, this leads to the entanglement between A and B of the form:

$$\rho_{NV,NV} = \sin^2(\theta)|\psi^+\rangle\langle\psi^+| + \cos^2(\theta)|\uparrow\uparrow\rangle\langle\uparrow\uparrow|. \tag{3.2}$$

In the section about the QDetector we will look more closely at the modelling of the photon detector in the middle. For more details about the experimental realisation of this process see [22]. Practically, this means a trade-off between fidelity of the entanglement and the rate at which it can be achieved. If θ is $\pi/2$, the output state is a perfect Bell-state, but no photons are released from the NV-centre. This means that the rate at which these states can be produced is 0.

For the QSourceEPL, obtaining the EPL distribution is a bit more involved. First we generate the single click timing distribution. We then take two samples from this distribution and try to perform distillation on these two. If distillation succeeds, the protocol is finished and we have our sample. If distillation fails two new timing samples are taken from the single click distribution, this keeps going until successful distillation. There is also the option to either generate the two entangled pairs for distillation in parallel or in sequence, which depends on the hardware.

The goal of these instances of QSourceMid is to run an entanglement generation protocol without expensive calculations. There is a lot of errors and noise sources that are not taken into account. In a separate project, that will be discussed starting from section 3.4, we try to model all the components using QNetSquid. This includes an accurate model of the QChannels, a QDetector in the middle and a model of the QSources that generate local entanglement.

3.3 EPL

In this section we compare the performance of swapping first and distilling first from section 2.3 using QNetSquid. The idea is to model all the events happening and run a discrete event simulation. After that we can look at some results of the simulation.

Both the distillation first protocol $EPL_{dist,swap}$ and the swap first protocol $EPL_{swap,dist}$ protocols are initialised by calling the class using a few parame-

ters. These include the communication time between nodes, the length of the quantum fibres between nodes, the time it takes for the QSource to generate entangled qubit pairs, the fidelity of these qubits and the decoherence rate of the quantum memories. These are then used to initialise the EntangleNodes protocol on four links in parallel, two between A and R and two between B and R.

For $EPL_{dist,swap}$ the ready events of the EntangleNodes protocol are linked to the distillation step. Once the two entangled pairs are ready on a link, these pairs are distilled to remove the phase and obtain better entanglement. The output of the distillation step is then linked to the swap protocol. This checks the outcome of the distillation step; if it failed we start the EntangleNodes protocol again and keep trying until both distillations have succeeded. After distillation has succeeded on both links, the entanglement swap is executed. Then finally a correction is made for the outcome of the Bell-state measurement and we look at the resulting state. If the final state between A and B is the Bell-state we expected, we call it a success and otherwise a failure. We also record the total time it took to run this round.

For $EPL_{swap,dist}$ the ready events of the EntangleNodes protocol are linked to the swap step. Whenever one of the four EntangleNodes protocols gives an output event, we check whether there is an entangled pair ready on the other link. If this is the case then the entanglement swap between these pairs is executed. If the swap is successful, we apply the correction for the swap and wait for the other swap to succeed to perform distillation. Finally we check the resulting state again and compare it to the Bell-state we expect.

In the case of no noise in the system, this means perfect memories and no loss in the fibres, we will always be able to predict the resulting Bell-state. We can then run the protocol a number of times to obtain the average simulation time. This can then be compared to the result of section 2.3, where the probability of success of the distillation first protocol was twice that of the swap first protocol. For both protocols entanglement pops into the system on all four links every time unit using the EntangleNodes protocol. After that either distillation or the entanglement swap can be done first. The nodes can communicate with each other instantaneously throughout the protocol, similar to how in section 2.3 classical communication between nodes was completely disregarded.

As we can see in Figure 3.2 the $EPL_{dist,swap}$ protocol completes roughly twice as fast. In this figure, there is no noise included, which means that the output pairs are perfect maximally entangled states. These results are consistent with lemma 5 from section 2.3.

The rest of this section is about the realistic entanglement generation protocol. The single click entanglement generation can be used to generate the four elementary links. We assume that the phase can not be stabilised, which

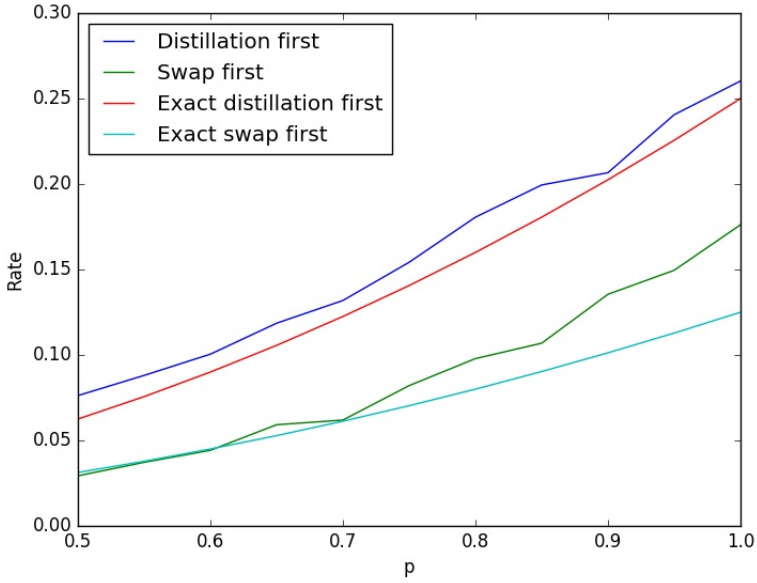


Figure 3.2: Comparison between $EPL_{dist,swap}$ and $EPL_{swap,dist}$ protocols; the rate of entangled states per number of input states for different values of p in the EPL-mixed state from equation (2.13). For both simulations entanglement pops into the system on all four links every time unit, after which either distillation or the entanglement swap can be done first. For the analytical solutions, resulting from lemma 5 in section 2.3, everything has to succeed in one time step. In the simulation it can be spread out over multiple time steps, giving a slightly higher rate. For example, in the case of doing distillation first, in the simulation the distillation can succeed independently on both links in different rounds, after which the entanglement swap can be made. In the analytical solution distillation has to succeed on both links in the same round. It can be seen that doing distillation first is also optimal in this model.

means that distillation is necessary. We also introduce dephasing over time to the quantum memories, which will mean that the output states are no longer maximally entangled. This is the only source of noise; operations on qubits are still assumed to be perfect. The distillation first and swap first protocols can then be run for different angles of θ, as discussed in the previous section. The goal is to find the optimal value of θ and compare the difference in rate and fidelity between the two protocols.

Comparing Figure 3.3 and Figure 3.4 it seems that the value of θ that optimises the rate is also close to optimal for the output fidelity. This can be explained by looking at the time the states have to be in memory. If the rate is low, there is a long waiting time before generating all four entangled links. Then states will be in memory for a while and therefore dephase more. This can also be seen in the resulting fidelity, which drops drastically due to dephasing as θ goes to $\pi/2$. In the limit of small θ the single click entanglement generation rate is high, but the resulting state has a low p parameter. This means that distillation succeeds rarely and the total rate will be low.

To find the optimal value of θ more precisely and find out whether there is a difference in fidelity of output states between the two protocols, a higher average number has to be taken. It would also be interesting to include more sources of noise, not just dephasing to the memories. The operations and measurements are in general noisy too and this could shift the distributions and reduce the output fidelity.

3.4 QDetector and multi-qubit measurements

Photon detectors have been used in a lot of entanglement generation protocols to measure photons that are entangled to a stationary qubit at a node. This concept has been discussed before, for example in the section about the QSourceMid. In that section QNetSquid was used for only a small part of the entanglement generation protocol, in this section we want a more detailed model. The most important missing component in QNetSquid to simulate this is a detector component, which is used in all entanglement generation protocols discussed before. A schematic overview of the system that the QDetector models can be found in Figure 3.5.

The idea of the QDetector is to measure photons that have just arrived over a quantum channel. There are different encodings that can be used to use a photon as a qubit. One of the most common ones is polarization encoding, which means that a detector should be able to distinguish different polarization states of the photon. Another one is presence-absence encoding, which means that the qubit is encoded in the presence or absence of a photon. In that case, presence could be the $|1\rangle$ state and absence the $|0\rangle$ state, forming a qubit. The last one that should be mentioned is time-bin encoding, which means that the

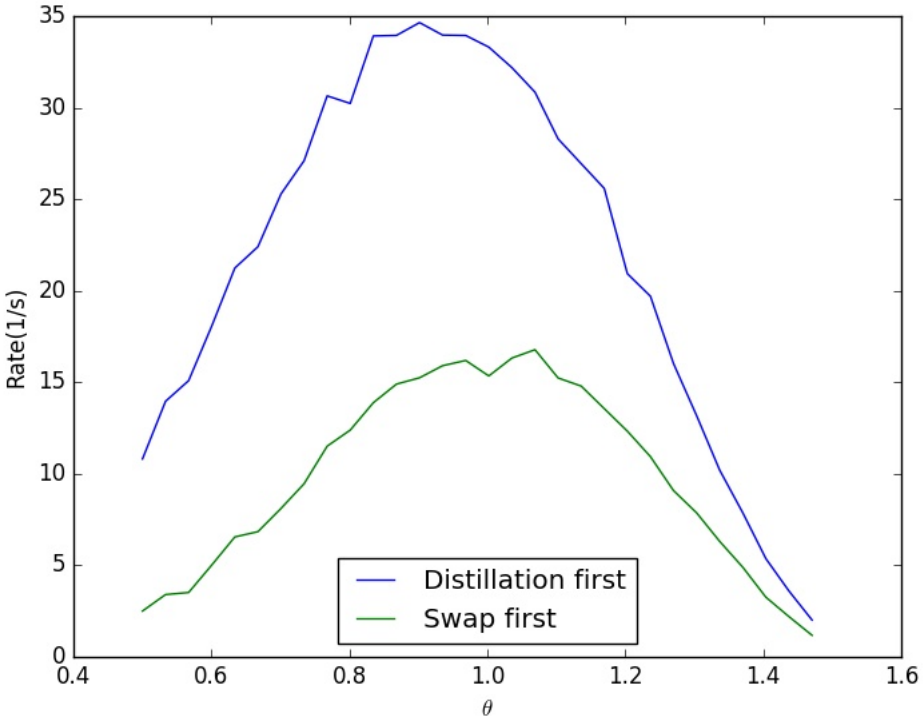


Figure 3.3: Comparison between $EPL_{dist,swap}$ and $EPL_{swap,dist}$ protocols; the rate of generated entangled pairs per second as a function of the angle θ giving the bright state population of the NV-centre. Similar to the previous cases, the distillation first protocol has a roughly two times higher rate. The optimal angle of θ is around 0.9, which means that the bright state is slightly less probable than the dark state. The generation time is averaged over 2000 generated entangled pairs.

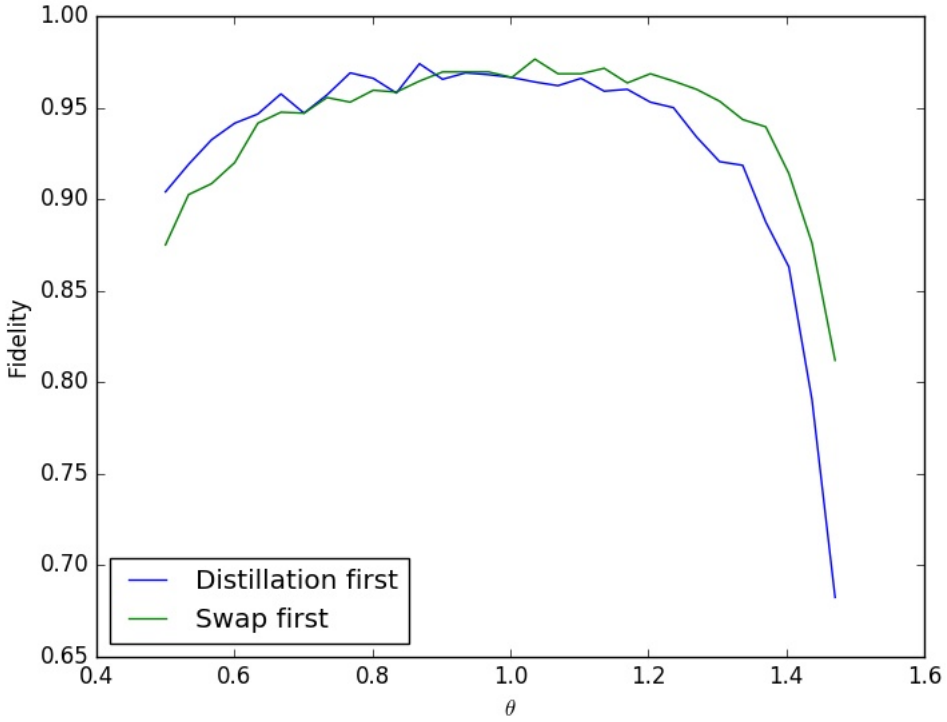


Figure 3.4: Comparison between $EPL_{dist,swap}$ and $EPL_{swap,dist}$ protocols; fidelity to the maximally entangled state of the generated entangled pairs. The fidelity in the ideal case would be 1; the difference is due to dephasing of the memory qubits. It is hard to see the difference in fidelity between the two protocols, but the highest fidelity is achieved for values of θ where the rate is also high. In the limit of θ close to $\pi/2$ the fidelity drops due to dephasing in the quantum memories. In this limit the generation rate of the single click protocol is low, which means that we have to store states for a long time while trying to generate the other states.

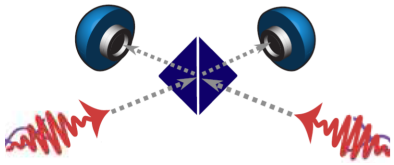


Figure 3.5: Schematic overview of the photon detection system for performing a Bell-state measurement on single photons. The photons enter the beam splitter from different sides. If the photons arrive at the same time, they interfere and the information from which side the photon came is erased. The photons are then measured by the detectors. This whole process is modelled by the QDetector. Image courtesy: the authors of [10].

photon can either reach the detector in the early time slot or in the late time slot. As the wave function of a photon is spread out in space and so also in time, we can have a superposition of early and late photons, which is also a qubit.

The functionality and measurement operators of the QDetector can be different based on the encoding of the photons that it should detect. For the QDetector that we developed, we assume presence/absence encoding, which will be used for the rest of this chapter. Now this QDetector can be used to perform a Bell-state measurement on two photons reaching it from different channels. If node A and B are generating entanglement, they coordinate to let the photons that are entangled to their stationary qubit reach the detector at the same time. If this is done in the right way, the information whether a photon came from A or from B is erased in the detector.

Now it is useful to have a closer look at the photon detectors. In our system, we assume that a maximum of one photon can be emitted from each NV-centre. The maximum number of photons that can reach the middle station is two, one coming from the right and one from the left. This puts a restriction on the detection events that can happen at the detector. We can look at the three cases, either the left detector clicks, or the right detector clicks or neither of them. In terms of POVM-elements, as defined in section 1.4:

$$
\begin{aligned}
P_L &= |2,0\rangle\langle 2,0|_{out} + |1,0\rangle\langle 1,0|_{out}, \\
P_R &= |0,2\rangle\langle 0,2|_{out} + |0,1\rangle\langle 0,1|_{out}, \\
P_0 &= |0,0\rangle\langle 0,0|_{out}.
\end{aligned}
$$

The first projector P_L happens if one photon reaches the left detector, or two photons reach the left detector and the same for P_R. Notice that it is not possible to have both the left and the right detector click at the same time, as will be shown later. Now these projectors can be inverted to see which input

states before the beam splitter result in these detection events. For this we need some quantum optics, in terms of creation and annihilation operators.

$$\hat{c}_L = \frac{1}{\sqrt{2}}(\hat{a} + \hat{b}) \qquad \hat{c}_R = \frac{1}{\sqrt{2}}(\hat{a} - \hat{b}) \tag{3.3}$$

Where $\hat{c}_L$ and $\hat{c}_R$ are the operators on the output of the beam splitter and $\hat{a}$ and $\hat{b}$ are the operators on the input of the beam splitter. This can be used to rewrite the operators P_L,P_R and P_0 in terms of input states of the beam splitter.

First calculate the input state for two photons reaching the left detector:

$$\begin{aligned}
|2,0\rangle_{out} &= \frac{1}{\sqrt{2}}\hat{c}_L^\dagger \hat{c}_L^\dagger |0,0\rangle_{out} \\
&= \frac{1}{2\sqrt{2}}(\hat{a}^\dagger + \hat{b}^\dagger)(\hat{a}^\dagger + \hat{b}^\dagger)|0,0\rangle_{in} \\
&= \frac{1}{2}(|2,0\rangle_{in} + |0,2\rangle_{in}) + \frac{1}{\sqrt{2}}|1,1\rangle_{in}
\end{aligned}$$

In the case of one photon reaching the left detector:

$$\begin{aligned}
|1,0\rangle_{out} &= \hat{c}_L^\dagger |0,0\rangle_{out} \\
&= \frac{1}{\sqrt{2}}(\hat{a}^\dagger + \hat{b}^\dagger)|0,0\rangle_{in} \\
&= \frac{1}{\sqrt{2}}(|1,0\rangle_{in} + |0,1\rangle_{in})
\end{aligned}$$

Similarly in the case of two photons reaching the right detector:

$$\begin{aligned}
|0,2\rangle_{out} &= \frac{1}{\sqrt{2}}\hat{c}_R^\dagger \hat{c}_R^\dagger |0,0\rangle_{out} \\
&= \frac{1}{2\sqrt{2}}(\hat{a}^\dagger - \hat{b}^\dagger)(\hat{a}^\dagger - \hat{b}^\dagger)|0,0\rangle_{in} \\
&= \frac{1}{2}(|0,2\rangle_{in} + |2,0\rangle_{in}) - \frac{1}{\sqrt{2}}|1,1\rangle_{in}
\end{aligned}$$

finally for one photon reaching the right detector:

$$\begin{aligned}
|0,1\rangle_{out} &= \hat{c}_R^\dagger |0,0\rangle_{out} \\
&= \frac{1}{\sqrt{2}}(\hat{a}^\dagger - \hat{b}^\dagger)|0,0\rangle_{in} \\
&= \frac{1}{\sqrt{2}}(|0,1\rangle_{in} - |1,0\rangle_{in})
\end{aligned}$$

Now it is possible to transform the POVM-elements P_L, P_R and P_0 from the output basis to the input basis. This results in the new POVM-elements N_L, N_R and N_0. Note that from our set-up it is not possible for two photons to reach the beam splitter from the left or from the right. This is because both nodes release a maximum of one photon each, so we can exclude the states $|2,0\rangle_{in}$ and $|0,2\rangle_{in}$.

$$\begin{aligned} N_L &= |\psi^+\rangle\langle\psi^+|_{in} + \frac{1}{2}|1,1\rangle\langle 1,1|_{in} \\ N_R &= |\psi^-\rangle\langle\psi^-|_{in} + \frac{1}{2}|1,1\rangle\langle 1,1|_{in} \\ N_0 &= |0,0\rangle\langle 0,0|_{in} \end{aligned}$$

Note that P_0 and N_0 are trivially the same. The sum of the POVM-elements $N_L + N_R + N_0 = I$, as required, on the reduced space. This measurement is not projective, because the element $|1,1\rangle_{in}$ occurs in two measurement operators. In this case we do not have to care about Kraus operators, because the post measurement state of the photons is not important. The photons are detected and measured away, the important part is the resulting states on the stationary qubits of A and B.

It is now also easy to verify that the state $|1,1\rangle_{out}$ can not occur, because it leads to an input state outside our basis.

$$\begin{aligned} |1,1\rangle_{out} &= \frac{1}{\sqrt{2}}\hat{c}_L^\dagger \hat{c}_R^\dagger |0,0\rangle_{out} \\ &= \frac{1}{2\sqrt{2}}(\hat{a}^\dagger + \hat{b}^\dagger)(\hat{a}^\dagger - \hat{b}^\dagger)|0,0\rangle_{in} \\ &= \frac{1}{2}(|0,2\rangle_{in} + |2,0\rangle_{in}) \end{aligned}$$

At the time the QDetector was being developed, QNetSquid did not have a function to do general measurements on multiple qubits. This is something that I implemented before continuing with the detector.

The detector component, as implemented in QNetSquid, can be broken down in a few different steps. The first step is to open the time window, which means that the detector will start detecting photons. In discrete-event simulator terms this means that photons, represented by a qubit, can be put on the detector. The detector then waits till the end of the time window and only then carries out the measurement with POVM-elements N_L, N_R and N_0. The output of the detector component will then be a number $0, 1$ or 2 according to which of the detection events happened. Only the measurement outcomes N_L and N_R lead to the desired entangled state between A and B. When we measure N_0 the resulting state will not be entangled and A and B have to throw away their

qubits and restart the generation process.

There is an important source of noise in photon detectors called dark counts. These dark counts are unwanted photons that the detector picks up from the environment. This may trigger the detector, even though no photons from A or B reached the detector. We can assume that these photons are uncorrelated from the rest of our set-up, which means that the analysis of dark counts can be done purely classically. For a simple model of these dark counts, we can look what happens for the different detection events under the influence of dark counts. The probability of a dark count p_d is assumed the same for the right and the left detector. The probability p_d can be calculated using the length of the time window t_w and the dark count rate R_{dc}.

$$p_d = 1 - \exp(-t_w \cdot R_{dc}) \tag{3.4}$$

The first scenario is that there are no photons from A or B, but there is a dark count on one of the detectors. This happens with probability $2p_0p_d(1-p_d)$ and lets us accept a classical state as the output. This process will reduce the fidelity of the output state.

The second scenario is that the left (right) detector clicks, but there is also a dark count in the right (left) detector, which happens with probability p_Lp_d (p_Rp_d). In that case we notice that both detectors triggered, but this is only possible in case of a dark count, so we have to reject an entangled state. This process leads to the reduction of the rate at which entangled states can be generated.

The dark count rate is an experimental parameter and the lower it is, the better the photon detector. In the loophole-free Bell-test [4], a photon detector with a dark count rate of $10s^{-1}$ was used. In the next section the effect of these dark counts on the output fidelity will be examined.

3.5 Entanglement generation protocol using QDetector

The detector component can now be used to make a full entanglement generation protocol using NV-centres between nodes A and B using the midpoint photon detector M. For this a few physical components are necessary:

- QSource that can generate a local entangled state between electron spin and photon. The time at which the photon is released from the NV-centre (QSource) is important because it will determine whether the photon reached the detector inside the time window.
- QFibre that can be used to let the photons travel from A to M and from B to M. In the case of loss of number encoded photons the qubits should

not be thrown away but amplitude damped. This is because photon loss should not affect the absence of a photon, but does impact the presence of a photon. This is the amplitude damping channel as discussed in section 1.4.

- QDetector that can detect the photons at the midpoint source, functionality as discussed in the previous section.
- CFibres from M to A and from M to B to communicate the classical output of the detector back to the nodes A and B.

Finally the timings of the entanglement generation should be coordinated, such that both photons arrive at the detector at the same time and the detection window is actually open at this point in time. Therefore when the entanglement generation protocol is triggered by some external protocol, the first step to take is to coordinate the timings. For example when A is closer to the midpoint than B, B should start its QSource before A does, because the QFibre from B to M takes longer than the QFibre from A to M.

One additional issue is the arrival of a photon outside the time window at the detector, in the case of presence/absence encoding. If we want to carry out the detector measurement defined in the previous section, we actually need two photonic qubits at the detector to measure. If the photon gets lost on the fibre, or arrives outside the time window, this means that we should have the absence of a photon within the time window. If the delay times of both the QSource and the QFibre are deterministic, this gives no problems, because we can always make sure that the photon arrives within the time window by coordinating the timings. However, in an experiment the QSource is not deterministic as the emission of the photon is caused by a decay process. The decay time can then determine whether or not the photon arrives within the time window of the detector.

In the discrete event simulation this can be fixed in a few different ways. The main point is to apply amplitude damping to the photon at some point within the simulation to map it from the presence state to the absence state, if it would arrive outside of the time window. The QSource would then emit photons at a deterministic time, but additional amplitude damping with damping parameter $1-\eta$ would be applied. Here η is the probability that the photon arrives within the time window.

3.6 Comparison of entanglement generation protocols

The difference between the two previously discussed models of entanglement generation can now be examined. First we will give an overview of the physical

Table 3.1: Summary of physical processes included in both simulations

Process	MidEntangleNodes	EntangleNodesDetect
Bright state population of NV-centre	Yes	Yes
Accurate spin-photon entanglement	No	Yes
Frequency conversion for transmission over fibre	Yes	Yes
NV-outcouping efficiency	Yes	Yes
Photon loss in fibre	Yes	Yes
Multiplexing	Yes	No
Dark counts at the detector	No	Yes

processes that both of them take into account. The MidEntangleNodes protocol, that samples from a distribution does not use the full functionality of the discrete event simulator. It uses a previously established result, which can now be compared to a full simulation of all processes in QNetSquid, which is called EntangleNodesDetect.

A lot of the physical processes happening in NV-centres are included in both simulations. The advantages of using MidEntangleNodes is that it can give a quick estimate of the time it takes to generate entanglement. The simulation is very fast, because most of the process is included in the distribution and we just have to take a sample. The drawback is that it does not really give an estimate of the output fidelity and that it is hard to include more general noise/loss models.

The advantage of the EntangleNodesDetect protocol is that the output fidelity is realistic, because all physical processes are actually simulated. The component-wise implementation makes it very easy to include more general noise/loss models. It makes it also easy to use a different fibre or a different detector but keep the rest of the entanglement generation protocol the same. The disadvantage is that the simulation takes a lot longer. For realistic parameters the photon number is very often zero at the detector, resulting in a failure case. To just generate one entangled pair may take a lot of time.

For the next simulations the following experimental parameters are used:

- Probability of frequency conversion of photons to telecom frequency $p_{fc} = 0.3$
- NV-outcoupling efficiency including emission into zero phonon line $p_{out} = 0.3$.
- The angle determining the bright state population of the NV-centre $\theta = 2\pi/5$.
- Photon loss on the fibre $0.2dB/km$.
- Initialising time of NV-centre before every run $\tau_i = 6.6\mu s$.

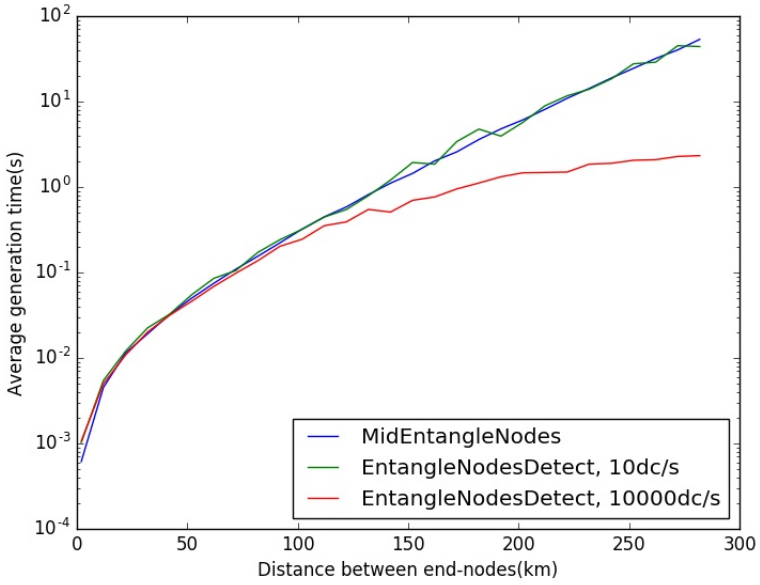


Figure 3.6: Average generation time as a function of the distance between end-nodes. The blue line is the MidEntangleNodes protocol without dark counts, whereas the green and the red lines are the EntangleNodesDetect protocol for different values of the dark count rate. It can be seen that at these distances 10 dark counts per second do not effect the generation time. Increasing the dark count rate lowers the generation time at high distances.

- Characteristic emission time of the NV-centre $\tau_{em} = 12ns$.
- Detector time window $t_w = 30ns$.

The dark counts have a higher impact for longer distances. This is because the photon loss in the fibre increases exponentially with distance, so the probability of a photon reaching the other side decreases rapidly. Therefore the rate at which dark counts occur relative to the probability of a photon from the end nodes being detected increases with distance. In Figure 3.6 and Figure 3.7 the effect of 10 dark counts per second is not visible, which means it has an negligible effect at distances of under 300 kilometers. At some distance the difference would become noticeable.

For a higher dark count rate the relative occurrence of dark counts is significant also for shorter distances. This reduces the average time it takes to generate entanglement, because the dark counts make us accept classical states. However, the fidelity of these states decreases to the point where there is no

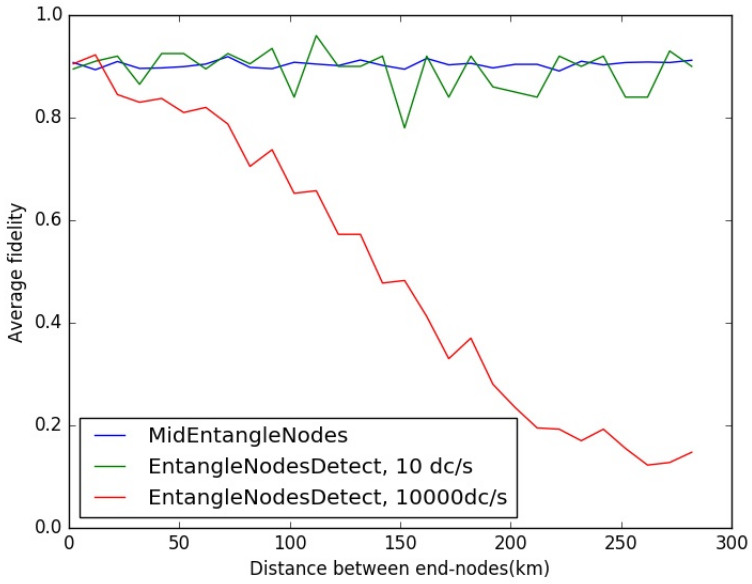


Figure 3.7: Average output fidelity to $|\psi^+\rangle$ as a function of the distance between end-nodes. The fidelity stays roughly constant as a function of distance, in the case of low dark counts. The lines are not smooth because the number of successful runs used to average is only 200 for the EntangleNodesDetect protocol.

useful entanglement left. This means that the dark count rate defines the maximum distance over which the entanglement generation protocol can be run.

Concluding this chapter, we have looked at the discrete-event simulator QNetSquid and applications on entanglement generation. The simulations about the EPL-protocol were extensions from the previous chapter. Now a realistic entanglement generation protocol was used, instead of assuming entanglement to pop into existence. This can be used to compare different entanglement generation protocols and see how they perform combined with distillation and entanglement swaps. The results are consistent with lemma 5 and doing distillation first is optimal in all analysed cases.

In the last sections a full simulation in QNetSquid was discussed. The first part of the simulation is modelling of the NV-centre with respect to emission of photons entangled to a memory qubit. The next steps are modelling of photon loss from the NV-centre to the detector and the modelling of the detector itself. The resulting rates and fidelities were very similar for the EntangleNodesDetect and MidEntangleNodes models, except for the regime of high dark counts. All of the physical components can be modelled in more detail in future projects, for example by including more sources of noise.

Another important point is the time it takes to run the simulations. In all the simulations of this chapter, only two or three nodes were involved. In a realistic quantum network this number may be a lot higher, making it even more important to simulate efficiently. The first step could then be to establish the distribution of entanglement generation time as a function of distance between nodes and other experimental parameters using QNetSquid. This distribution could then be used on all links of a large quantum network and entanglement generation times could be obtained by sampling from this distribution. This would use the power of the discrete-event simulator to both obtain distributions and sample from them, instead of using pre-established results.

Chapter 4

Entanglement tracking in a quantum network

4.1 Introduction and entanglement identifiers

The entanglement tracker is a protocol that runs locally on every node in a network and is in charge of keeping track of classical information about entanglement. This is useful in a larger quantum network, for example to complement an entanglement routing protocol. Every node will have an entanglement tracker that maintains a database of current entanglement between this node and any other node in the network. Entanglement IDs are the objects that are stored in the database and hold information about the entanglement between two nodes, including some estimate of the goodness of the entanglement. For example, whenever entanglement is generated, swapped, distilled or consumed, the entanglement tracker should be updated. Every node has an entanglement tracker and the entanglement trackers of different nodes can message each other with information to update their databases.

Define the network with nodes (repeater stations or end stations) and connections between neighbouring nodes (possible links to generate entanglement). Every node and connection has some quantum capabilities, which distinguishes it from the elements of a classical network. It is possible to generate elementary entanglement between neighbouring nodes, which may not be there at any time step. The entangled state could also already have been in storage for a while and so has decohered to a noisy state.

The entanglement tracker acts on two databases, one for current entanglement information and one for past entanglement information. When we say that the entanglement tracker deletes an identifier, this means that it will be moved to the past entanglement information. As a results, information about past entanglement is still accessible if it is needed to update other nodes. The

past entanglement information will be kept for some amount of time specified by the network, for example the maximum life-time of qubits/entangled pairs in the network. After this time a node can assume that it will not need this information any more. This prevents information from being kept indefinitely and overloading the memories of the node.

We now have a quick look at how entanglement identifiers are made for a general entanglement generation protocol. Each entanglement generation protocol has a source, which assigns a sequence number to the entanglement it creates. If A and B are trying to generate entanglement, this source can be at node A or at node B or at any place on the connection from A to B. Every time the source creates an entangled pair, it assigns a sequence number to the pair, which should be unique between A and B. The sequence number gets communicated to A and B and they both store this information locally. The sequence number combined with the node identifiers of A and B should be unique within the whole network and form the basis of the entanglement identifier. In general an entanglement identifier (EntID) holds the following information:

Own node ID	ID of node at which the entanglement identifier is stored
Other node ID	ID of node with which the entanglement is shared
Sequence number	Sequence number that is unique between the two nodes generating entanglement and globally unique combined with the two node IDs
Goodness	Heuristic estimate of the fidelity of the entangled pair
$t_{goodness}$	Time at which the goodness parameter was last calculated
corr	Correction information resulting from an entanglement swap that should still be used

The goodness parameter is stored to keep track of the fidelity of the entangled pair. If the state has decohered in storage for a while, or has undergone a lot of operations, the entanglement may have been lost. Note that the goodness parameter is based on the physical properties of the quantum hardware. The means of updating the goodness parameter may be highly hardware-dependent. The goodness parameter and the time of goodness may not hold too much information, without also knowing at least something about the physical set-up. This means that whenever the entanglement tracker is used, it may need to know something about the physical hardware in order to perform operations on these entanglement identifiers. This information about the quantum hardware may just be a T_1 and a T_2 time for the quantum memories. How to update goodness parameters will be discussed in more detail in section 4.4.

We now want to obtain entanglement between two nodes A and B, where A and B are any two nodes in the network. We can use entanglement swaps (Bell-state measurements) along some path from A to B in the network. For example: there exists a node R such that there are entangled pairs shared be-

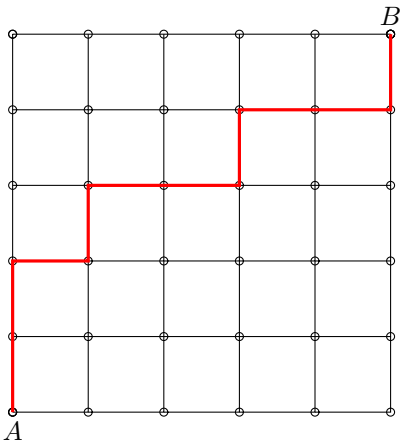


Figure 4.1: Possible structure of a quantum network, the nodes are evenly spaced and entanglement can be generated between neighboring nodes. To establish entanglement between A and B multiple entanglement swaps will have to be executed. It is possible to reserve the red path and only generate entanglement on these links, after which entanglement swaps can connect A and B. There are many other paths from A to B, which means that this is a routing problem. The routing algorithm may be in charge of determining which nodes generate entanglement and when they make entanglement swaps. The entanglement tracker assists the routing protocol by handling the classical communication between the nodes.

tween A and R and between B and R. In that case R could make a Bell-state measurement on both of its qubits and in that way entangle A and B. After that R would communicate to A and B that they are now entangled and send some correction information for the swap to one of the sides. Sending these update messages is an important part of the entanglement tracking protocol.

The tracker uses the standard communication lines between nodes, which can be prone to delays and messages getting lost. It is therefore important to specify on which kind of hardware the tracker is running. In the extreme case of the wires being very noisy, no message will ever reach the other side and there can be no expectations from the tracker to update other nodes. There may be a separate protocol in place that re-emits messages if they have not reached the other side after some amount of time, which means a delay on the delivery. In this project we will assume that messages can be delayed, but not completely lost.

In a big quantum network there may not be one repeater that can directly connect two end nodes, but we might need multiple nodes to make entanglement swaps to connect A and B. In Figure 4.1 a possible structure of a quantum net-

work is shown; routing for this grid structure is analysed in [18] and for a ring and sphere structure in [19] in a very idealized scenario. It is possible to find a path between A and B and reserve all the connections and nodes on this path to generate this specific entanglement. However, this would not be ideal as there may be a lot of other nodes in the network that also want to generate entanglement and use these connections. A more dynamical approach can therefore be chosen, where every node decides based only on local knowledge, which entanglement swaps it will make. Local knowledge also has the advantage that nodes do not have to communicate before making their decisions, but only to inform others about which actions they have taken. What would the communication look like in such a network? How do we make sure that everyone knows which entangled states they have and who they are connected to? The goal of the entanglement tracker is to answer these questions and come up with a concrete protocol to handle this communication. The entanglement tracker can then be used to investigate routing in more realistic models of a quantum network.

We also consider imperfect quantum memories and operations, which means that the fidelity will decrease over time when we try to entangle far away nodes. For this reason we also have to consider entanglement distillation, a way to use two low-fidelity entangled pairs between A and B to make one pair of higher fidelity between A and B.

4.2 The entanglement tracking protocol

We can start by formulating the overall goal of the entanglement tracker by defining the Current and Past databases:

Definition 1 (Current)**.** For all nodes: for all EntID $\in$ Current, EntID either describes an entangled pair in the network or the entanglement has been modified at most time T_{update} ago. For all entangled pairs in the network, either EntID is in Current for both nodes involved or the entangled pair has been modified at most T_{update} ago.

Definition 2 (Past)**.** For all nodes: for all EntID $\in$ Past, EntID describes an entangled pair in the network that was deleted at most $T_{lifetime}$ ago. For all entangled pairs in the network that were deleted at most $T_{lifetime}$ ago, either EntID is in Past for both nodes involved or the entangled pair has been deleted at most T_{update} ago.

The entanglement tracker should make sure that these requirements are met, assuming the modifications of entanglement made in the network are correctly passed on to the tracker. The relevant times T_{update} and $T_{lifetime}$ may depend

on the network, but the goal of the tracker is to update the Current and Past databases as soon as possible.

4.2.1 Commands from higher layer

The entanglement tracker can be used by a higher layer protocol that makes decisions about the entanglement within the network. The entanglement tracker is not in charge of making such decisions, only to process the resulting changes to the identifiers. A routing protocol may decide that in order to generate long-distance entanglement, an entanglement swap should happen. This routing protocol may then first ask the hardware layer to perform the physical steps required to realise the entanglement swap. After these steps have been executed, the entanglement tracker gets updated and is expected to communicate to the relevant nodes about the changes in entanglement. The entanglement tracker supports the following commands: ADD, DELETE, SWAP, DISTIL, LOOK-UP.

1. ADD(EntID)
   Adds the EntID to the tracker, should happen after generation of entanglement by both nodes individually. No update message will be send to the other node.

   Result: returns msg with:
   $msg \in$ {ok, already in Current, already in Past, list of IDs full}
   if msg = ok:
   EntID $\in$ Current
   if msg = already in Current:
   EntID $\in$ Current
   if msg = already in Past:
   EntID $\in$ Past

2. DELETE(EntID)
   Deletes an EntID from the tracker and notifies the other side of the entangled pair to do the same.

   Result: returns message msg with:
   $msg \in$ {ok, already in Past, unknown ID}
   if msg = ok:
   EntID $\in$ Past
   if msg = already in Past:
   EntID $\in$ Past

3. SWAP(EntID1,EntID2)
   Deletes the two entanglement IDs from the tracker and sends a message

to the other node involved in EntID1 and the other node in EntID2.

Result: returns message msg with:
$msg \in$ {ok, ID(s) unknown or Past }
if msg = ok:
    EntID1 $\in$ Past
    EntID2 $\in$ Past
if msg = ID(s) unknown or Past:
    No change to the IDs

4. DISTILL(Old EntIDs, New EntIDs)
Deletes the entanglement IDs in the list of Old EntIDs and adds the IDs in the list of New EntIDs.

Result: returns message msg with:
$msg \in$ {ok, Old ID(s) not in Current, New ID(s) already in Current}
if msg = ok:
    for all EntID in Old EntIDs:
        EntID $\in$ Past
    for all EntID in New EntIDs:
        EntID $\in$ Current
if msg = Old ID(s) not in Current:
    No change to the IDs
if msg = New ID(s) already in Current:
    No change to the IDs

5. LOOK-UP(node B, k)
Searches Current for entangled pairs with node B and returns either a list of k EntIDs or a list of all EntIDs with B if there are less than k EntIDs.

Result: returns list of EntIDs such that:
for all EntIDs in ID-list:
    other(EntID)=B

The python implementation of the entanglement tracker can be found in Appendix A. This uses the discrete event simulator QNetSquid, as discussed in the previous chapter.

4.2.2 Performance metrics

In this section the performance metric of the entanglement tracker will be defined. The first thing to check is that the commands discussed in the previous section are handled correctly and that the requirements of the Current and

Past databases are fulfilled. After that it is also important to keep track of the entanglement in the network in a fast and efficient way. To make this more concrete we can define performance metrics. Note that these metrics are just some parameters that may be useful to evaluate the tracker, but not all of them have to be relevant in all cases.

Time to completion: Time it takes for the Current and Past databases of all nodes in the network to be in the correct state after a number of operations. This time should never exceed T_{update} as discussed previously.
Goodness evaluation: Compare the heuristic guess of the fidelity to the actual fidelity of the entangled pair. Can there be any guarantee on the difference ϵ between goodness parameter and actual fidelity, $|Goodness - F_{real}| < \epsilon$?
Entanglement quality: This quantity is closely related to the time to completion, as the quality of final entanglement decreases with time. Depending on the entanglement tracker update protocol the messages may reach other trackers at different timings, thus influencing their decisions to make further operations. This then has an effect on the quality of the final entanglement.
Number of messages: Total number of messages needed throughout the network to update all the nodes after a set of commands. How does this scale with the number of input commands?
Length of messages: Average length of messages sent from tracker to tracker. How does this scale with the number of input commands?
Computation time: How long do nodes need (on average) to process a message send by the tracker?
Classical memory: How much classical memory do the nodes require?

Looking at the input commands to the tracker, the SWAP command is the most involved one. This is because the node making the entanglement swap can not rely on any information from the nodes it is trying to connect. Every node can decide at any point to make an entanglement swap and no matter the order of these swaps, the final entanglement identifiers should be correct. For the other commands, not as many messages have to be send and forwarded. It is therefore natural to look at a test-case where a lot of entanglement swaps happen. Before doing some numerical experiments to see how the entanglement tracker is performing in section 4.5, the next sections will be about how to update entanglement identifiers after an entanglement swap.

4.3 Identifiers after entanglement swap

In this section, the part of the entanglement identifier that is used for identifying the entangled pair will be examined. The goal of the identifier part is to give every entangled pair a unique ID within the network, such that nodes can effectively communicate about the qubits they will use to execute a quantum protocol. The goal of this section is not to prove the correctness of swap update

messages, but just to give some examples about what entanglement identifiers look like and how they can be updated after an entanglement swap. For a more detailed description see Appendix A about the internal protocol of the entanglement tracker.

As discussed in the start of this chapter, the identifier part of an elementary link consists of a few elements:

Own node ID	ID of node at which the entanglement identifier is stored
Other node ID	ID of node with which the entanglement is shared
Sequence number	Sequence number that is unique between the two nodes generating entanglement and globally unique combined with the two node IDs.

Disregarding the goodness parameters within this section, we will denote entanglement identifiers by a vector. For example, the entanglement ID in the storage of A of an entangled pair between A and B would look like:

$$E_{AB} = [A, B, seq_{AB}] \tag{4.1}$$

Note that the sequence number seq_{AB} can simply be a counter, which counts the number of entangled pairs that have been created between A and B. It does not depend on A and B, but the indices are just an indication that it is a sequence number between A and B, which means that $seq_{AB} = seq_{BA}$. If there are multiple connections between A and B, over which entanglement is generated, it could be a combination of the connection ID and the sequence number. For now we will just assume that it is a unique number between A and B.

We now take two entanglement identifiers $E_{BA} = [B, A, seq_{AB}]$ and $E_{BC} = [B, C, seq_{BC}]$, which means that node B shares an entangled pair with A and with C. Now B can make an entanglement swap and send an update message to A and C informing them that they are now entangled. This message would consist of the two identifiers E_{AB} and E_{CB}, where B flipped the order of the nodes because this is the form A and C have in their tracker. When A then receives the message, it looks in its entanglement tracker and updates the entanglement ID to entanglement with C. What would this new E_{AC} look like?

One natural option is $E_{AC} = [A, C, seq_{AC}]$, but this is hard because A and C never communicated and they have no way to make a unique sequence number, that they agree on. Simply counting is not possible, because they may obtain two entangled pairs by entanglement swap and update messages from these swaps may reach A and C in different orders. They would then confuse the two entangled pairs and entanglement would become useless.

Another option is to make the new entanglement ID $E_{AC} = [A, C, seq_{AB}seq_{BC}]$. Now the length of the entanglement ID scales with the number of swaps, but this is not the most important problem. Notice that these seq_{AB} and seq_{BC} are just sequence numbers. There could be an entanglement swap carried out

by a node D on E_{DA} and E_{DC} that also connects A and C with the ID $E_{ID} = [A, C, seq_{AD}seq_{DC}]$. It could now be the case that $seq_{AD}seq_{DC} = seq_{AB}seq_{BC}$ and then we lose the uniqueness of the entanglement identifier.

The entanglement identifiers $E_{AB} = [A, B, seq_{AB}]$ and $E_{CB} = [C, B, seq_{BC}]$ were already unique, so why not keep them exactly as they are? The new entanglement ID E_{AC} can then be given by a combination of the two:

$$E_{AC} = [[A, B, seq_{AB}], [C, B, seq_{BC}]] \tag{4.2}$$

After receiving the update message from B, then C would end up with the following entanglement identifier:

$$E_{CA} = [[C, B, seq_{BC}], [A, B, seq_{AB}]] \tag{4.3}$$

Note that the structure of the entanglement identifier has not changed that much. The first entry of the first vector is still the own node, the first entry of the second vector is now the node holding the other side of the entanglement. The rest of the identifier simply exists to make the entanglement identifier unique.

To keep the form of the identifiers fixed, it is possible to also give the identifiers of elementary links two components. For example a freshly generated link between A and B in the entanglement tracker of A would look like:

$$E_{AB} = [[A, B, seq_{AB}], [B, A, seq_{AB}]] \tag{4.4}$$

The nice property of these entanglement identifier is that A can just update the second entry of the identifier in case a node makes a swap. When B made the swap, the identifier of B then simply gets replaced by the identifier of C. What happens now when C decides to make an entanglement swap on E_{CA} and E_{CF}? It will send and update message to A and F, including E_{AC} and E_{FC}. Then A knows that it is now entangled to F and it can update the entanglement ID to:

$$E_{AF} = [[A, B, seq_{AB}], [F, C, seq_{FC}]], \tag{4.5}$$

where the second entry of the entanglement identifier is replaced to indicate entanglement with node F. This entanglement ID is unique and agreed upon by A and F after the update message from C.

However, there is still a question using this way of updating entanglement IDs. What happens when multiple nodes make a swap at the same time? In the above example, A and F were connected using two entanglement swaps, one by B and one by C. Would it also have worked if B and C made an entanglement swap at the same time? It could also be the case that the message from B was delayed to A, so the message from C to A was delivered first. In general we want the entanglement tracking protocol to be consistent for any sequence of

commands, including an arbitrary number of entanglement swaps.

In the case of multiple swaps happening at the same time, nodes also need to be able to forward update messages resulting from swaps, even if they do not hold the entanglement themselves anymore. In the last example if B and C swap at the same time, B will send an update message to C and C will send an update message to B. It is than the receiving node's responsibility to forward the messages, B forwards it to A and C should forward it to F. This is the purpose of the past entanglement database that every node holds for at least some time after they swapped it away.

These scenarios are often easier to analyse using a simulation. It turns out that there are examples, especially when the length of the repeater chain is longer, where the fixed length of the entanglement identifiers leads to serious problems. When the identifier is of fixed length, we are throwing away part of the history of the entanglement swaps that happened to establish the current identifier. However, other repeaters making entanglement swaps may still send update messages using these old versions of the entanglement identifier, because their information is outdated. In that case the node receiving the messages can not process the incoming message and the entanglement tracker does not succeed in keeping track of the entanglement.

It is therefore possible to have a more robust identifier, that scales with the number of swaps. The way to update entanglement identifiers would then be to simply add them to the list. For example when A receives the update message from B about $E_{AB} = [[A, B, seq_{AB}], [B, A, seq_{AB}]$ and $E_{BC} = [[B, C, seq_{BC}], [C, B, seq_{BC}]]$ being connected, he would update:

$$E_{AC} = [[A, B, seq_{AB}], [B, A, seq_{AB}], [B, C, seq_{BC}], [C, B, seq_{BC}]] \tag{4.6}$$

Then when he gets the message from F, he can update it again to

$$E_{AF} = [[A, B, seq_{AB}], [B, A, seq_{AB}], [B, C, seq_{BC}], \tag{4.7}$$
$$[C, B, seq_{BC}], [C, F, seq_{FC}], [F, C, seq_{FC}]] \tag{4.8}$$

The advantage of this is that nothing gets deleted from the entanglement ID, because A may receive a new update message which uses its old identifier E_{AC}. If A deleted the information about its entanglement with C, it does not recognize this E_{AC} and can not process the message.

4.4 Goodness parameter after entanglement swap

The goodness parameter has been mentioned before as part of the entanglement identifier. The goal of keeping this goodness parameter is not to have an exact

estimation of the state or fidelity of the entangled pair. The qubit can experience many different forms of decoherence and it is impossible to keep track of all of them. The goal is to have a heuristic guess of the fidelity, but of course it would be great if this guess is actually close to the real fidelity. There may be protocols in a quantum network that do not really need an estimation of the goodness at any point, in this case the goodness parameter can simply be disregarded. It is not necessary to keep it updated for the entanglement tracker, but just for a user that wants to use the entangled state for an application.

When generating entanglement over long distances within the network, the goodness paramter can be used as an indication of when to perform entanglement distillation. In that case it is necessary that the goodness parameter is tracker well, because otherwise the state may decohere beyond recovery before it is distilled.

There are often many sources of decoherence in a quantum system and many of them may be minor compared to the main sources of noise. It is therefore also important to know which sources to include and how often to update the goodness parameter. For example, the entanglement tracker of a node has an entanglement identifier in its database with some other node. Every time a gate is performed on the relevant qubit, there could be an update to the goodness parameter indicating the noise this operation has introduced. There can even be an update message to the other node after this single operation, because the entanglement has decohered and the other node may want to know this. It is clear that there is a trade-off between the quality of the goodness estimate and the amount of effort required to update it.

The rest of this section will examine a noisy entanglement swap and the effect it has on the fidelity of the entangled state. This can then be used to update the goodness parameter after a single entanglement swap.

First define the density matrices of the entangled pair between A and R and the pair between B and R:

$$\rho_{AR1} = F_1\phi^+ + (1-F_1)/3(\phi^- + \psi^+ + \psi^-), \tag{4.9}$$

$$\rho_{R2B} = F_2\phi^+ + (1-F_2)/3(\phi^- + \psi^+ + \psi^-), \tag{4.10}$$

where $\phi^\pm = |\phi^\pm\rangle\langle\phi^\pm|$ and $\psi^\pm = |\psi^\pm\rangle\langle\psi^\pm|$.

Lemma 6. *Let $F_1 = \langle\phi^+|\rho_{AR_1}|\phi^+\rangle$ be the fidelity of ρ_{AR_1} and $F_2 = \langle\phi^+|\rho_{BR_2}|\phi^+\rangle$ of ρ_{BR_2}. If R makes a noisy Bell-state measurement on R_1, R_2, defined by the following POVM-elements:*

$$
\begin{aligned}
M_{\phi^+} &= F_m\phi^+ + \frac{1-F_m}{3}(\phi^- + \psi^+ + \psi^-),\\
M_{\phi^-} &= F_m\phi^- + \frac{1-F_m}{3}(\phi^+ + \psi^+ + \psi^-),\\
M_{\psi^+} &= F_m\psi^+ + \frac{1-F_m}{3}(\phi^+ + \phi^- + \psi^-),\\
M_{\psi^-} &= F_m\psi^- + \frac{1-F_m}{3}(\phi^+ + \phi^- + \psi^+).
\end{aligned}
$$

Then the fidelity F of the entangled state between A and B to $|\phi^+\rangle$ after correction for the outcome is at least:

$$F = F_m(F_1F_2 + (1-F_1)(1-F_2)/3) + \frac{1-F_m}{3}(1 - F_1F_2 + (1-F_1)(1-F_2)/3).$$

Proof. Now look at the different unnormalised parts of the mixture $\rho_{AR1} \otimes \rho_{R2B}$, starting with the $\phi^+ \otimes \phi^+$ component:

$$|\phi\rangle_{AR1R2B} = |00\rangle|00\rangle + |00\rangle|11\rangle + |11\rangle|00\rangle + |11\rangle|11\rangle,$$

and reorder the qubits:

$$
\begin{aligned}
|\phi\rangle_{R1R2AB} &= |00\rangle|00\rangle + |01\rangle|01\rangle + |10\rangle|10\rangle + |11\rangle|11\rangle\\
&= (|\phi^+\rangle + |\phi^-\rangle)|00\rangle + (|\psi^+\rangle + |\psi^-\rangle)|01\rangle + (|\psi^+\rangle + |\psi^-\rangle)|10\rangle + (|\phi^+\rangle - |\phi^-\rangle)|11\rangle\\
&= |\phi^+\rangle|\phi^+\rangle + |\psi^+\rangle|\psi^+\rangle + |\psi^-\rangle|\psi-\rangle + |\phi^-\rangle|\phi^-\rangle
\end{aligned}
$$

If R measures $|\phi^+\rangle$, no correction by A or B is needed, if R measures ϕ^- then A or B should apply Z etc. This correction then defines which other parts of the mixture also give a $|\phi^+\rangle_{AB}$ contribution.

Without calculating everything explicitly, it can be seen that for the F_m part of the measurement, the only terms that give $|\phi^+\rangle_{AB}$ after the correction are when we select the part of the mixture where the Bell-states are the same in ρ_{AR1} and ρ_{R2B}. The probability of A and B having the same Bell-state is $p_{right} = F_1F_2 + (1-F_1)(1-F_2)/3$. The first contribution is thus $F_m p_{right}$.

For the $(1-F_m)$ part the state ends up in a wrong Bell-state, but if the Bell-states ρ_{AR1} and ρ_{R2B} are also not the same, there is a chance to still end up with $|\phi^+\rangle_{AB}$. The probability that ρ_{AR1} and ρ_{R2B} are not the same is $1 - p_{right} = 1 - (F_1F_2 + (1-F_1)(1-F_2)/3)$. Whenever the two Bell-states are not the same, there is exactly one component in the measurement that results in $|\phi^+\rangle_{AB}$, which happens with probability $(1-F_m)/3$. Effectively this

means that there were two errors that cancelled each other: one of the states was depolarised, but also the measurement was depolarised. This results in a contribution $(1 - F_m)/3 \cdot (1 - p_{right})$, summing up the two contributions the final fidelity is thus:

$$F = F_m(F_1 * F_2 + (1-F_1)*(1-F_2)/3) + \frac{1-F_m}{3}(1-F_1*F_2-(1-F_1)*(1-F_2)/3) \tag{4.11}$$

□

Note that as discussed in section 3.2.1 of [13] any state that is not Bell-diagonal can be depolarised by doing random, correlated local unitary operations to a state of the form discussed in the lemma, without affecting the fidelity to a maximally entangled state.

The depolarising parameter p, as defined in equation (1.7) and the fidelity, as used in lemma 6, are directly related in the following way:

$$F_1 = \frac{3p_1 + 1}{4}, \qquad F_2 = \frac{3p_2 + 1}{4}, \qquad F_m = \frac{3p_m + 1}{4}. \tag{4.12}$$

Substituting this into equation (4.11), the final fidelity can be expressed as:

$$F = \frac{3p_1p_2p_m + 1}{4}. \tag{4.13}$$

Note that this is exactly the fidelity we would get if we applied depolarising noise once with $p = p_1p_2p_m$.

This can then be used to update the goodness parameter after an entanglement swap. Before making the update for the entanglement swap, the goodness of the individual links can be updated to account for any previous noise. For example, if this node has kept the qubit in storage for a while, the goodness estimate of the fidelities F_1 and F_2 in lemma (6) should be updated before carrying out goodness update for the entanglement swap. After the goodness has been updated, the new entanglement ID can be sent to the other two nodes involved in the entanglement. They can then update their entanglement identifiers, including the goodness parameters.

A more difficult case for updating goodness parameters is when the nodes receiving updates about entanglement swaps have already swapped away their entanglement, especially when a lot of swaps are happening at the same time to make a long connection. All kinds of orders of updating entanglement IDs may happen. One of the ways to solve this is just to keep track of the number of entanglement swaps that have been made to establish an entangled link.

For example when there were N entanglement swaps made, the goodness parameter, estimating the fidelity of the final entangled pair, could look something like:

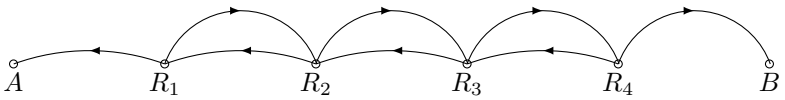
$$goodness = \frac{3(p_{create})^{N+1}(p_{meas})^{N}(p_{storage})^{2N+2}+1}{4}, \tag{4.14}$$

where p_{create} is an estimate for the depolarising parameter of a newly generated elementary link and p_{meas} is the average depolarising parameter of the Bell-state measurement. The depolarising factor $p_{storage}$, due to qubits decohering in memories is a function of time. This could be estimated using the average time a repeater stores a quantum state before executing the entanglement swap. The nodes executing the entanglement swaps may have different quantum memories, resulting in a different T_1 and T_2 time as discussed in section 1.4. There could be a guarantee of a minimal T_1 and T_2 time for all nodes in the network, so that other nodes can use these times to calculate the goodness parameter.

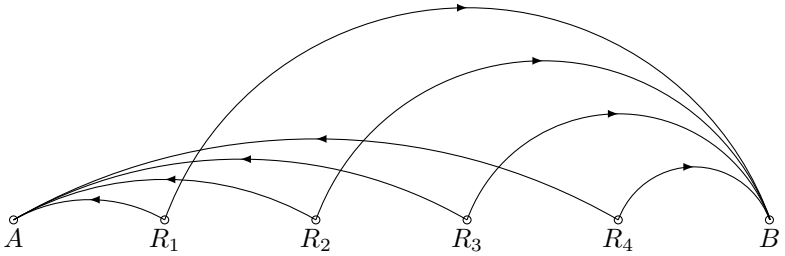
Of course this estimate is not very exact, as it assumes all the noise to be depolarising, such that we can simply multiply the depolarising parameters. It also assumes all links to be identical, whereas in a real network all nodes may have different distances and hardware. In the case of just depolarising noise it would make sense to store the depolarising parameter rather than the fidelity as a goodness parameter. Otherwise we are continuously converting depolarising parameter to fidelity and back to depolarising parameter. However, the noise may not always be just depolarising noise and then fidelity to a maximally entangled state may be a more intuitive goodness parameter.

4.5 Numerical testing

We look at a well-known example of a repeater chain. There are two end nodes A and B that want to generate entanglement, but they are far away from each other. They then use a series of quantum repeaters $R_1, R_2, ..., R_N$ to establish entanglement. Because the tracker is in charge of the classical communication in such a set-up we can look only at the classical information. In the model, at random times entanglement pops into the system separately on all links in the chain. Instead of running an actual simulation of the quantum processes in the background, the only information that is used are entanglement identifiers. These are the only objects the tracker needs and for now the quantum hardware is not taken into account. For example, when entanglement is generated on the connection between R_1 and R_2, this means that an entanglement ID will be added to the trackers of R_1 and R_2. Whenever a repeater holds two entanglement identifiers, which means that it is connected to both its neighbour on the left and the right side, it is ready to make the entanglement swap. Now there are two options to send update messages after performing such a swap:



Local knowledge protocol



Global knowledge protocol

Figure 4.2: Schematic display of local knowledge and global knowledge protocols for a repeater chain consisting of four repeaters. In the local knowledge model messages are sent only to neighbors and then passed on through the chain to A and B. In the global knowledge protocol the messages are sent directly to A and B.

Global knowledge: In this model all repeaters know which nodes are the end nodes and how to send a message to the end nodes. Instead of sending update messages to the nodes they are are entangled to, update messages can directly be sent to the end nodes. This protocol requires the minimum amount of communication: just one message to node A and one message to node B for every swap.

Local knowledge: In this model the entanglement tracking protocol will be used to send update messages. The only information nodes have is the entanglement identifiers they hold and the messages they send go to the node holding the other side of the entanglement. The number of messages in this case is not fixed and messages have to be forwarded and processed all the time.

The two communication models are also displayed in Figure 4.2. The goal of the simulation is to compare these two cases regarding one of the performance metrics defined in section 4.2. There are a few parameters in this simulation, that will be the same for the local knowledge and global knowledge protocol.

N Number of repeaters $R_1, R_2, ..., R_N$ connecting A and B.

Generation model Distribution according to which entanglement pops into the system, assumed to be the same for all connections. As entanglement generation is

normally probabilistic, samples from this distribution will be taken.

Communication model The model that is used for sending classical messages in the network. To keep things simple, every node (including the repeaters) will be connected with a classical connection to any other node in the chain. The time it takes to communicate is proportional to the distance between the nodes and represents the time it takes for photons to travel over a fibre to another node.

Processing model This parameter is about the processing of messages in a node. Whenever a node receives a message, it will need some time to process it, for example by looking in its local entanglement databases. This processing time will depend on the processing power of the node, on the size of the entanglement database, etc. In this simulation it will be assumed fixed, but in general it is an interesting question how this processing time behaves as a function of the network size, for example the length of the repeater chain.

In the following simulations the repeaters will all be separated by the same distance to their neighbours. We set $A = R_0$ and $B = R_{N+1}$ and define the distance in the chain for nodes R_a and R_b as $|a - b|$; to get the actual distance between nodes this has to be multiplied by the elementary separation.

The entanglement tracking protocol starts immediately at the start of the simulation. This means that whenever entangled links are generated, they can be put into the entanglement tracker and can be processed and reacted to. For example, in the case of global information, the end node A can always receive update messages from entanglement swaps that have happened somewhere in the chain, even if it hasn't generated entanglement with R_1 yet. It then stores these update messages until they can be processed and combined with the information in its Current entanglement database.

As a first example we can assume very simple models for all of the time distributions. After one time step entanglement pops into the system on all links and at that point all the entanglement swaps happen at the same time. The results are then either communicated directly to the end-nodes using global knowledge, or passed through the chain using the (local knowledge) entanglement tracker. The time units are arbitrary, entanglement generation always takes 1 unit, data processing takes 1 unit, communication takes 1 unit times the distance in the chain.

In Figure 4.3, it can be seen that there is a factor difference between the global and local knowledge model. This is due to the processing time. For example, when sending a message directly from R_4 to R_1, the total time is the sum of the distance between R_4 and R_1 and processing time of R_1, which is $3+1 = 4$. In the case of passing it through the chain, the total time is $3+3 = 6$, because R_2 and R_3 also have to process it. This may not be a very realistic model, but it shows that when processing of messages takes time, it is important

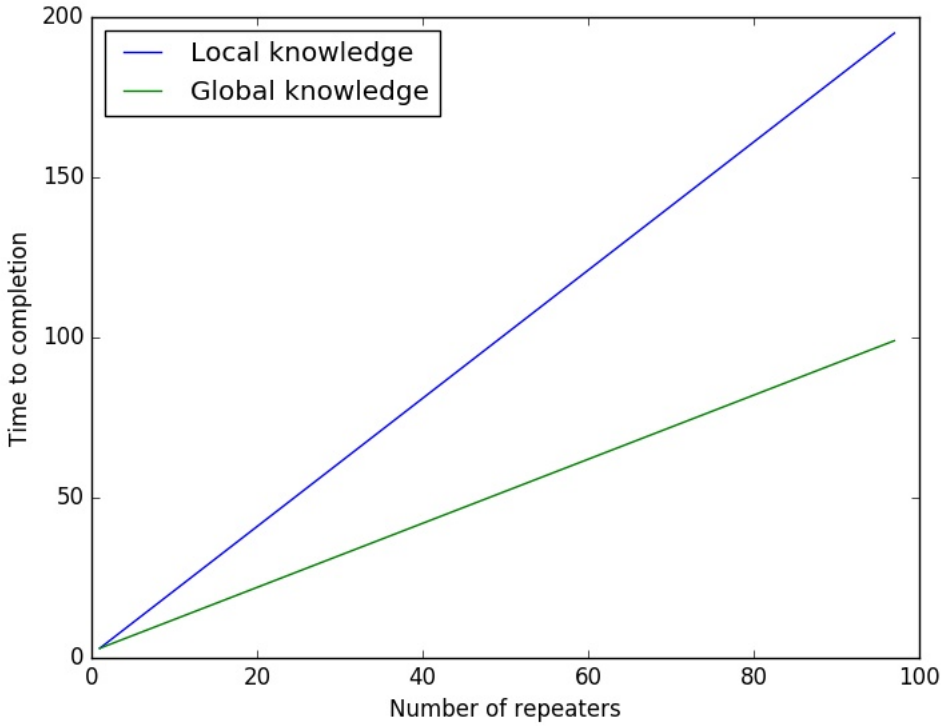


Figure 4.3: Time to completion of the entanglement tracker, as a function of the length of the repeater chain. The repeaters are always separated by the same distance, which means that the total distance scales linearly with the number of repeaters. Either the updates are sent directly to the end nodes (global information), or the information is passed through the chain (local information). The time units are arbitrary, entanglement generation always takes 1 unit, data processing takes 1 unit, communication takes 1 unit times the distance in the chain.

to send messages efficiently.

The total time to completion can be calculated exactly too, using the fact that the message that takes the longest time is always sent from R_N to A and from R_1 to B. The distance is in this case N and the processing time 1 in the case of global knowledge, which means only processing by A and B. In the case of local knowledge, all the intermediate repeaters have to process, which means a total processing time of N. Adding one for the time it takes to generate entanglement, this results in a total time for the global knowledge protocol of $N + 2$ and for local knowledge of $2N + 1$.

Now it is possible to look at a more realistic entanglement generation protocol. The time it takes to generate entanglement is now given by the single click distribution, as discussed in the QNetSquid section. For every link in the chain, a sample is taken from this distribution. The communication and processing models are the same as in the previous example, but now an actual distance between the repeaters is set to $10km$. The speed at which messages can be send is set to $2 \cdot 10^5 km/s$. This means that the communication time is $5 \cdot 10^{-5} s$ times the distance in the chain and the processing time is chosen to be the same as elementary link communication $5 \cdot 10^{-5} s$. The result is displayed in Figure 4.4.

In Figure 4.4 it can be seen that the local information and global information protocols are much closer for realistic entanglement generation. This is because entanglement generation takes most of the time to completion and the classical communication protocol has only a minor influence. The communication and processing model is still not very realistic, but it demonstrates the difference between local and global information.

Concluding this chapter, we have investigated the properties of an entanglement tracker. It is important to define entanglement identifiers that are assigned to every entangled pair in the network. These can be used by the entanglement trackers to keep their databases about current and past entanglement information up-to-date. The entanglement identifiers can also give a heuristic guess of the quality of the entanglement, called the goodness parameter. How exactly to update these goodness parameters after entanglement swaps depends on the details of the network and application by the user. For some applications it may be enough to keep track of only the major factors of noise, where in other applications we may need a precise calculation of all the noise sources to have a good estimate for the goodness.

The main function of the entanglement tracker is to be a tool that can be used by higher layer protocols, such that they do not have to worry about how to handle classical communication. Instead they can focus on the routing decisions that have to be taken, for example which entanglement swaps to make in the network. Here the entanglement tracker has also been used to directly simulate the time it takes to generate a repeater chain using a number of repeaters.

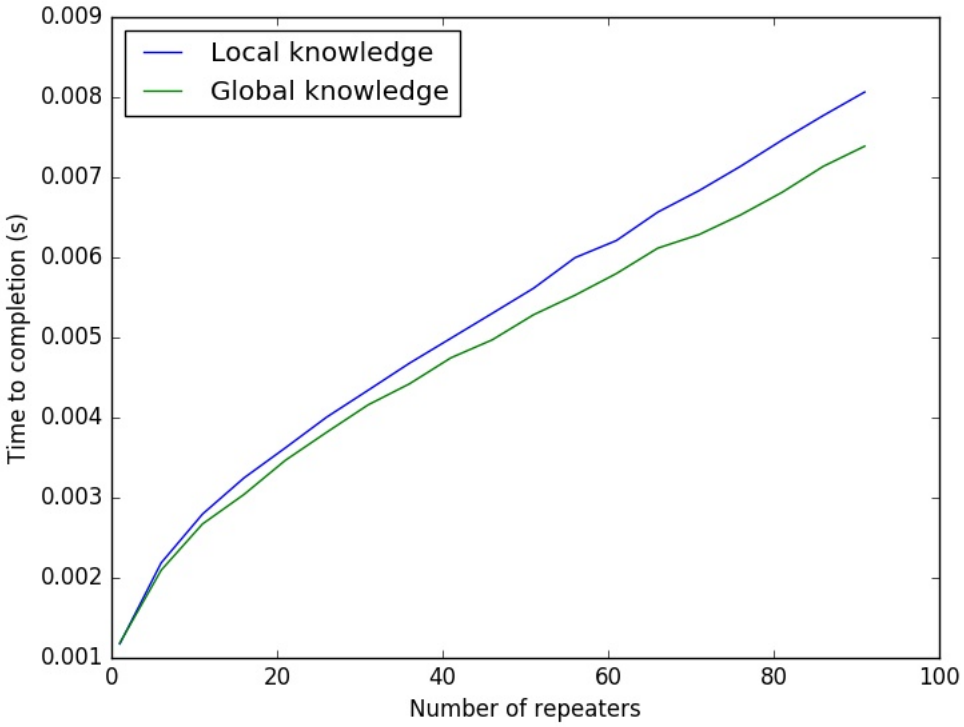


Figure 4.4: Time to completion of the entanglement tracker, as a function of the length of the repeater chain. Either the updates are sent directly to the end nodes (global information), or the information is passed through the chain (local information). The communication time is $5{\cdot}10^{-5}s$ times the distance in the chain and the processing time is the same as elementary link communication $5 \cdot 10^{-5}s$. The entanglement generation time is taken from the single click distribution - as discussed in section 3.2 - and a separate sample from this distribution is taken for every link in the chain. Because the entanglement generation is probabilistic, an average over a large number of runs has to be taken, which is 1000 for this plot.

Using a model for the entanglement generation time, the entanglement tracker can do all the classical communication required.

For further development, it would be interesting to look at the interaction between the entanglement tracker and the physical layer where the quantum processes happen. The decisions taken by the entanglement tracker could then effect the fidelity of entangled states provided by the quantum hardware. The properties of the physical set-up can then also be used to give a more realistic model for classical communication and classical processing of the messages sent by the entanglement tracker. The performance of the tracker can then be tested in a more detailed case.

The entanglement tracker can also be used to investigate a more complicated network than the fixed repeater chain. For example, the problem of generating long distance entanglement within the network in the most efficient way, where multiple paths between parties are possible. The combination of entanglement tracking and routing may lead to new insights about how to distribute entanglement in a quantum network.

Bibliography

[1] Masahide Sasaki. Quantum networks: where should we be heading? *Quantum Science and Technology*, 2(2):020501, 2017.

[2] V. Lipinska, G. Murta, and S. Wehner. Anonymous transmission in a noisy quantum network using the W state. *ArXiv e-prints*, 1806.10973, June 2018.

[3] Jérémy Ribeiro, Le Phuc Thinh, J ędrzej Kaniewski, Jonas Helsen, and Stephanie Wehner. Device independence for two-party cryptography and position verification with memoryless devices. *Phys. Rev. A*, 97:062307, Jun 2018.

[4] B. Hensen, H. Bernien, A. E. Dréau, A. Reiserer, N. Kalb, M. S. Blok, J. Ruitenberg, R. F. L. Vermeulen, R. N. Schouten, C. Abellán, W. Amaya, V. Pruneri, M. W. Mitchell, M. Markham, D. J. Twitchen, D. Elkouss, S. Wehner, T. H. Taminiau, and R. Hanson. Loophole-free Bell inequality violation using electron spins separated by 1.3 kilometres. *Nature*, 526:682–686, October 2015.

[5] A. O. Niskanen, K. Harrabi, F. Yoshihara, Y. Nakamura, S. Lloyd, and J. S. Tsai. Quantum coherent tunable coupling of superconducting qubits. *Science*, 316(5825):723–726, 2007.

[6] J. I. Cirac and P. Zoller. Quantum computations with cold trapped ions. *Phys. Rev. Lett.*, 74:4091–4094, May 1995.

[7] A. Blais, R.-S. Huang, A. Wallraff, S. M. Girvin, and R. J. Schoelkopf. Cavity quantum electrodynamics for superconducting electrical circuits: An architecture for quantum computation. *Physical Review A*, 69(6):062320, June 2004.

[8] Lilian Childress and Ronald Hanson. Diamond nv centers for quantum computing and quantum networks. *MRS Bulletin*, 38(2):134–138, 2013.

[9] N. Kalb, A. A. Reiserer, P. C. Humphreys, J. J. W. Bakermans, S. J. Kamerling, N. H. Nickerson, S. C. Benjamin, D. J. Twitchen, M. Markham, and R. Hanson. Entanglement distillation between solid-state quantum network nodes. *Science*, 356(6341):928–932, 2017.

[10] F Rozpędek, R. Yehia, K. Goodenough, M. Ruf, P. Humphreys, , R. Hanson, S. Wehner, and D. Elkouss. Near-term repeater experiments with NV centers: overcoming the limitations of direct transmission. In preparation.

[11] N. Sangouard, C. Simon, H. de Riedmatten, and N. Gisin. Quantum repeaters based on atomic ensembles and linear optics. *ArXiv e-prints*, 0906.2699, June 2009.

[12] N. H. Nickerson, J. F. Fitzsimons, and S. C. Benjamin. Freely Scalable Quantum Technologies Using Cells of 5-to-50 Qubits with Very Lossy and Noisy Photonic Links. *Physical Review X*, 4(4):041041, October 2014.

[13] D. Deutsch, A. Ekert, R. Jozsa, C. Macchiavello, S. Popescu, and A. Sanpera. Quantum Privacy Amplification and the Security of Quantum Cryptography over Noisy Channels. *Physical Review Letters*, 77:2818–2821, September 1996.

[14] J. Dehaene, M. van den Nest, B. de Moor, and F. Verstraete. Local permutations of products of Bell states and entanglement distillation. *Physical Review A*, 67(2):022310, February 2003.

[15] F. Rozpedek, T. Schiet, L. P. Thinh, D. Elkouss, A. C. Doherty, and S. Wehner. Optimizing practical entanglement distillation. *Physical Review A*, 97(6):062333, June 2018.

[16] W. Dür and H. J. Briegel. Entanglement purification and quantum error correction. *Reports on Progress in Physics*, 70:1381–1424, August 2007.

[17] S. B. van Dam, P. C. Humphreys, F. Rozpędek, S. Wehner, and R. Hanson. Multiplexed entanglement generation over quantum networks using multi-qubit nodes. *Quantum Science and Technology*, 2(3):034002, September 2017.

[18] M. Pant, H. Krovi, D. Towsley, L. Tassiulas, L. Jiang, P. Basu, D. Englund, and S. Guha. Routing entanglement in the quantum internet. *ArXiv e-prints*, 1708.07142, August 2017.

[19] E. Schoute, L. Mancinska, T. Islam, I. Kerenidis, and S. Wehner. Shortcuts to quantum network routing. *ArXiv e-prints*, 1610.05238, October 2016.

[20] N. Kalb, A. A. Reiserer, P. C. Humphreys, J. J. W. Bakermans, S. J. Kamerling, N. H. Nickerson, S. C. Benjamin, D. J. Twitchen, M. Markham, and R. Hanson. Entanglement distillation between solid-state quantum network nodes. *Science*, 356:928–932, June 2017.

[21] S. D. Barrett and P. Kok. Efficient high-fidelity quantum computation using matter qubits and linear optics. *Physical Review A*, 71(6):060310, June 2005.

[22] P. C. Humphreys, N. Kalb, J. P. J. Morits, R. N. Schouten, R. F. L. Vermeulen, D. J. Twitchen, M. Markham, and R. Hanson. Deterministic delivery of remote entanglement on a quantum network. *Nature*, June 2018.

[23] C. Cabrillo, J. I. Cirac, P. García-Fernández, and P. Zoller. Creation of entangled states of distant atoms by interference. *Physical Review A*, 59:1025–1033, February 1999.

[24] E. T. Campbell and S. C. Benjamin. Measurement-Based Entanglement under Conditions of Extreme Photon Loss. *Physical Review Letters*, 101(13):130502, September 2008.

Appendix A

Internal protocols of the entanglement tracker

This is the python code for the implementation of the entanglement tracker in QNetSquid. EasySquid is a repository extending QNetSquid from which the implementation of quantum nodes, classical connections and protocols using these nodes and connections are used. The first functions of the class are the input commands ADD, DELETE, SWAP and DISTILL and LOOK-UP. The remainder of the class is about processing information from other nodes and sub-protocols that should be executed whenever one of the commands is given. The full code consists of three classes: the entanglement tracker, and two classes that make an entanglement identifier in a different way.

```
from easysquid.easyprotocol import EasyProtocol
from netsquid import pydynaa
from netsquid.pydynaa import EventHandler
from netsquid.simutil import sim_time
import numpy as np
import copy


class EntanglementTracker(EasyProtocol):
    """
    Entanglement tracking protocol to run locally on a node. The entanglement tracker
    will keep track of all classical information about entanglement that this node is
    involved in. It will interact with the EntanglmentTracker protocols of different
    nodes to keep information up-to-date.
    Every time this node generates entanglement with some remote node,
    an entanglement ID (EntID) should be put into the tracker using the add() function.

    Every time entanglement is consumed, it should be removed using the delete() function.

    If this node made an entanglement swap using its qubits of two entangled pairs, the
    swap() function should be called. This will update the other nodes about their new
```

```
    entanglement and delete the pairs from our database.
    The distil() function is for updating the entanglement IDs after entanglement
    distillation, removing the old ID(s) and adding the new ID(s).
    To be able to handle messages arriving to this node out of order, there are two
    databases, current and past. Messages are assumed to be able to be delayed,
    but should eventually reach the target for the tracker to stay consistent.
    Parameters
    ----------
    node : :obj:`easysquid.qnode.QuantumNode`
        Node at which the entanglement tracker will be run
    conns : list of :obj:`easysquid.connection.ClassicalConnection`
        Classical connections of this node that should be used by the entanglement tracker
        .
        Note that all messages from these connections will be taken by the tracker.
    database_size : int
        Number of entanglement IDs that can be stored in the current database of the
        tracker,
    qubit_lifetime : float, optional
        Time after which the tracker should throw away entanglement
        identifiers from the past list.
        NOTE: when the qubit_lifetime is not the default setting None, the tracker
        will keep checking the past database every qubit_lifetime time units. This
        will not terminate, so the simulation should be run for a fixed amount of
        time, or terminated by a different protocol.
    Attributes
    ----------
    node : :obj:`easysquid.qnode.QuantumNode`
        Node at which the entanglement tracker will be run
    size : int
        Number of entanglement IDs that can be stored in the tracker,
        assumed the same for the current, past and update database.
    current : list
        List of all the entanglement IDs of entangled pairs this node
        currently holds. This list gets updated continuously by updates
        from other nodes and for example the entanglement generation protocol
        of this node.
    past : list of two lists
        List of all entanglement IDs that were recently deleted from current.
        This database is used to forward swap-update messages after entanglement
        has already been swapped away to another node.
        The first list is used for entanglment IDs that were deleted.
        Second list is used to store entanglement IDs that
        were consumed in an entanglement swap.
    """
    # Possible messages in the entanglement tracking protocol:
    CMD_SWAP_UPDATE = 0
    CMD_DELETE_UPDATE = 1

    def __init__(self, node, conns, database_size, qubit_lifetime=None):
        super().__init__(node=node)
```

```
        self.node = node
        self.conns = conns
        self.size = database_size
        self.qubit_lifetime = qubit_lifetime

        # set up empty databases for current and past
        self.current = []
        self.past = [[], []]

        # Listen to connections of self.node for update messages from other trackers
        self._listen_to_conns()

        # Set up a check of the past database for outdated information
        self._remove_outdated_past()

        # Set up command handlers for update messages
        self.commandHandlers = {
            self.CMD_SWAP_UPDATE: self.cmd_SWAP_UPDATE,
            self.CMD_DELETE_UPDATE: self.cmd_DELETE_UPDATE,
        }

    def add(self, EntID):
        """
        Adds EntID to the tracker. No update messages will be send to other nodes,
        the other side of the entanglement should input the ID to its own tracker.

        returns:
            "ok"                    -> EntID successfully added to self.current
            "already in Current"    -> EntID was already in self.current
            "already in Past"       -> EntID was already in self.past
            "list of IDs full"      -> self.current is full
        """
        return self._enter(EntID)

    def delete(self, EntID, notify_remote=True):
        """
        Deletes an EntID from the tracker and notifies the other side of the
        entangled pair to do the same.
        Parameters
        ----------
        EntID : list
            Entanglement ID to delete
        notify_remote : bool, optional
            If False, does not notify remote node of EntID of deletion of entanglement.
            In this case both nodes should know that this entanglement has been deleted.
        returns:
            "ok"                -> EntID succesfully removed from self.current, added to
                self.past
            "already in Past"   -> EntID was already in self.past
            "unknown ID"        -> EntID unknown to the tracker, no further actions will
```

```
            be taken.
        """
        if (self._in_current(EntID) and notify_remote):
            # The first entry of the last element of the list of identifiers is always the
                other
            # side of the entanglement
            node_A = EntID.ID[-1][0]

            # Invert the identifiers of the entanglement ID so that node_A can recognise
                it
            EntID.inv_entID()

            # Prepare the message for node_A and put it on the connection
            msg = [self.CMD_DELETE_UPDATE, EntID]
            self.node.nodes[node_A].connection.put_from(self.node.nodeID, msg)
        return self._remove(EntID)

    def swap(self, EntID1, EntID2, corr=np.array([0, 0])):
        """
        Delete the two entanglement IDs from the tracker and sends a swap-update message
        to the two other nodes to update their entanglement IDs.
        corr is classical information about the outcome of the Bell-state measurement
        that was used to perform the entanglement swap. This is normally two bits
        corr = [a, b] and one of the sides of the entangled pair should apply the
        correction C = X**a * Z**b. This commutes with more entanglement swaps, which
        means that the correction can be applied only at the very end of the protocol.
        returns:
            "ok"                        -> moved IDs from self.current to self.past
            "ID(s) unknown or Past"     -> one of the IDs was not in self.current
        """
        # Check if both of the entanglement IDs are in self.current
        if (self._in_current(EntID1) and self._in_current(EntID2)):
            # Find the other nodes in the entanglement IDs
            node_A = EntID1.ID[-1][0]
            node_B = EntID2.ID[-1][0]

            # Add the IDs of the two entanglement IDs together
            ID = EntID1.ID[::-1] + EntID2.ID

            # Prepare entanglement IDs for node_A and node_B, use TravelID to communicate
            # All the correction information goes into the message to node_A
            ID_to_A = TravelID(ID=ID, goodness=EntID1.goodness * EntID2.goodness, corr=
                EntID1.corr + EntID2.corr)
            ID_to_B = TravelID(ID=ID[::-1], goodness=EntID1.goodness * EntID2.goodness,
                corr=0)

            # Add the correction information in the entanglement ID for node_A
            ID_to_A.corr += corr

            # Prepare the messages for node_A and node_B and put them on the connection
```

```
            msg_A = [self.CMD_SWAP_UPDATE, ID_to_A]
            msg_B = [self.CMD_SWAP_UPDATE, ID_to_B]
            self.node.nodes[node_A].connection.put_from(self.node.nodeID, msg_A)
            self.node.nodes[node_B].connection.put_from(self.node.nodeID, msg_B)

            # Remove the entanglement IDs from our current database
            self.current.remove(EntID1)
            self.current.remove(EntID2)

            # Make a copy of the ID send to A and B to add to our past database
            ID_past_A = copy.deepcopy(ID_to_A)
            ID_past_B = copy.deepcopy(ID_to_B)

            # Set correction information to 0 because it is now in the entanglement ID to
                A
            ID_past_A.corr = np.array([0, 0])
            ID_past_B.corr = np.array([0, 0])

            # Add the entanglement IDs to our past database
            self.past[1].append(ID_past_A)
            self.past[1].append(ID_past_B)
            return "ok"
        else:
            return "ID(s) unknown or Past"

    def distil(self, old_entIDs, new_entIDs):
        """
        Deletes the set of old_entIDs and replaces them by the new_entIDs. There are
        no restrictions on the size of the lists, but normally distillation consumes
        more pairs than it produces, but with a higher fidelity.
        Parameters
        ----------
        old_entIDs : list of EntID
            IDs to be deleted from self.current
        new_entIDs : list of EntID
            IDs to be added to self.current

        returns:
            "ok"                            -> added and removed the EntIDs from self.
                current
            "Old ID(s) not in Current"      -> one of the old_entIDs was not in self.
                current
            "New ID(s) not unknown"         -> one of the new_entIDs was already in self.
                current
        """
        # Check if all consumed EntIDs are in current
        for entID in old_entIDs:
            if not self._in_current(entID):
                return "Old ID(s) not in Current"
```

```
        # Check if all new EntIDs are unknown to self.past and self.current
        for entID in new_entIDs:
            if (self._in_current(entID) or self._in_past(entID)[0]):
                return "New ID(s) not unknown"

        # Remove the old_entIDs from self.current and add them to self.past
        for entID in old_entIDs:
            self._remove(entID)

        # Add the new_entIDs to self.current
        for entID in new_entIDs:
            self._enter(entID)

    def look_up(self, remote_ID, num_pairs=1):
        """
        Returns at most num_pairs entanglement IDs corresponding to
        entanglement between this node and node remote_ID.
        """
        # Make empty list to fill with EntanglementIDs
        id_list = []
        counter = 0
        for x in self.current:
            # Check if other side of the entanglement is remote_ID
            if x.ID[-1][0] == remote_ID:
                id_list.append(x)
                counter += 1

                # Return if we have found num_pairs of entanglement IDs with remote_ID
                if counter == num_pairs:
                    return id_list

        # Return all the entanglement IDs with remote_ID we could find
        return id_list

    def _listen_to_conns(self):
        # Loop over all incoming connections of self.node
        for conn in self.conns:

            # Check which side of the connection self.node is and set remoteID to the
                other side
            if self.node.nodeID == conn.idA:
                channel_in = conn.channel_from_B
                remoteID = conn.idB
            if self.node.nodeID == conn.idB:
                channel_in = conn.channel_from_A
                remoteID = conn.idA

            # Set up event handlers using the process function
            update_handler = EventHandler(lambda e, remoteID=remoteID: self.process(
                remoteID))
```

```
            self._wait(update_handler, entity=channel_in, event_type=channel_in.
                _EVT_MESSAGE_READY)

    def _remove_outdated_past(self):
        """Check of past database for old information and remove it"""
        # Only run this function if self.qubit_lifetime is not None
        if self.qubit_lifetime is not None:

            # Find the current simulation time
            time = sim_time()

            # Loop over the deleted entanglement IDs
            for EntID in self.past[0]:
                # Check if the last time the goodness was updated was more than
                    qubit_lifetime ago
                if time - EntID.t_goodness > self.qubit_lifetime:
                    # Remove the EntID from past
                    self.past[0].remove(EntID)
            # Do the same for the swapped away Entanglement IDs
            for EntID in self.past[1]:
                if time - EntID.t_goodness > self.qubit_lifetime:
                    self.past[1].remove(EntID)

            # Set up the next check of the past database for old information
            ev_check_past = pydynaa.EventType("CLEAN_PAST", "Check past for old
                information")

            # Schedule this event every self.qubit_lifetime time units
            event = self._schedule_after(self.qubit_lifetime, ev_check_past)

            # React to the ev_check_past event by calling the remove_outdated_past
                function again
            handler = EventHandler(lambda event: self._remove_outdated_past())

            # Wait for the event to happen
            self._wait_once(handler, entity=self, event=event)

    def process(self, remoteID):
        """
        Handle incoming messages from the EntanglementTracker of other nodes.
        Supported messages:
            - swap-update
            - delete-update
        """

        # Get the message of the connection
        [(cmd, data), t] = self.node.nodes[remoteID].connection.get_as(self.node.nodeID)

        # Process the message based on the command
        self.commandHandlers[cmd](data)
```

```
def cmd_SWAP_UPDATE(self, data):
    # Obtain the entanglement ID from the data
    EntID = data

    # Check if EntID overlaps with any entanglement ID in self.current
    for x in self.current:
        # Automatically updates x if there is overlap between x and EntID
        if self.combine_if_overlap(x, EntID):
            # We can return because the information of EntID has been added to x
            return

    # Check if the EntID overlaps with any entanglementID that was swapped away
    for x in self.past[1]:
        # Updates x if there is overlap between x and EntID
        if self.combine_if_overlap(x, EntID):

            # Check if we are not swapping entanglement in a circle, resulting in a
                loop of messages
            if not x.check_ids_unique():
                return

            # Find node that entanglement was swapped away to
            node_A = x.ID[0][0]

            # Forward the update message to node_A using the element of self.past
            msg = [self.CMD_SWAP_UPDATE, x]
            self.node.nodes[node_A].connection.put_from(self.node.nodeID, msg)

            # Make a copy of the ID sent to node_A and replace it in self.past
            ID_past = copy.deepcopy(x)
            ID_past.corr = np.array([0, 0])
            self.past[1].remove(x)
            self.past[1].append(ID_past)
            return

    # Check if EntID overlaps with any entanglementID that was deleted
    for x in self.past[0]:
        if self.combine_if_overlap(x, EntID):
            # No further action needed
            return

def cmd_DELETE_UPDATE(self, data):
    EntID = data
    for x in self.current:
        # Check if the first identifier matches, this is enough to delete it
        if (EntID.ID[0] == x.ID[0]):
            self._remove(x)

def _in_current(self, EntID):
```

```
        """
        Checks if EntID is in the current database.
        """
        for x in self.current:
            if EntID.equal_ID(x):
                return True
        return False

    def _in_past(self, EntID):
        """
        Checks if EntID is in the past database and also returns whether EntID is stored
        in self.past as a pair or alone.
        """
        # EntID in self.past due to deletion
        for x in self.past[0]:
            if EntID.equal_ID(x):
                return [True, 1]

        # EntID in Past due to an entanglement swap
        for x in self.past[1]:
            if EntID.equal_ID(x):
                return [True, 2]

        return [False, 0]

    def _enter(self, EntID):
        """
        Try to add an EntID to self.current
        """
        if self._in_current(EntID):
            return "already in Current"
        if self._in_past(EntID)[0]:
            return "already in Past"
        if len(self.current) >= self.size:
            return "list of IDs full"
        else:
            self.current.append(EntID)
            return "ok"

    def _remove(self, EntID):
        """
        Try to remove an EntID from self.current and add to self.past
        """
        if self._in_past(EntID)[0]:
            return "already in Past"
        if self._in_current(EntID):
            self.current.remove(EntID)
            self.past[0].append(EntID)
            return "ok"
        else:
```

```
        return "unknown ID"

def combine_if_overlap(self, EntID1, EntID2):
    """
    Combine EntID1 with EntID2 after an entanglement swap.

    returns bool: True if there is overlap in the identifiers, otherwise False

    Parameters
    ----------
    EntID1 : :py:class:`~easysquid.examples.entanglementtracker.EntanglementID`
        Entanglement ID that will be updated using this function
    EntID2 : :py:class:`~easysquid.examples.entanglementtracker.EntanglementID`
        Entanglement ID that EntID1 should be compared to, in the case
        of successful combination EntID2 can be thrown away.
    """
    # Make a list for the indices of EntID2.ID that occur in EntID1.ID
    ind = []

    # The number of identifiers that match between EntID2.ID and EntID1.ID
    overlap = 0

    # Loop over elements of EntID1.ID and check if they are in EntID2.ID
    for ids in EntID1.ID:
        if ids in EntID2.ID:
            if overlap == 0:
                # First overlapping element, add the index of EntID2.ID
                ind.append(EntID2.ID.index(ids))
                overlap = 1
            elif (EntID2.ID.index(ids) == ind[-1] + 1):
                # Overlapping elements have the same order in EntID1.ID and EntID2.ID
                ind.append(EntID2.ID.index(ids))
                overlap += 1
            else:
                # This element of EntID1.ID does not overlap with EntID2.ID anymore,
                    break.
                break

    # For successful match the overlap should at least be two, the length of a newly
        created EntanglementID.ID
    if overlap < 2:
        return False

    # Calculate the factor that should be added to EntID1.goodness based on lenghts of
         EntID1.ID and EntID2.ID
    goodness_factor = EntID2.goodness ** ((len(EntID2.ID) - overlap) / len(EntID2.ID))
    EntID1.goodness = EntID1.goodness * goodness_factor

    # Set the time of goodness by checking the simulation time
    EntID1.t_goodness = sim_time()
```

```
        # Add the correction information of EntID2 to EntID1
        EntID1.corr += EntID2.corr

        # The index of the first element of the overlap in EntID1.ID
        my_ind0 = EntID1.ID.index(EntID2.ID[ind[0]])

        # True if EntID2.ID has more elements after the overlap than EntID1.ID
        if len(EntID2.ID) - ind[0] > len(EntID1.ID) - my_ind0:
            # Add the extra elements of EntID2.ID to EntID1.ID at the end
            EntID1.ID += EntID2.ID[ind[0] + overlap:]

        # True if EntID2.ID has more elements before the overlap than EntID1.ID
        if ind[0] > my_ind0:
            # Add the extra elements of EntID2.ID to EntID1.ID at the begin
            EntID1.ID = EntID2.ID[:EntID1.ID.index(EntID2.ID[ind[0]]) - ind[0]] + EntID1.
                ID
        return True


class EntanglementID():
    """
    Construct an entanglement ID

    Parameters
    ----------
    nodeID1 : int
        ID of the node that will store this entanglement ID
    nodeID2 : int
        ID of the node that holds the other side of the entangled pair
    pair_ID : int (float?)
        ID of the entangled pair
    goodness : float, optional
        Heuristic estimate of the fidelity of the entangled pair.
    ToG : float, optional
        Simulation time at which the goodness parameter was calculated
    ToC : float, optional
        Simulation time at which the entangled pair was created
    Attributes
    ----------
    ID : list
        ID of the entangled pair
    goodness : float
        Heuristic estimate of the fidelity of the entangled pair.
    t_goodness : float
        Simulation time at which the goodness parameter was calculated
    t_creation : float
        Simulation time at which the entangled pair was created
    corr : numpy.array of int (length 2)
        corr is classical information about the outcome of the Bell-state measurement
```

```
        that was used to perform the entanglement swap. This is normally two bits
        corr = [a, b] and one of the sides of the entangled pair should apply the
        correction C = X**a * Z**b. This commutes with more entanglement swaps, which
        means that the correction can be applied only at the very end of the protocol.

    For the entanglement ID to be unique the following conditions should hold:
    - nodeID1, nodeID2 is unique within the network
    - pair_ID is unique for any entangled pair between nodes nodeID1 and nodeID2
    Note that the entanglement ID that nodeID1 stores is different from the
    entanglement ID that nodeID2 stores. This makes updating changes to entanglement
    easier, for example because our node is always self.ID[0][0].
    """
    def __init__(self, nodeID1, nodeID2, pair_ID, goodness=1, t_goodness=0, t_creation=0):
        self.goodness = goodness
        self.t_goodness = t_goodness
        self.t_creation = t_creation
        self.ID = self._make_new_id(nodeID1, nodeID2, pair_ID)
        self.corr = np.array([0, 0])

    def _make_new_id(self, nodeID1, nodeID2, pair_ID):
        """
        Make new ID between nodes nodeID1 and nodeID2 using pair_ID

        Parameters
        ----------
        nodeID1 : int
            ID of the node that will store this entanglement ID
        nodeID2 : int
            ID of the node that holds the other side of the entangled pair
        pair_ID : int (float?)
            ID of the entangled pair

        """
        # Construct the identifiers that make the ID
        PI1 = [nodeID1, nodeID2, pair_ID]
        PI2 = [nodeID2, nodeID1, pair_ID]

        # Add them in a list to make the ID
        ID = [PI1, PI2]
        return ID

    def equal_ID(self, other):
        """
        returns bool: True if the identifiers of self and other EntanglementID are the
            same, otherwise False

        Parameters
        ----------
        other : :py:class:`~easysquid.examples.entanglementtracker.EntanglementID`
            Entanglement ID that self should be compared to
```

```
        """
        if (self.ID == other.ID):
            return True
        else:
            return False

    def inv_entID(self):
        """
        Reverse the identifiers of EntID
        """
        self.ID.reverse()

    def check_ids_unique(self):
        """
        Checks if no elements occur twice in self.ID

        returns bool: True if there are duplicates in self.ID, otherwise False
        """
        ID = []
        for x in self.ID:
            if x in ID:
                return False
            else:
                ID.append(x)
        return True


class TravelID(EntanglementID):
    """
    Class can be used to quickly make an EntanglementID object, if we already have
    the right form of ID.
    """

    def __init__(self, ID, goodness=1, t_goodness=0, t_creation=0, corr=np.array([0, 0])):
        self.ID = ID
        self.goodness = goodness
        self.t_goodness = t_goodness
        self.t_creation = t_creation
        self.corr = corr
```

Appendix B

Lost Snippets

Here are some small projects that I started to work on, but did not go into a lot of detail about in the main text.

B.1 Network

This is about entanglement generation in a larger network. Here we set up the network using an adjacency matrix, where each non-zero entry indicates a connection between two nodes. The protocol can then be initialised by supplying this adjacency matrix and some other parameters of the network. For example the generation time of qubits, the fidelity of the qubits and the noise rates of memories and fibres. We then make the physical components in the network; QFibres on all connections and a QSource for each connection. All the nodes also get a QMemory for all the connections they have. These components can then be used to initialise the EntangleNodes protocol for every connection. This can be the basis of a network where different protocols wait for the result of the EntangleNodes protocols to perform tasks like quantum teleportation or QKD. The idea of this snippet was to test some routing protocols using this network. Then we noticed that handling the classical communication in such a network was hard, which resulted in the construction of the entanglement tracker.

B.2 Distillation

The topic of entanglement distillation has been mentioned a lot of times before. This motivated us to generalise this distillation protocol and make it a separate snippet. This snippet can then be inserted in any protocol to compare its performance with or without distillation.

For the distillation protocol both nodes A and B require two qubits as input, where the two qubits of A should be entangled to the two qubits of B. They can then run different distillation protocols, to obtain one entangled pair with

higher fidelity. The distillation protocols from section 1.6 are implemented: the EPL-distillation as in algorithm 1 and DEJMPS-distillation from algorithm 2. In case the distillation results in failure, both nodes throw away their qubits and schedule an event that two new pairs should be put into the distillation protocol. The protocol continues like this until successful distillation, in which case it schedules an output event indicating that the distilled state can be used for an application.