# *Simulation of Quantum Algorithms Over Distributed Network Systems*

A Thesis

Presented to the

School of Interdisciplinary Informatics

and the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfillment

of the Requirements for the Degree

Masters of Science

University of Nebraska at Omaha

by

J. Joel vanBrandwijk

April 2016

Supervisory Committee:

Dr. Abhishek Parakh

Dr. William Mahoney

Dr. Mahadevan Subramaniam

## ***Abstract***

*Simulation of Quantum Algorithms Over Distributed Network Systems*

J. Joel vanBrandwijk, MS

University of Nebraska, 2016

Advisor: Dr. Abhishek Parakh

For students and researchers alike, the field of quantum computing is difficult to approach. The nature of quantum computing runs counter to the intuitive framework applied to most other domains in the computer sciences. Specialized quantum computing systems are not readily available if they exist at all, making kinesthetic learning and experimentation difficult tasks.

The difficulty in providing tangible demonstrations of quantum computers and algorithms provides the motivation for this thesis. In this project we developed a tool for simulating quantum algorithms, including entanglement based algorithms. The resulting tool aims to empower the learner, stimulate research, and enable experimentation.

The tool is capable of simulating quantum algorithms across distributed network systems. Included in this requirement are demonstrations of quantum teleportation as well as cryptographic algorithms using unentangled and entangled qubits. This tool can be used to simulate various quantum cryptographic algorithms and thus verify calculations regarding their comparative efficiency and security.

The theoretical contributions of this thesis work include a new method for implementing entanglement in a completely distributed manner and enabling operations and measurements on them. No other existing simulator provides this.

## *Dedication*

To my Grandfather, Harold Heidrick, whom I never met but who sought to understand his universe, and to my daughter, Osa Rose, that she will keep a sense of wonder.

## *Author's Acknowledgement*

Firstly, to Nicole, without whom little of what I do would be possible. You have been patient and gracious through my long hours of work and studying, and removed more worries than I think even you know.

To all the faculty in the Information Assurance department, from whom I have learned much. Especially to Dr. Parakh, who has guided me as we explored deeper into the subject of this work, and given to me a great wealth of knowledge about secrets hidden from most of the world.

To my mother and father, who encouraged my learning and always pushed me to ask more questions, and who gave me access to books and teachers when I asked what they couldn't answer.

# *Table of Contents*

# *List of Figures*

# *List of Tables*

# Chapter 1

# Introduction

Quantum computing refers to the use of quantum mechanics to construct a circuit or machine which transforms or transmits data [2]. The field of quantum computation holds many exciting promises for the field of information assurance. Shor's algorithm can break current state of the art cryptography [3]. But "What quantum computing takes with one hand, it gives back with the other" [1] . Many quantum algorithms have been developed to provide unbreakable encryption based on the laws of physics [3] [4]. Although much research has been done on quantum communication protocols, we are still, in terms of sophistication and scale, at the very beginning of implementing practical quantum networks.

At it's base, quantum computing relies on a rudimentary understanding of quantum mechanical principles such as superposition, no-cloning, measurement, and entanglement. However, these topics run counter to classical knowledge about what a computing machine should be and how it should behave. Where classical models describe information as a series of discrete bits in zeros and ones, quantum computing instead has a single quantum bit, or qubit, represented by probabilities of zero or one - while existing as both until measured. In classical models, bits can be copied by reading and writing to memory, but with qubits, reading the quantum state is a violation of the laws of physics; the state is collapsed by measurement, thus losing the quantum information.

In this thesis, we will first attempt to demystify some of the basics of quantum computing, including methods for representing quantum bits, implications of measurement and no-cloning, and entanglement in section 2.1. A brief discussion of some of the mechanisms available as physical implementations for quantum computers follows in section 2.2. Section 3.1 discusses three widely known quantum encryption protocols, as well as the concept of teleportation, which is useful for wide area quantum networks. In section 3.2, we will examine some of the common causes for error in quantum systems along with methods for detecting and correcting integrity problems. In this section, we have contributed research on the subject of rotational error detection and correction.

The remaining chapters of the thesis are original work related to the theory, design, implementation, and practical use of a simulator for quantum algorithms across distributed network systems. One of the key problems with simulation of quantum communication protocols is addressed in section 4.1.2, along with our novel solution. Section 4.2 discusses the technical aspects of the QooSim system, including comparison to previous works as well as a brief history of the evolution of the system. Results and analysis from simulations run in QooSim are provided in section 4.3, and this thesis concludes in chapter 5. Source code from some of the “iRunnable” objects which implement quantum applications are provided in Appendix A.

# Chapter 2

# Quantum Information Representation

## *2.1 Quantum Fundamentals*

Before one can meaningfully describe and investigate the simulation of quantum information processing, an understanding of some of the underlying quantum principles is needed. The most basic unit of quantum computation is the quantum bit or qubit. In classical computing, information is represented as a series of bits valued as either a logical zero or one. These are discrete and deterministic values: an ON or OFF switch. Quantum computing uses a representation of information similar to the classical bit, but suspends the restrictions of discrete values. Quantum mechanics allows that the universe is not a deterministic neighborhood, and that discrete values are mere observations of continuous phenomena. Quantum computing applies quantum mechanics to information processing. Just as with classical bits, quantum bits may have values of zero and one. However, quantum bits may also have a linear combination, or "superposition" of states.

### *2.1.1 Representing Qubits*

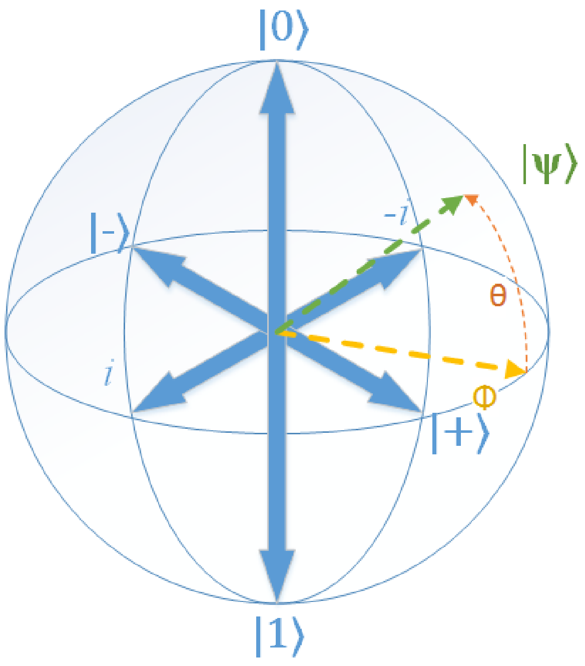

*Figure 2.1: Bloch sphere representation of a qubit.*

Qubits can be manifested through a number of different physical phenomena, some of which are described in section 2.2. However, it is useful to treat qubits as abstract mathematical phenomena, to avoid dependence on a physical implementation [24]. Single qubits are often envisioned as a point on the surface of a unit sphere, known as a Bloch sphere. The logical values for zero and one, written as $|0\rangle$ and $|1\rangle$, are pictured at the top and bottom pole of the z-axis on the sphere in figure 2.1, respectively. The poles of the x-axis are labeled as $|+\rangle$ or $|-\rangle$, where as the poles of the y-axis represent the phase of the qubit, labeled as $i$ and $-i$. Information is encoded onto a qubit through the process of rotation. In figure 2.1, the angle $\theta$ projects the amplitude, while the angle given by $\phi$ projects the phase. The superposition of the $|0\rangle$ and $|1\rangle$ states is described by the vector $|\psi\rangle$. The Bloch sphere is a useful tool for representing the state of a single qubit and visualizing operations performed on that state. However, in the three-dimensional world, there does not exist an intuitive representation for multiple qubits in non-trivial states [24].

When viewed on the Bloch sphere, the z-axis is referred to as the "computational" or "rectilinear" basis. This basis equates to performing a measurement in the vertical direction; or without rotation before measurement in the simulator. The x-axis is referred to as the "diagonal" basis. The east and west poles of this basis are referred to as $|+\rangle$ and $|-\rangle$, respectively. The rectilinear and diagonal basis can be exchanged by a rotation of $\pm\pi/2$ about the y-axis on the bloch sphere. A qubit at a pole position in either of these bases is in a superposition with respect to the other two bases. This is a critical component of the BB84 cryptographic protocol discussed in section 3.1.1.

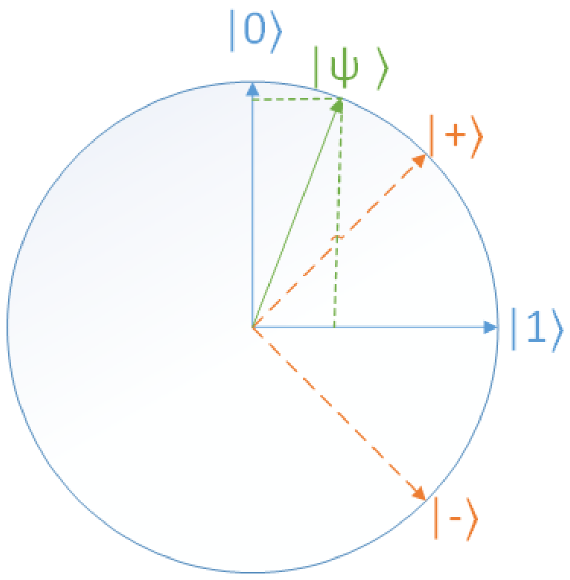

*Figure 2.2: Bloch circle representation of a qubit.*

Stemming from the Bloch sphere method, a two-dimensional visualization is available in the form shown in figure 2.2. In this form, the plane intersecting the z- and x- axes, which are the computational and diagonal bases, respectively, is taken to represent the whole system. Note that in this representation, angles are presented at half the degree at which they would be on the sphere, owing to the flattening computation. The rectilinear and diagonal bases can be exchanged by a rotation of $\pm\pi/4$ on the Bloch circle [20]. The utility of the circular representation is that the "shadow" of a qubit's vector on the circle correlates to its probability. That is, the cos and sin of the angle can be visualized directly.

By convention, the probability of a qubit being measured in the state zero is given as $|\alpha|^2$, where the complex number $\alpha = cos(\theta/2)$. Similarly, the probability of one as the outcome is given as $|\beta|^2$ where $\beta = sin(\theta/2)$. Using the Bloch sphere model, since $sin^2\theta + cos^2\theta = 1$ by the trigonometric identity, and the radius of the sphere is 1, the point which represents the qubit in a pure state is always located on the surface of the sphere. In section 3.2, error mechanisms will be introduced which distort the shape of the sphere. There also exist what are known as "mixed" states for qubits, in which the qubit would be at some point inside the sphere. However, discussion of such states is outside the scope of this work, and the associated simulator does not support these states. In fact, when a quantum system falls into a mixed state, the simulator will re-normalize the associated vectors to return it to a pure state.

Most often, phase is ignored during calculations. This is because phase does not impact measurement outcomes. Consider the case $\frac{1}{\sqrt{2}} = \alpha = \beta$. Then the probability, $P_r$ of the states $|0\rangle$ and $|1\rangle$ are $P_r(|0\rangle) = |\alpha|^2 = 1/2 = |\beta|^2 = P_r(|1\rangle)$. There are some conditions in which phase is relevant, most notably in the case of superdense coding. However, for the remainder of this thesis, phase will be ignored unless specifically noted.

A qubit is most commonly represented using one of two equivalent methods, based on which is mathematically convenient. The first is known as "ket" notation, and emulates polynomial equations. In ket notation, each possible state, $|q_i\rangle$, is associated with an amplitude, $\sqrt{P_r(|q_i\rangle)}$. For the generic system $Q = \sum \sqrt{P_r(|q_i\rangle)}|q_i\rangle$ where $P_r(|q_i\rangle)$ is the probability for the state $|q_i\rangle$, as in equations 2.1. The convention is to use greek letters $\alpha$ and $\beta$ for the probability amplitudes of the states $|0\rangle$ and $|1\rangle$.

As seen in equation 2.2, more than one qubit can be represented by a single "ket" notation equation. This is typically done by enumerating all possible states with associated greek lettering. Ket notation is useful for shorthand notation of states, as in equation 2.3, but has limited utility when applying gate operations to qubits.

$$|Q_A\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.1}$$

$$|Q_{AB}\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle \tag{2.2}$$

$$|Q_{ABCD}\rangle = \alpha|0000\rangle + \beta|1111\rangle \tag{2.3}$$

The second, more formal method for representing a qubit is through matrix algebra. This representation is useful for calculation and proofs, more naturally accommodating gate operations. The reason for the utility of this method is that quantum logic gates and quantum channel errors can be represented as matrices as well. For example, rotation can be represented by the product of a two-by-two rotational gate matrix and the one-by-two matrix representing the qubit. Equations 2.4 and 2.5 demonstrate the application of rotation on a single qubit using matrix algebra.

$$R_y(\theta)|Q\rangle = \begin{pmatrix} cos(\theta/2) & -sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{2.4}$$

Thus, for $\theta = \pi/2, |Q\rangle = |0\rangle$

$$\begin{pmatrix} cos(\theta/2) & -sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \tag{2.5}$$

Matrix representation is a powerful tool in describing quantum systems and operations. It can be used not only to describe singular qubits, but quantum systems in multiple degrees of entanglement, quantum operators which act on these entangled systems, and the error inducing phenomena which occur in quantum systems. For this reason, quantum operations in the QooSim simulator are almost exclusively based around matrix algebra and the matrix representation. The matrix representations of some of the more common gates are listed in table 2.1.

| Symbol | Matrix | Function |
|---|---|---|
| $\sigma_x$ | $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ | Swap amplitudes (quantum NOT gate) |
| $\sigma_y$ | $\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$ | Swap amplitudes and phases |
| $\sigma_z$ | $\begin{pmatrix} 1 & i \\ 0 & -1 \end{pmatrix}$ | Swap phases |
| $H$ | $\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$ | Hadamard gate - switching basis. |
| $R_x(\theta)$ | $\begin{pmatrix} cos(\theta/2) & -isin(\theta/2) \\ -isin(\theta/2) & cos(\theta/2) \end{pmatrix}$ | Rotate about the x axis |
| $R_y(\theta)$ | $\begin{pmatrix} cos(\theta/2) & -sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{pmatrix}$ | Rotate about the y axis |
| $R_z(\theta)$ | $\begin{pmatrix} cos(\theta/2) & 0 \\ 0 & cos(\theta/2) \end{pmatrix}$ | Rotate about the z axis |

Table 2.1: Quantum gate matrices.

### *2.1.2 Measurement & No-cloning*

The mechanisms in section 2.1.1 are useful for mathematical modeling quantum systems. One can initialize quantum bits, apply operations, and observe the probabilities of different outcomes. However, this is not true of physical quantum systems. Information maintained in qubit, once encoded, cannot be observed without altering the state of the qubit itself. Information can only be read from a quantum bit through a non-deterministic and irreversible process known as measurement.

In a single qubit system, measurement collapses the superposition state $\alpha|0\rangle + \beta|1\rangle$ to either of the states $|0\rangle$ or $|1\rangle$. The act of measuring the qubit destroys the non-deterministic information which was contained within the quantum state. It is impossible to know to which state the qubit will collapse before measurement [23]. This uncertainty property provides much of the basis for the security of protocols described in sections 3.1.1, 3.1.2, and 3.1.3. These protocols rely on uncertainty as well as the destructive properties of measurement operations to provide confidentiality both by obscuring the secret quantum information and by providing a mechanism to detect eavesdroppers.

A clever adversary with infinite resources might at this point speculate that these protocols could easily be defeated by making a statistically significant number of copies of the qubits, measuring all of them, and extrapolating an approximation of the original quantum state from the results. However, it is not possible to create an identical copy of an arbitrary unknown quantum state. In fact, the effects of cloning a system by means of measurement is that the system will collapse to a deterministic state. There will simply be more copies of the classical, deterministic state; but always exactly the same state [6]. Thus, quantum systems can be thought of as conserving state in the same way that mass and energy are conserved in a classical sense.

The no-cloning theorem has several important implications for quantum communication. The first is that since quantum states cannot be cloned or inspected, any eavesdropper will necessarily have to measure the state, thus disrupting the non-determinism of the state and creating an error with some given probability. Second, since quantum states cannot be copied, our traditional mechanisms for signal amplification by "repeating" will not be directly applicable in the quantum realm. Third, classical mechanisms for error correction, such as syndrome decoding and checksums, cannot be directly applied to quantum channels.

### *2.1.3 Entanglement*

The commonly referred to "Copenhagen interpretation" is a collective name for the principles guiding modern understanding of quantum mechanics [21]. The term is somewhat murky, owing to its origins as a sort of philosophical view of quantum mechanics espoused by Bohr, Heisenberg, and others; who did not necessarily always agree on the finer points [21]. For the purposes of this thesis, when we refer to the Copenhagen interpretation, we are describing systems in which exist in a superposition of states until measurement, in which measurement collapses the quantum state. For our purposes, the matrix representations given earlier will be sufficient to describe any quantum systems.

Simulation of entanglement across distributed systems is one of the problems at the heart of QooSim. This phenomena has no classical equivalent, and is so strange Einstein called it "spooky action at a distance" [7]. Put most simply, once two particles are entangled, there will be a correlation between their measurements. The fascinating thing about entanglement is that particles, once entangled, stay entangled until, according to the Copenhagen interpretation, they are acted upon by a non-entangling operation, such as noise or measurement. This holds true no matter the distance separating them. If we were able to send half of an entangled pair to Pluto over a noise-free quantum channel, the entanglement would still hold despite the distance. Moreover, local actions on one-half of the entangled pair will have an effect on the entire system. And yet, strange though it is, entanglement has been experimentally verified time and again, most recently by astronauts on the International Space Station [8].

This unbelievable property is perhaps not as spooky as it might seem. If one considers a pair of gloves and a pair of glove shaped boxes, the gloves, once placed inside their respective boxes, are indistinguishable. However, once opening a box and finding the left-hand glove, one immediately knows that the other box must contain a right-hand glove. This analogy should not be taken to imply that entangled qubits somehow contain secret variables about each other, however.

The Einstein-Podolsky-Rosen (EPR) thought experiment proposed just such an arrangement in an article entitled "Can Quantum-Mechanical Description of Physical Reality be Considered Complete?" [24] The crux of the EPR argument was that the Copenhagen interpretation was incomplete. Given two quantum states, say $A$ and $B$, if each state is equally balanced to either $|0\rangle$ or $|1\rangle$ outcomes, any of four combinations of results should be possible. EPR argued that in the case of entanglement, a measurement of $A$ as $|0\rangle$ always correlates with a value of $|1\rangle$ when $B$ is measured implies some hidden variable governing the related outcomes [24].

Some thirty years later, John Bell proposed an experiment which would put EPR to the test. The experiment supposed that the hidden variable theory implied by EPR was true, then a certain inequality about the observed measured values of the two qubits should hold. However, the inequality does not hold, and has been shown to be incorrect experimentally [22, 23]. The fundamental problem with EPR's view of quantum mechanics with respect to entanglement was a failure to consider the two qubits as parts of a whole larger system. It is not necessary that $A$'s result is instantaneously transmitted to $B$, because the entire system, which includes $A$ and $B$ are governed as one. All that is necessary is to form a representation of the system which describes these restrictions, shown below as Bell's states [18].

Copenhagen entanglement turns out not only to be true, but to play a central role in the effort to make practical quantum networks and protocols as well. As discussed in section 3.2, quantum signals degrade as they travel through a medium. Entanglement has been shown to have the potential to correct errors in which the bit values or phases are flipped [12] [13] [17]. Our current research explores the effects of a similar entanglement based error correction mechanism on partial rotational errors [64]. A “dual-rail” system has been proposed using entangled qubits to solve amplitude damping errors [14]. Entanglement has been theorized as the mechanism behind quantum “repeaters”, which promise to solve the problem of amplifying a signal which cannot be read without being destroyed [15] [16].

Quantum entanglement can be used to improve non-entangled quantum encryption schemes such as BB84 [9, 72, 74], or to develop new encryption schemes, such as SARG04 [19]. Superdense coding, which allows for more efficient data transfer and quantum teleportation, are both possible because of quantum entanglement [10, 11].

An entangled state of two qubits has the special condition that there exists no tensor product of pure single qubit states capable of generating the entangled state [23]. To describe entanglement in another way, there is no way to factor an entangled state into composite pure states. In terms of the equation 2.2, at least one of the state amplitudes must be zero, but no two amplitudes may be zero in such a way that the value of one of the pair of entangled qubits is determined.

This rule can be restated as follows:

$$\text{Let } A = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$$

Then if $A$ satisfies the condition that the sum of every row and column is not equal to zero, $A$ is an entangled state. This has important implications for our simulator, in which it is desirable to distinguish between entangled and unentangled states. Indeed, by extending this $2 \times 2$ representation of bipartite entanglement, we can represent $n$-party entanglement to a $2 \times n^2/2$ representation and “factor” unentangled qubits out of the system, as shown in section 4.2.6.

A special subset of entangled states are the Bell states [23]. The Bell states

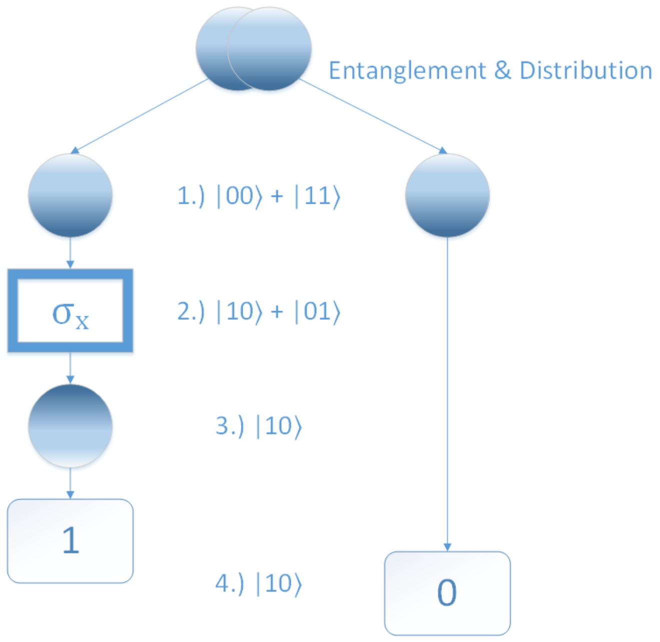

*Figure 2.3: Behavior of entangled qubits.*

represent the four maximally entangled states of two qubits. These four states are:

$$\frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle)$$

$$\frac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle)$$

That is, when one of the qubits in the pair is measured, the value of the other member of the pair, if measured in the same basis, is certain [24]. An illustration of this phenomena is shown in figure 2.3.

1. A pair of qubits is, $P$ and $Q$ are entangled in such a way that if one qubit is measured to be 0, the state of the system collapses to 00. Likewise for the measurement of 1, the collapse will be 11. There is an equal chance of either outcome from the measurement. The system is in the state $(\frac{1}{\sqrt{2}})|00\rangle + (\frac{1}{\sqrt{2}})|11\rangle$.

2. A $\sigma_x$ gate is applied to $P$, changing the state of the system so that if $P$ is measured to be 0, the state of the system collapses to 01. If $P$ is measured to

be 1, the state of the system collapses to 10. Likewise for $Q$. The system is in the state $(\frac{1}{\sqrt{2}})|10\rangle + (\frac{1}{\sqrt{2}})|01\rangle$.

3. A measurement is performed on $P$, resulting in 1. The system collapses to 10.

4. As a result, if measured in the same basis, $Q$ will be measured as 0.

In the matrix representation of quantum states, the mathematical method of generating an entangled state is to take two qubits in non-entangled initial states, say $|q_1\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ and $|q_2\rangle = |0\rangle$ and applying the controlled-NOT, or $C_{NOT}$ gate to the qubits.

$C_{NOT}$ is the quantum equivalent of XOR. It takes two parameters, the control qubit and the input qubit, and stores the result in the input qubit. If the control bit is zero, the input bit will not be changed. If the control bit is one, the input bit will be flipped. The matrix representation of the $C_{NOT}$ gate is given by equation 2.6. Refer to equation 2.2 for the probability amplitude coefficients for this state. Here we see that $\gamma$ and $\delta$ have been swapped. This fits with our logical expression of $C_{NOT}$, as in the first two states, $|00\rangle$ and $|01\rangle$, the control bit is zero, so nothing happens. However, in the remaining two states, $|10\rangle$ and $|11\rangle$, the control bit is one, so the second bit is inverted, and thus, the probability amplitudes of the states are swapped [23, 24].

$$C_{NOT}(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, |0\rangle) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} \tag{2.6}$$

$C_{NOT}$ is a unitary matrix, and is its own inverse; $C_{NOT}(|q_1\rangle, C_{NOT}(|q_1\rangle, |q_2\rangle)) = |q_1 q_2\rangle$. As a consequence, $C_{NOT}$ can be used to entangle and disentangle pairs of qubits. This result will be important for understanding error correction codes, quantum repeaters, and teleportation.

## *2.2 Physical Implementations*

In classical information processing, bits are stored, processed, and transmitted using a variety of physical systems: radio waves, photons, electrical pulses, magnetic orientation, optical media, and hole punches to name a few. Each mechanism has a preferred use case. Radio waves are used for wireless transmission to lighten infrastructure costs, but cannot transmit data as rapidly as fiber optic cables. Optical media is more stable for long term storage, but not as dense as magnetic disks. So, too, can qubits be stored in many different ways, each with advantages or disadvantages depending on the environment and intended use [24, 25]. All that is required for a physical implementation of a qubit system is a pair of physical states, an uncertainty preventing direct observation of those states, the ability to encode information by manipulating the system, and the ability to collapse the system to one physical state or the other for measurement [23–25].

The uncertainty of states is a special kind of condition referred to as "superposition." If we think along the lines of the Bloch sphere model, superposition does not mean that the qubit is oriented along a vector between the state vectors, but rather that it is oriented along both state vectors at the same time, but with different strengths or probability "amplitudes" [23]. However, for simplicity, the qubit is often thought of as exhibiting a rotation with respect to the axial state vectors. In this construct, what is actually being represented is the probability of a given state vector, not an actual vector between the two.

### *2.2.1 Ion Traps & Electron Spin*

The spin of subatomic particles can be thought of in terms of angular momentum. This angular momentum of a charged particle, in turn, creates a magnetic field as in figure 2.4 [24]. When observed, this magnetic orientation will reveal whether the electron is in the spin up state or the spin down state. However, until observation, the particle is in a superposition of both spin states. The terms $+1/2$ and $-1/2$ are used to indicate up and down spin of an electron, respectively, and correspond to the $|0\rangle$ and $|1\rangle$ quantum states [24] [25].

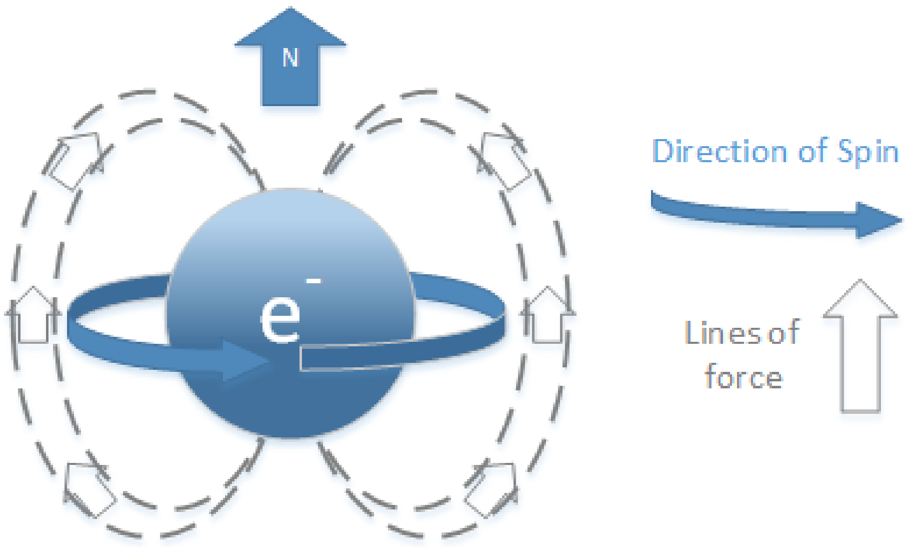

*Figure 2.4: Magnetic field created by electron spin.*

Both electrons and atomic nuclei exhibit spin. It would typically be difficult to isolate and measure these spins separately. However, in an ion trap, ions with a single outer shell electron are subjected to extreme cold temperatures. This causes the damping of almost kinetic momentum, and thus comparatively amplifies the electron spin and associated magnetic fields with respect to the kinetic nuclear spin [24,25]. To perform operations on electron qubits, a laser pulse at a specific frequency is applied to the ion. These pulses perform rotations of the electron, which can be combined to construct any quantum gate [24]. Measurement of a qubit represented by an electron is performed by use of another specialized laser, which will interact with only one of the possible states. If the qubit collapses to the required state, a photon will be emitted, which can be captured by a charge-coupled device. Thus, the capture of this photon and subsequent emission of a charge allows the measurement of the qubit to be registered as a “one” [24] [25].

Ion trap electron qubits are extremely useful as “quantum processing” qubits. Their lifetime is approximately one second. While this seems not to have much advantage over nuclear magnetic resonance presented in the next section, ion traps can perform more than $10^{13}$ operations in one second [27] [26]. However, the demanding environmental conditions and apparatus for manipulating and measuring electron qubits make it difficult to conceive of this mechanism being used for long distance communication. Even barring these conditions, there would, of course, be the additional challenge of developing a medium on which trapped ions could be sent [26] [24].

### *2.2.2 Nuclear Magnetic Resonance*

Nuclear Magnetic Resonance (NMR) operates on the same principles as ion trap electron spin quantum computing. However, in the case of NMR, the spin of a whole molecule is used to represent a quantum state. Molecules are prepared in an initial state by polarizing them in a magnetic field. Molecular spin is manipulated using radio frequency waves, a widely used technique in nuclear chemistry. Measurement is accomplished by reading the signal voltage of the magnetic spin [24].

Scientists had been hopeful that NMR would represent a mechanism to fulfill the long term storage role for quantum computing [27]. Lifetimes of greater than 3 hours were shown to be possible [28]. However, there are some fundamental problems with this representation. First, implementations of NMR use a matrix of millions of molecules to represent a single quantum state. This is necessary to support radio frequency operations [24] [28]. Thus, measurement is not the measurement of a single sub-atomic state, but rather the measurement of the average of millions of states. This has lead some scientists to question whether or not NMR implementations are truly quantum computers, or simply very sophisticated physical simulators [29].

Whether or not present implementations of NMR systems are quantum or classical computers will continue to be a subject of debate, research, and investigation. What is certain is that as the number of molecules used for NMR are reduced, so are the arguments against the "quantumness" of the system. There is therefore hope that NMR will still present the quantum solution to long term storage, once sufficiently advanced technology for manipulation and measurement of atomic nuclei arrives [29].

### *2.2.3 Photons*

The definition of a photon is something of a complicated matter in and of itself. For the purposes of quantum communication, it is most useful to think of photons as a unit of light represented in its wave-particle duality form. To create photons for use as qubits, the most common method is through the use of an attenuated laser. Attenuation tunes the laser to produce a beam of photons of the desired strength for use as initial state qubits. Information is represented on photons by using the polarization of the light to which they are associated. By convention, a horizontal polarization represents $|0\rangle$, while a vertical polarization represents $|1\rangle$, although the correlation of these values is arbitrary. Any two polarization angles may be used, as long as they are orthogonal [24].

Information can then be encoded on the photon through polarization. Polarization can be accomplished by using static polarizing sheets, similar to but more refined than the coating of polarized sunglass lenses. These sheets are mechanically manipulated as the photon beam is transmitted in order to encode information. More sophisticated systems use the Kerr effect, which alters the polarization of a material in the presence of electrical current. In these systems, the beam always passes through the Kerr material, and when a polarization is desired, an electrical charge is applied to the material [24].

Measurement in a photon based quantum system is done by using a photon detector. This can be as simple as a sort of camera which requires hundreds or

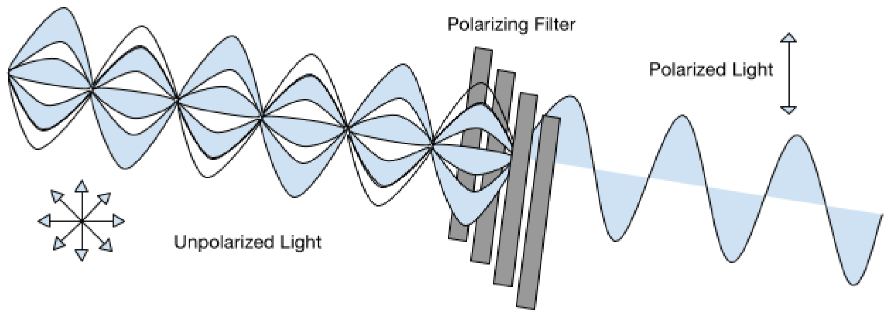

*Figure 2.5: Polarization of photons passing through a filter.*

thousands of photons in order to detect a bit arrival. Detectors can be as complex as an avalanche photon detectors (APD), which use the photoelectric effect to register a change in electrical state when a photon impacts an ion in the detector. APDs are extremely sensitive, and can detect a signal even with only a few photons present [33].

It should be noted that because neither system can perfectly detect the arrival of a single photon, channel noise notwithstanding, a qubit is not equivalent to a single photon, but rather the state of a qubit is encoded on a great number of photons at once. It is, of course, desirable to reduce the overall number of required photons in order to improve efficiency of the channel and reduce the likelihood of an undetected eavesdropper.

Qubits encoded in photons are most commonly sent using one of two mechanisms. Open-air transmission of photons is possible through the use of directionally aimed laser pulses. This mechanism requires that there be a line-of-sight between the sender and receiver. Open-air transmission has an advantage in that it does not require any infrastructure to connect two sites other than the equipment at the end nodes. Currently, on Earth, the longest continuous line of sight is around 500 km. While one might think to send photons to an orbiting space platform for communication, this too has a theoretical limitation of around 500 km due to signal loss caused by interaction with particles in the air. The longest open-air transmission to date has been 143 km, as reported by Ma, Xiao-Song, et al. [30].

Photons can also be used to transmit qubits across fiber optic lines. While existing fiber optic media may be used, as will be shown later, any non-quantum intermediary hardware, such as a router or a repeater, will actually serve to completely collapse the quantum state. Nonetheless, existing uninterrupted fiber channels may be used to send qubits encoded with photons. Photons experience a much greater loss rate over fiber optic channels than in open air. The noise generated by fiber optic channels currently limits effective transmission of cryptographic material to distances of 100 km or less [31]. At this distance, one percent of the transmitted photons will arrive. Channel noise is exponential; a rapid drop off occurs after this point, with only one in one million photons arriving to the 500 km point [32].

Photonic qubit quantum communication is already being used to experiment with quantum protocols, such as BB84, described in section 3.1.1. Created in 2003 at BBN labs, the DARPA Quantum Network is the first continuously operated quantum network. In 2004, nodes at Boston University and Harvard University were added. The network consists of ten nodes in A/B (i.e., Anna/Boris) pairs operating the BB84 protocol. "A" paris are key generators, while "B" pairs are key receivers. An "A" node can never perform key exchange with another "A" node, and similarly for "B" nodes [34, 35].

The fiber links connecting BBN to Harvard and Boston University (BU) are 10 and 19 km, respectively. The BU link suffers from high attenuation due to campus traffic, resulting in a complete inability to perform key exchange over this link. This link is still utilized in order to maintain continuous operation of the network, even though no useful data can be received. It is intended that in the future, upgrades to the fiber line will resolve the link failure. The remaining fiber links operate between 3.3 and 5 million pulses of photons per second. Key-generation rates of around 500 bits/s have been observed over inter-site lines, while intra-site links have been demonstrated at a key-rate of over 10 kbits/s [34, 35].

The Ali-Baba open-air channel is intended to cover 730 meters between two NIST buildings. The channel is expected to operate with an error rate of 3%, and will operate at an estimated 1.25 billion pulses per second. Key-rates of greater than 1mbits/s are thus anticipated [34, 35].

Encoding qubits on photons for communication has many practical advantages. Photons are cheap and easy to generate, requiring comparatively little in the way of specialized equipment. Photonic qubits exist at room temperature, and do not need to be held in a magnetic field. Additionally, because photons are not charged particles, they are relatively inert, and exhibit comparatively little interaction with their environment. Photonic qubits can be sent over existing fiber media, provided that they are not interrupted by classical repeaters or routers. Photons also have the distinct advantage of efficient transmission over open-air; the only wireless method of quantum communication known. The remainder of this thesis will be presented in the context of photonic qubits.

# Chapter 3

# Quantum Cryptography & Communication Protocols

## *3.1 Protocols*

In this section, some of the major quantum cryptographic & communication protocols and concepts are described. The purpose of this section is to provide an understanding of these protocols, which are later simulated in section 4.3. These protocols were selected because they are widely studied and understood, but a multitude of alternatives exist [63, 68].

### *3.1.1 BB84*

The BB84 Quantum Key Distribution (QKD) protocol is one of the oldest and most well explored quantum communication protocols. It is used as a proof-of-concept, benchmark and demonstration protocol in many experimental applications [34]. Since its original incarnation, a wide spectrum of variations [62,66,67] have been developed by those seeking to improve on the efficiency or practical security of the protocol [23, 24, 36].

The BB84 protocol is presented as a two-party key exchange system. The protocol assumes that the parties have a both quantum and classical channels between them. The classical channel is assumed to be insecure. The parties will exchange secret data by relying on the quantum channel. They will then use this data to generate a symmetric key for use over the classical channel. By convention, one party, Alice, generates qubits while another, Bob, measures them [23, 24, 36]. In our implementation, these are referred to as the generator and determiner parties, respectively. The protocol proceeds as follows:

1. Alice generates a pair of random set of bits, $A_{bases}$ and $A_{bits}$. Alice encodes $A_{bits}$ onto a set of qubits, $Q$ of equal length, initially valued at $|0\rangle$. For each member $A_{bases_i}$ of the set of bases, if $A_{bases_i}$ is one, Alice applies a Hadamard gate to $Q_i$. If $A_{bits_i}$ is one, Alice applies a $\sigma_x$ gate to $Q_i$. The resulting set of states is shown in figure 3.1 using the circular visualization. Alice saves her choices for bits and bases, then transmits $Q$ to Bob.

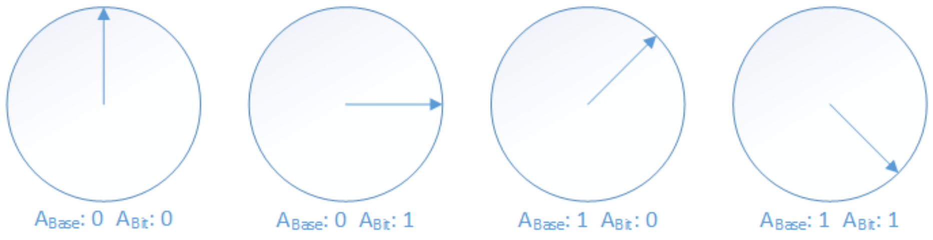

*Figure 3.1: BB84 encoding states.*

2. Bob generates a random set of bits, $B_{bases}$. Bob attempts to decode Alice's bits. If $B_{bases_i}$ is one, Bob applies a Hadamard gate to $Q_i$ and measures in the rectalinear basis. Bob then measures $Q_i$ and stores the result in $B_{bits}$. Now assuming no channel noise, if Alice and Bob have chosen the same base for position $i$, then $A_{bits_i}$ is equal to $B_{bits_i}$, that is, Alice and Bob have shared information. If they have chosen different bases, then the qubit at position $i$ will have randomly collapsed to a value of zero or one.

3. In order to determine which bits have been successfully shared, Bob transmits $B_{bases_i}$ to Alice over the insecure classical channel. Note that it does not matter at this point if an attacker can read the base string; the qubits have already passed through the channel, and the attacker could not have copied qubits for delayed measurement because of the no-cloning theorem.

4. Alice receives Bob's bases and compares them to hers. She sends a message over the classical channel informing Bob of which bases he has correctly guessed.

5. Alice and Bob now share a set of "raw" key bits for use in generating their shared key. They must now compare a subset of their bits in the clear in order to estimate channel error and potentially detect an eavesdropper. If the channel error rate is known, it can be compared to the actual error rate experienced. An error rate which is higher than expected may indicate that an eavesdropper has introduced additional errors by performing measurements on the quantum bits and replacing them on the channel.

6. If Alice and Bob determine that their error rate is acceptable, they move on to the process of eliminating errors in their shared bits through cascade error correction [60]. This process can take the form, for example, of exchanging checkbits of sufficiently large chunks of data. If the checkbits match, the data is kept. If they do not, Alice and Bob may discard the chunk or further subdivide it to determine where in the chunk the error exists. The size of these chunks of data will depend on the error rate of the channel.

7. After errors have been eliminated from the "raw" key bit stream, Alice and Bob apply a privacy amplification scheme. For our implementation, we chose to take a hash of the remaining "raw" key bits, and then use this hash in generating a symmetric key. The initial hashing serves two purposes. First, it takes the variable length string of remaining qubits to a fixed length string for use in

key creation. Second, it ensures that even if an eavesdropper had managed to capture a portion of the "raw" key bits, she will not be able to use that knowledge to decipher information encoded with the final key.

### *3.1.2 E91*

One drawback of the BB84 protocol is that in step 5, Alice and Bob must publicly disclose some of their key generating bit values in order to detect an eavesdropper. A similar protocol proposed by Arthur Ekert aims to avert this disclosure. Ekert's protocol (E91) is again a bipartite protocol, but his design uses three measurement bases on each party's system. E91 additionally has the requirement that the qubits used for key generation must initially be entangled in the state $\frac{|01\rangle+|10\rangle}{\sqrt{2}}$. These qubits may be generated by either party in the key exchange or some third party. The protocol remains secure even if the eavesdropper herself generates the entangled qubits [38] [37].

One party in the exchange elects to use the bases $\{-\pi/8, 0, \pi/8\}$, while the other uses $\{0, \pi/8, \pi/4\}$ (angles in the context of the Bloch circle). The parties share two bases, and will measure using the same basis 2/9 times; just under 1/4. When the parties measure along the same basis, the initial entangled state will ensure that they receive opposite results 100% of the time [38].

As with BB84, the parties publicly announce their bases. However, unlike BB84, instead of discarding the bits resulting from measuring in unmatched bases, they then publicly announce the bit values as well. Based on Bell's theorem, alluded to in 2.1.3, if the qubits measured were interrupted by an eavesdropper who measured them, entanglement would be broken. The two qubits would then exist as two locally distinct systems, as with EPR's view, and Bell's inequality would hold true. Therefore, if the parties calculate that Bell's inequality has been broken by a sufficient margin, they can assume that the qubits were entangled when received, and no eavesdropper is present [38] [37].

### *3.1.3 Kak06*

The two protocols described previously share many of the same mechanics. Qubits are exchanged and measured in randomly selected bases, the bases are then announced to identify the cases in which they agree. These processes, by their nature, are incapable of facilitating the exchange of anything but random information. While this restriction is fine for key exchange protocols, loss of large portions of data in a message would be unacceptable. Additionally, these protocols rely on a classical channel for the post-quantum measurement key negotiation. This, in turn, implies that the classical channel must be reasonably secure from tampering.

Subhash Kak designed a protocol which uses an entirely different paradigm, and is immune to the above mentioned restrictions. Kak's protocol can be used for both key negotiation, i.e., for symmetric key distribution, or directly for secure communication [39, 40]. The protocol, Kak06, uses only quantum channels for exchanging information. Unlike BB84 and E91, Kak06 is the quantum version of a classical protocol: Shamir's Three Pass Protocol [61, 73].

In Kak's protocol, each party chooses a secret quantum transformation. Returning to the conventional QKD parties, $U_A$ for Alice and $U_B$ for Bob. These transformations are unitary matrices, and must commute. The transformations may, for example, be rotational gates. Alice first encodes the information she wishes to send on a qubit. This could be either a classical zero or one state, or it could be some quantum state she wishes to send to Bob. Alice then applies her secret transformation, $U_A$ to the qubit. She then transmits the qubit to Bob, who applies his secret transformation, $U_B$, and returns the qubit to Alice. Since quantum gate operations are commutative, Alice then applies the inverse of her transformation, $U_A^\dagger$ to the qubit. Alice transmits the qubit back to Bob, who similarly applies $U_B^\dagger$ [39].

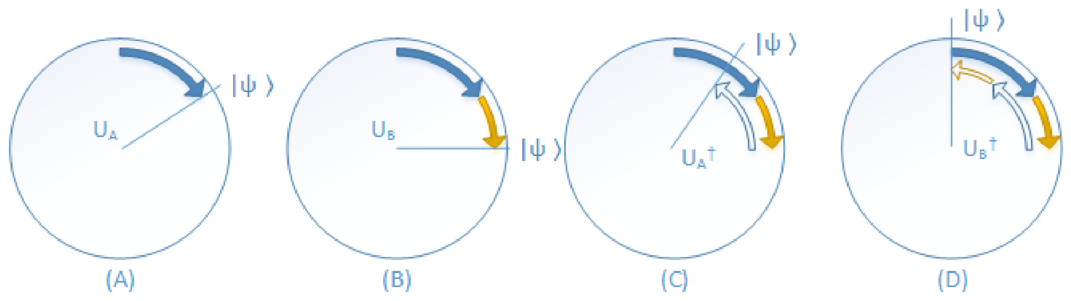

*Figure 3.2: Stages of encoding and decoding in Kak06.*

If the quantum channel has not been disturbed, the qubit Bob receives will be in exactly the state Alice encoded. However, if an eavesdropper has at any point intercepted and measured part of the transmission, the message will be garbled. Kak does not provide a method of detecting corruption or eavesdropping in his description of the protocol. However, one could presumably exchange checksum data as with classical information packets in order to validate data on receipt. These checkbits may certainly be encoded in the Kak06 protocol as well.

An item of interest about Kak's protocol is that the parties may change their secret transformations as often as they wish, and they may change them independently. The only requirement, of course, is that they keep knowledge of the transformations around long enough to invert them. This additional variance in the encoding done by the protocol provides an extra layer of security against a probabilistic eavesdropper [39].

### *3.1.4 Quantum Teleportation*

The motivation behind Kak's protocol was to send classical or quantum information using only quantum channels. Teleportation, conversely, allows for the transmission of quantum information using only two classical bits of information. Moreover, neither party is required to have knowledge of the internal state of the qubit. The key to this process is as in the E91 protoco and other quantum key distribution algorithms [70]l, a pre-sharing of entangled states.

Suppose that Alice and Bob each have one half of the state $|Q\rangle = |q_A q_B\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$, and Alice wants to transmit the arbitrary state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. The progression of the system is described by the equations in 3.1. Alice performs a $C_{NOT}$ operation on $|\psi\rangle$ and $|q_A\rangle$ with $|\psi\rangle$ as the control bit. Alice then applies a Hadamard gate to $|\psi\rangle$. She measures both of her qubits, which results in one of the systems of equations described by 3.2 [24, 25].

$$\begin{aligned} |Q\psi\rangle &= \alpha/\sqrt{2}|000\rangle + \beta/\sqrt{2}|001\rangle + \alpha/\sqrt{2}|110\rangle + \beta/\sqrt{2}|111\rangle \\ C_{NOT}(\psi, q_A)|Q\psi\rangle &= \alpha/\sqrt{2}|000\rangle + \beta/\sqrt{2}|101\rangle + \alpha/\sqrt{2}|110\rangle + \beta/\sqrt{2}|011\rangle \\ HC_{NOT}(\psi, q_A)|Q\psi\rangle &= \alpha/2|000\rangle + \alpha/2|001\rangle + \beta/2|010\rangle - \beta/2|011\rangle + \beta/2|100\rangle \\ &\quad -\beta/2|101\rangle + \alpha/2|110\rangle + \alpha/2|111\rangle \end{aligned} \tag{3.1}$$

$$\begin{aligned} q_A = 0, \psi = 0, |q_B\rangle &= \alpha|0> +\beta|1> \\ q_A = 0, \psi = 1, |q_B\rangle &= \alpha|0> -\beta|1> \\ q_A = 1, \psi = 0, |q_B\rangle &= \beta|0> +\alpha|1> \\ q_A = 1, \psi = 1, |q_B\rangle &= -\beta|0> +\alpha|1> \end{aligned} \tag{3.2}$$

Now, if Alice transmits her measured bits as classical information to Bob, Bob knows exactly which transformations to perform to align $|q_B\rangle$ to the initial state $|\psi\rangle$, given in table 3.1.

Teleportation is a fascinating consequence of entanglement. It appears to violate

| Results of $q_A$ and $\psi$ | Bob's operation |
|---|---|
| 00 | – |
| 01 | $\sigma_z$ |
| 10 | $\sigma_x$ |
| 11 | $\sigma_x \sigma_z$ |

Table 3.1: Teleportation Transformations.

the no-cloning theorem, in particular. However, it is important to realize that no information has been "copied" as a result of the teleportation process. Instead, in the end, one is left with a single qubit in the state described by $|\psi\rangle$, and two classical bits. One could not use teleportation to create an arbitrary number of qubits in the same state; only to transfer state from one qubit to another. This transference of state, however, is an exciting result in terms of quantum communication. Teleportation is the mechanism at the core of many proposals for quantum repeaters, described in the next section.

## *3.2 Error Detection & Correction*

Classical information systems are subject to a wide variety of hardware and software phenomenon which can compromise the integrity of data in storage, processing, and transmission. These errors present themselves in two groups: misread bits, where a zero is read as a one or vice-versa, and lost data, due to signal or current degradation. The former case is commonly resolved through the use of redundant bits or checksums. In the later case, data is often freshened through the use of a repeater (or, in terms of computer memory, refreshing). A repeater reads the signal off of an incoming line and rebroadcasts a fresh signal on an outgoing line. This has the general effect of amplifying the signal thereby counteracting signal attenuation.

The same types of errors present themselves in quantum information systems. However, the classical correction techniques are not directly applicable to the realm of quantum communication. In order to use a checksum, all data needs to be read and summed. Similarly, redundant bits need to be compared against each other to detect and correct any inconsistencies. Either of these processes would require measurements, and once the qubit is measured, all of the quantum information is lost. Likewise, classical repeater techniques would violate the no-cloning theorem prohibiting the copying of a qubit's internal state [24, 41].

Quantum systems are also subject to types of errors which have no classical analogy. Classical signal loss is similar to amplitude damping in qubits, but quantum systems also experience phase damping. A bit-flip error is represented as a rotation of the Bloch sphere about the y-axis. Quantum systems experience phase-flip errors, which are rotations about the z-axis [24,41]. In addition, it is not necessary that these rotations completely flip the quantum state. Arbitrary degrees of rotation errors are possible about any combination of axes [24, 47].

Quantum systems are necessarily interact with their environments. Each point of interaction carries with it a certain probability of error introduction. As time passes, the system evolves from one probabilistic state to another. In this evolution, the qubit is termed as passing through a channel. This channel represents a passage through space-time, and may or may not be associated with transmission. For example, a qubit at rest in an ion trap system is subject to an amplitude damping channel as the system attempts to reach temperature equilibrium with its environment [24] [43].

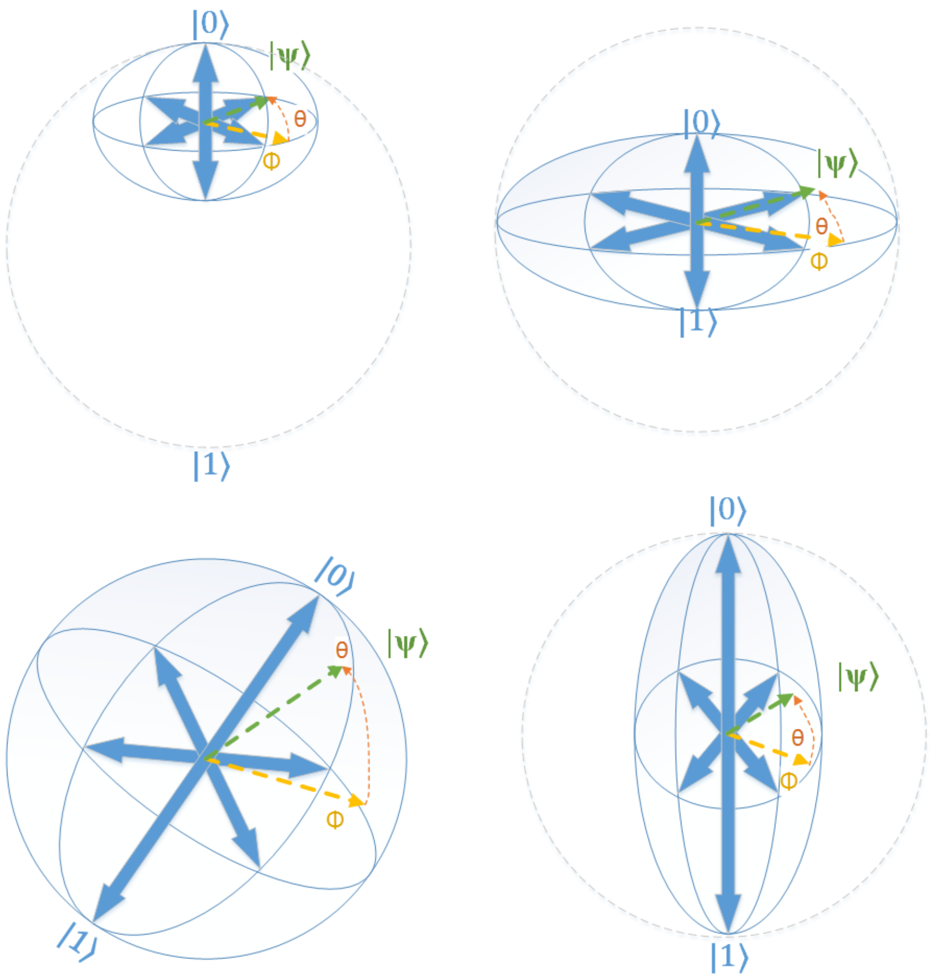

*Figure 3.3: Effects of Errors on the Bloch Sphere Representation of Qubits. (Clockwise from top left: amplitude damping, bit-flip, phase-flip, rotation).*

### *3.2.1 Kraus Operators*

Amplitude damping and phase damping quantum error channels are destructive processes, in which information is lost from the quantum system. As such, it is not invertible, and cannot therefore be represented by a unitary matrix, necessary for quantum gate operations. Instead, such processes are described by matrices in the form of Kraus operators.

The definition of Kraus operators requires another tool for representing quantum states: the density matrix $\rho$. The density matrix of a system is given by $\sum_i p_i|\psi_i\rangle\langle\psi_i|$, where $|\psi_i\rangle$ is some state in the system, $\langle\psi_i|$ is the transpose and conjugate of the state, and $p_i$ is the state's probability [24,45,46]. The utility of the density matrix is that it allows for the expression of a quantum state in a way that can be used in conjunction with a Kraus operator. Equation 3.3 shows the mathematics for generating the density matrix of the state $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = |+\rangle$ [45].

$$\rho = \sum_i^n p_i|\psi_i\rangle\langle\psi_i| = |+\rangle\langle+| = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \tag{3.3}$$

Kraus operators represent the probability of a given error being induced. These operators are generally presented as a system of matrices: $E_{0\ldots n}$, and applied to quantum states through the operator-sum method, $E(\rho) = \sum_i^n E_i\rho E_i^\dagger$, where $E_n$ is the matrix associated with the $n^{th}$ outcome occurring. For example, $E_0$ may be the operator for attenuation not occurring, and $E_1$ is the operator if it has.

### *3.2.2 Amplitude Damping*

In the Bloch sphere representation, a qubit affected by an amplitude damping channel, as shown in figure 3.3 (top-left) appears to "shrivel" towards the north pole, or $|0\rangle$

value. This behavior is an illustration of energy loss by the quantum system to the environment. Amplitude damping is a major factor affecting the stability of quantum systems, and part of the necessity for highly controlled environments in most physical implementations. For the purposes of this paper, amplitude damping is framed in terms of photonic qubits passing through a media; fiber or open-air. The transmission distance limits discussed in section 2.2.3 are due to attenuation caused by amplitude damping channel, associated with photonic interaction with fiber material or atoms in the air [24].

The Kraus operators for amplitude damping are given in equation 3.4, where $\eta$ is the probability of energy dissipation occurring. The Kraus operators are applied to a quantum state as shown for state $Q$ in equation 3.5 (ignoring phase) [24] [43] [42].

$$E_{AD_0} = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-\eta} \end{pmatrix}; E_{AD_1} = \begin{pmatrix} 0 & \sqrt{\eta}) \\ 0 & 0 \end{pmatrix} \tag{3.4}$$

$$E_{AD}(Q) = \sum E_{AD_i} Q E_{AD_i}^{\dagger} = \begin{pmatrix} 1-(1-\eta)(1-\alpha) \\ \beta(1-\eta) \end{pmatrix} \tag{3.5}$$

## Logical Means for Correction of Damping Errors

One classical method of reducing transmission errors is to use repeater nodes at specific intervals in the transmission line. Fiber optic runs will typically have a repeater node every 50 km or so. These repeaters absorb and interpret the signal received through the incoming photons and generate new photons to retransmit on down the line. Clearly, the no-cloning theorem, which prohibits the copying of one qubit state onto another qubit applies to this classical repeater situation. A photon absorbed by a classical repeater would collapse into a measured state. The new photon would be sent with a random polarization, and all of the information encoded upon the initial photon would be lost. Therefore, quantum communication is incompatible with classical hardware repeaters; classical and quantum communication cannot co-exist on these networks.

In order for a photonic quantum repeater to function, a special consequence of entanglement known as teleportation must be employed. Instead of either one of the nodes generating the entangled qubits, they will be generated by an intermediate node, with one qubit from each pair being sent in either direction. If the repeater nodes are separated by 100 km, a qubit sent between them will have an amplitude damping error rate associated with that distance, say $e_r$. However, if a pair of qubits are sent to each repeater node by a centrally located entanglement source equidistant from each repeater, the error rate will be $\sqrt{e_r}$ [51, 52].

Because of the properties of teleportation, neither node needs to know anything about the state of the qubits being repeated. One can envision a hierarchy or tree of these repeaters, spaced at regular intervals. A quantum state could be sent from one node to the other along a path of repeaters which are hierarchically related through entangled qubits. However, this scheme is not without drawbacks. Most obviously, a continuous stream of entangled qubits is required in order to send information between nodes. Quantum measurements introduce errors of their own. These errors will be multiplied by the measurements required at each hop in the repeater chain. Finally, entangled states cannot be sent using teleportation. This is a serious deficiency, and precludes many of the most important quantum communication protocols.

### *3.2.3 Bit- & Phase-Flip Errors*

While amplitude damping contracts the state of a qubit towards the $|0\rangle$ state, bit- and phase-flip error channels induce a probability of inverting either the amplitude or phase, respectively, of the quantum state. Visualized on the bloch sphere in figure 3.3, the effect of the flip channels is to squeeze the qubit along the z- or x-axis. A bit flip channel has the operators described in equation 3.6, and phase-flip errors are described in equation 3.7 [24] [43] [42].

$$E_{BF_0} = \begin{pmatrix} \sqrt{\eta} & 0 \\ 0 & \sqrt{\eta} \end{pmatrix} ; E_{BF_1} = \begin{pmatrix} 0 & \sqrt{1-\eta} \\ \sqrt{1-\eta} & 0 \end{pmatrix} \tag{3.6}$$

$$E_{PF_0} = \begin{pmatrix} \sqrt{\eta} & 0 \\ 0 & \sqrt{\eta} \end{pmatrix} ; E_{PF_1} = \begin{pmatrix} \sqrt{1-\eta} & 0 \\ 0 & -\sqrt{1-\eta} \end{pmatrix} \tag{3.7}$$

**Logical Means for Correction of Bit- & Phase-Flip Errors**

In classical systems, bit flip errors may be detected and corrected through the use of majority voting schemes. In these systems, redundant copies of bits are stored or transmitted. When they are read, the values of the bits are compared. If all bits are equal, then no error is presumed to have occurred. However, if one or more bits disagree, an error has occurred. These schemes work on the assumption that errors on the channel must be relatively rare. Thus, if a majority of the bits hold the same value, the dissenting bits are presumed to be in error.

The application of a majority voting scheme to quantum channels requires some careful maneuvering to get around the no-cloning theorem in creating redundant copies of bits as well as the destructive nature of measurement in evaluating those bits.

Suppose Alice wishes to send the quantum state $\alpha|0\rangle + \beta|1\rangle$ to Bob. Alice will first generate two additional qubits, both in the state $|0\rangle$ and entangle them with her state through use of the $C_{NOT}$ gate. The resulting state will be $\alpha|000\rangle + \beta|111\rangle$. Note that this operation has not copied the state of Alice's original qubit. A measurement of any qubit in the triplet will result in a definitive value for the other two qubits, and thus state is preserved [24] [43] [23].

Alice sends all of the bits in her triple to Bob. Assume that one of those qubits, say the last one, experiences a bit-flip error. Bob will receive the state $\alpha|001\rangle + \beta|110\rangle$.

Bob now prepares two ancillary qubits, $|a_1\rangle$ and $|a_2\rangle$, both in the state $|0\rangle$. He applies a $C_{NOT}$ gate to his ancillary qubit with the first qubit in the message as the control bit. The result is the four-party entangled state, $\alpha|0010\rangle + \beta|1101\rangle$ [24] [43] [23].

Bob next applies another $C_{NOT}$ gate to his ancillary qubit, this time with the second qubit as the control bit, resulting in $(\alpha|001\rangle + \beta|110\rangle)|0\rangle$. The ancillary qubit is then measured; in this case to be a value of zero. Because Bob sees a zero as the result of this measurement, he can deduce that the first and second qubits are identical. Assuming that errors are rare on the channel, no error has occurred on the first or second qubits [24] [43] [23].

Bob now applies the same procedure using his second ancillary qubit, $|a_2\rangle$ with the second and third message qubits as control bits, respectively. In this case, when the ancillary qubit is measured, it will result in a value of one. Bob knows that a value of one in this measurement means that the second and third qubits do not share the same value. Since he knows that the first and second qubits are in agreement, he can apply the majority voting principle to infer that a bit-flip error has occurred on the third qubit. Bob can now apply the $\sigma_x$ gate to correct the error [24] [43] [23].

A similar process is available to detect and correct phase flip errors. All that is needed is to apply the Hadamard gate to each of the message qubits before performing entanglement with the ancillary qubits. Error detection is accomplished in the same way, but now a $\sigma_z$ gate will be applied to correct the error. The encoding for phase-flip error correction is given by 3.8 [24] [43] [23].

$$\alpha\frac{|000\rangle + |111\rangle}{\sqrt{2}} + \beta\frac{|000\rangle - |111\rangle}{\sqrt{2}} \tag{3.8}$$

Bit- and phase-flip detection and correction can be combined into the nine-qubit system defined by Shor code. The initial qubit is entangled with two additional qubits, and all three qubits are passed through the Hadamard gate as with phase-flip detection. However, each qubit is then additionally entangled with two (for a total of six) qubits. The encoding for Shor code is given in equation 3.9 [24] [43] [23].

$$\alpha\frac{(|000\rangle + |111\rangle)^3}{2\sqrt{2}} + \beta\frac{(|000\rangle - |111\rangle)^3}{2\sqrt{2}} \qquad (3.9)$$
$$\text{Where } |Q\rangle^3 = |Q\rangle|Q\rangle|Q\rangle$$

Whether using protection against bit- or phase-flip errors, or the Shor code, a maximum of one arbitrary bit- or phase-flip error can be detected and corrected.

## *3.2.4 Rotational Errors*

Rotation errors occur when a photon is perturbed in such a way that its polarization is altered. This kind of error has the same effect as applying an additional polarization filter to the qubit. Indeed, this may be the result of an unintended defect in the photon's path [47, 64]. Satellite based quantum networks hold great promise for long distance quantum communication. However, satellite-to-ground communication may be affected by rotational errors due to the relative motion of the satellite and the ground station [48, 49].

For all the information available on bit- and phase-flip errors, there is surprisingly little literature about the nature and correction of rotational errors. Through our research into quantum error correction [64, 65], we found that using the three-bit bit-flip code presented in the section above, we were able not only to detect a single rotational error, but also correct it, simply through application of the code. In fact, while we were only able to correct a single rotational error, we were able to discern that a rotational error had occurred even if all three qubits had been affected.

In the case of a single qubit, suppose the qubits have been entangled as for bit-flip detection, and that a rotational error of $\omega$ about the y-axis has occurred on the second qubit during storage or transmission. Then the state of the qubits will be $\alpha(cos\ \omega)|000\rangle + \alpha(sin\ \omega)|010\rangle - \beta(\sin\ \omega)|101\rangle + \beta(\cos\ \omega)|111\rangle$. Note that the terms associated with cos $\omega$ correspond to the unaltered state, while the terms associated with sin $\omega$ correspond to a bit-flip error.

After applying the $C_{NOT}$ operations on the first ancillary qubit ($|a_1\rangle$) with the first and second message qubits as control qubits in turn, the resulting state is $\alpha(\cos\ \omega)|0000\rangle + \alpha(\sin\ \omega)|0101\rangle - \beta(\sin\ \omega)|1011\rangle + \beta(\cos\ \omega)|1110\rangle$. When the ancillary qubit is measured, one of two cases will result. If it is measured as zero, with probability $\cos^2\ \omega$, the state collapses to $\alpha|000\rangle + \beta|111\rangle$, and the system has been restored. If not, the system collapses to $\alpha(\sin\ \omega)|010\rangle - \beta(\sin\ \omega)|101\rangle$; which is a bit- and phase-flip error on a single qubit. This scenario, if expanded to a nine-qubit system, can then be resolved by the use of Shor's code.

If the same technique is repeated, supposing a rotational error on both the first and second qubits instead, the system progresses through the application of the first two $C_{NOT}$ gates through the first ancillary qubit as given by equation 3.10.

$$\alpha|000\rangle + \beta|111\rangle \xrightarrow{R(\omega)\otimes R(\omega)\otimes I}$$

$$\begin{aligned} &\alpha\cos^2\omega|000\rangle + \alpha\cos\omega\sin\omega|001\rangle \\ &+\alpha\cos\omega\sin\omega|010\rangle + \alpha\sin^2\omega|011\rangle \xrightarrow{C_{NOT}(q_1,a_1),C_{NOT}(q_2,a_1)} \\ &+\beta\sin^2\omega|100\rangle - \beta\cos\omega\sin\omega|101\rangle \\ &-\beta\cos\omega\sin\omega|110\rangle + \beta\cos^2\omega|111\rangle \end{aligned} \tag{3.10}$$

$$\begin{aligned} &\alpha\cos^2\omega|0000\rangle + \alpha\cos\omega\sin\omega|0010\rangle \\ &+\alpha\cos\omega\sin\omega|0101\rangle + \alpha\sin^2\omega|0111\rangle \\ &+\beta\sin^2\omega|1001\rangle - \beta\cos\omega\sin\omega|1011\rangle \\ &-\beta\cos\omega\sin\omega|1100\rangle + \beta\cos^2\omega|1110\rangle \end{aligned}$$

Note that the outer four terms are associated with the probability of the ancillary qubit being measured to be zero. If this is the case, then the system collapses to the equation shown in the first line of equation 3.11. The system progresses through the application of the second two $C_{NOT}$ gates through the second ancillary qubit. Similarly, the inner four terms are associated with the probability of the ancillary qubit being measured as one, and the system progresses as in equation 3.12 in this case.

$$a_1 := 0 \qquad \begin{array}{c} \alpha\cos\omega|000\rangle + \alpha\sin\omega|001\rangle \\ -\beta\sin\omega|110\rangle + \beta\cos\omega|111\rangle \end{array} \xrightarrow{C_{NOT}(q_2,a_2),C_{NOT}(q_3,a_2)}$$
$$\begin{array}{c} \alpha\cos\omega|0000\rangle + \alpha\sin\omega|0011\rangle \\ -\beta\sin\omega|1101\rangle + \beta\cos\omega|1110\rangle \end{array} \xrightarrow{Measure a_2} \tag{3.11}$$
$$\begin{array}{ll} a_2 := 0) & \alpha|000\rangle + \beta|111\rangle \\ a_2 := 1) & \alpha|001\rangle - \beta|110\rangle \end{array}$$

$$\begin{aligned} &\alpha\cos\omega|010\rangle + \alpha\sin\omega|011\rangle \\ &+\beta\sin\omega|100\rangle - \beta\cos\omega|101\rangle \end{aligned} \xrightarrow{C_{NOT}(q_2,a_2),C_{NOT}(q_3,a_2)}$$

$$a_1 := 1 \qquad \begin{aligned} &\alpha\cos\omega|0101\rangle + \alpha\sin\omega|0110\rangle \\ &+\beta\sin\omega|1000\rangle - \beta\cos\omega|1011\rangle \end{aligned} \xrightarrow{Measure a_2} \tag{3.12}$$

$$\begin{aligned} a_2 := 0) &\qquad \alpha|011\rangle + \beta|100\rangle \\ a_2 := 1) &\qquad \alpha|010\rangle - \beta|101\rangle \end{aligned}$$

What is most interesting about this procedure is that under certain circumstances, such as if $\omega$ is small, it has a high probability of correcting rotational errors on more than one qubit. If the receiving party should measure both of his ancillary qubits to be zero, he can be reasonably certain that no bit-flip or rotational errors have occurred; or that those rotational errors which did occur have been cleaned up. It remains to be seen if we can apply this research to a general error correction scheme by combination with Shor code or dual-rail systems.

It is clear that if an ancillary qubit, $a_1$ or $a_2$ is measured to be one, this scheme cannot correct the error on its own. While the combinations of ancillary qubit measurements are unique within cases affecting one, two, and three qubits, they are not globally unique across all three cases. Thus, it is not possible to tell whether a rotational error occurred on only one qubit or if three qubits had been affected. In short, our scheme allows for the "hidden" correction of rotational errors in cases where the ancillary qubits measure zero. In cases where they measure one, it can only be discerned that at least one error has occurred.

# Chapter 4

# Quantum Simulation

## *4.1 Asynchronous Operations on Entangled Qubits*

Among the more daunting challenges for simulating quantum information processing on distributed systems is the problem of entanglement. In the physical world, entangled qubits appear to be "aware" of their partner's states instantaneously, regardless of the distance between them. This property of entangled systems runs counter to classical ideas of physics and of information theory. In designing simulation of entanglement over distributed systems, several different schemes were examined to support entanglement, including a master"registry" service to hold global states, and a chatty"lock-and-update" (lock) protocol, where every operation is transmitted to all entangled nodes immediately in transactional manner.

Ultimately, both the registry and lock protocols were unsatisfactory. The registry required at least one entity in the communication, if not a dedicated party, to own all qubits in the system. A peer-to-peer design was thought to be more desireable, allowing a more fluid implementation of new protocols and scalability to several nodes. The lock protocol was a viable alternative, but with several drawbacks. First, the overhead involved in each operation would grow with each node. Thus, the total traffic, assuming all nodes generate constant traffic, would grow exponentially with each new participating node. Second, each node would be required to have full, up-to-date copies of all entangled states at all times. But each node only needs to know about the qubits which it holds, and how those qubits are impacted by their partners. Systemic omnipotence is not a requirement of the simulator, but of the lock design; and an expensive requirement at that.

In experimentation with entangled operations, we came to the idea of asynchronous operations. The idea, described in detail in section 4.1.2, is to allow the node holding a portion of an entangled system to continually operate on the qubits which make up that portion. A node only needs to update its peers when an operation either affects and entangled state in which those peers participate, such as a measurement, or when local operations such as $C_{NOT}$ add or remove qubits from a shared entanglement.

### *4.1.1 Linear Extension Operations*

Any operation on one qubit in an entangled pair state necessarily has an effect on the other qubit, as described in 2.1.3 and shown in figure 2.3. For unentangled qubits, gates operations are the product of the matrix representation of the gate ($G$) with the matrix representation of the qubit [54,55]. $G$ is a unitary matrix, with coefficients $a$ and $b$ such that $|a|^2 + |b|^2 = 1$. To operate on the generic pair state, $R$, the tensor product of $G$ and the identity matrix $I$ are used to map a $2 \times 2$ unitary matrix onto a $4 \times 4$ matrix [54, 55].
Given

$$R = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle \tag{4.1}$$

and

$$G = e^{i\varphi} \begin{pmatrix} a & b \\ -\bar{b} & \bar{a} \end{pmatrix} \tag{4.2}$$

$$(I \otimes G)R = e^{i\varphi} \begin{pmatrix} a & b & 0 & 0 \\ -\overline{b} & \overline{a} & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -\overline{b} & \overline{a} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix}$$

$$(I \otimes G)R = e^{i\varphi} \begin{pmatrix} \alpha a + \beta b \\ \alpha(-\overline{b}) + \beta(\overline{a}) \\ \gamma a + \delta b \\ \gamma(-\overline{b}) + \delta(\overline{a}) \end{pmatrix}$$

$$\begin{aligned} (I \otimes G)R = e^{i\varphi}&(\alpha a + \beta b)|00\rangle \\ &+ (\alpha(-\overline{b}) + \beta(\overline{a}))|01\rangle \\ &+ (\gamma a + \delta b)|10\rangle \\ &+ (\gamma(-\overline{b}) + \delta(\overline{a}))|11\rangle \end{aligned} \tag{4.3}$$

### *4.1.2 Asynchronous Operations*

When simulating quantum systems and algorithms, it is at present necessary to use classical hardware and methods. This includes simulation of entangled states distributed across multiple computer systems over a classical network. It is in many cases necessary to apply quantum gates to these entangled quantum states after they have been dispersed across physical systems.

The conventional method of applying the tensor product of the gate and the identity matrix becomes challenging when considering these conditions. However, we will show that there exists an equivalent asynchronous operation on a single qubit which can be used to simulate a series of synchronous operations on the system.

We start by describing the system for simulation, shown in figure 4.1. At the moment a pair of qubits $P, Q$ are entangled, each system $\aleph, \beth$, respectively must begin maintaining a history of its actions. Additionally, the simulator must maintain a method for each qubit to communicate with the other, such as a network address. Conventional, single qubit gates may be applied without limitation to each qubit in the pair individually for the duration of the entanglement. When entanglement is broken, such as by measurement, the simulators must reconcile the state of their portion of the entangled state.

If the qubit $P$ is measured, $\aleph$ then should alert $\beth$ of the measurement, and inform $\beth$ of all operations performed on $P$ since entanglement but before measurement. Conceptually, $\beth$ should "rewind" the operation history of $Q$, restoring the initial entangled state. $\beth$ will then update the amplitudes of $Q$ to reflect the outcome of the measurement, and then replay both the operational history of $P$ and $Q$ against $Q$. The result will provide the state of $Q$, disentangled, after the measurement of $P$. A measurement of $Q$ will now match observable real world outcomes. The following

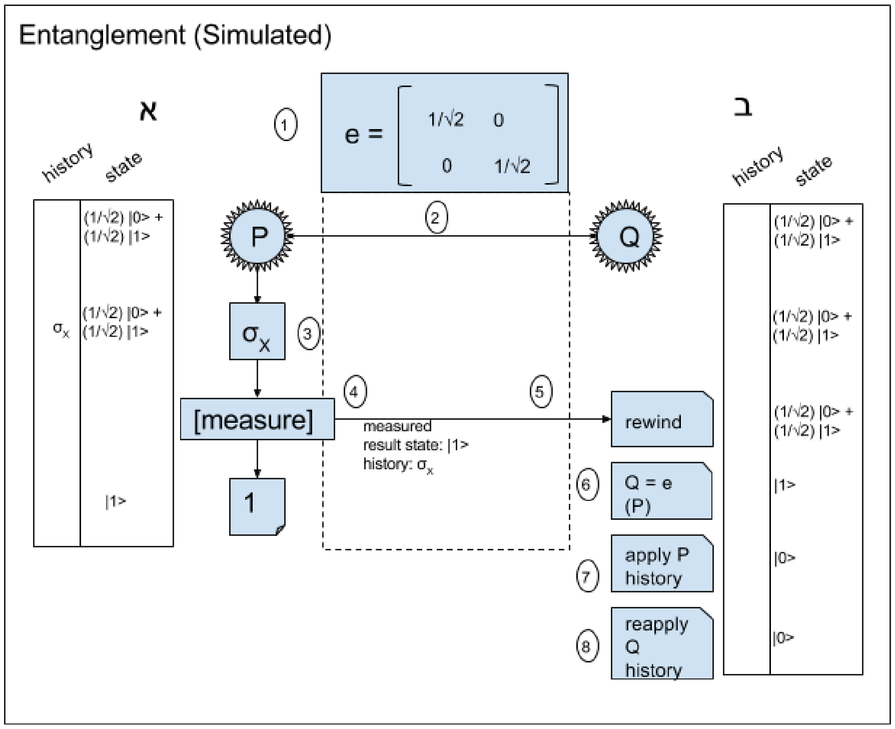

*Figure 4.1: Simulated entanglement through asynchronous operations.*

example interaction is shown in figure 4.1:

1. $P$ is set in the state $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$, and a $C_{NOT}$ gate is applied to entangle $Q$ into the same StateVector object. $Q$ is transmitted across a classical network.

2. A $\sigma_x$ gate is applied to $P$. $P$ processes the gate operation and stores the matrix corresponding to the gate in its history vector. Note that the gate operation has no discernable effect on the measurement outcome for the individual qubit represented by $Q$.

3. A measurement operation is applied to $P$. $P$ measures the qubit to be one. In the process of reducing the StateVector object, the system identifies $Q$ as a remote Qubit, and then notifies the network peer of the measurement results through the ChannelService.

4. The ChannelService notifies the remote state vector containing $Q$ that a measurement operation has taken place on $P$. Included in this notification are $P$'s result and history. $Q$ "rewinds" any operation which has been performed on itself by applying the inverse matrix operation.

5. $Q$ now applies $P$'s history, applying the $\sigma_x$ gate, which sets its state to $|0\rangle$.

6. $Q$ would now re-apply any operations which had taken place on it prior to notification from the entanglement object. No operations took place in this scenario. Note that if $Q$ were now measured, the result would be $|0\rangle$ with 1.0 probability.

We will now show by direct proof that performing asynchronous operations on a single qubit in the pair is equivalent to performing the same operation as a linear extension on the entangled pair.

Given (4.1) and (4.2), and assuming $R$ to be composed of the qubits $P = p_0|0\rangle + p_1|1\rangle$

and $Q = q_0|0\rangle + q_1|1\rangle$

$$\begin{aligned} P(G \otimes Q) = e^{i\varphi}(p_0|0\rangle((aq_0 + bq_1)|0\rangle \\ + ((-\bar{b})q_0 + (\bar{a})q_1)|1\rangle) \\ + p_1|1\rangle((aq_0 + bq_1)|0\rangle \\ + ((-\bar{b})q_0 + (\bar{a})q_1)|1\rangle)) \end{aligned}$$

$$\begin{aligned} P(G \otimes Q) = e^{i\varphi}((ap_0q_0 + bp_0q_1)|00\rangle \\ + ((-\bar{b})p_0q_0 + (\bar{a})p_0q_1)|01\rangle \\ + (ap_1q_0 + bp_1q_1)|10\rangle \\ + ((-\bar{b})p_1q_0 + (\bar{a})p_1q_1)|11\rangle) \end{aligned}$$

And now, since $p_0q_0 = \alpha$, $p_0q_1 = \beta$, $p_1q_0 = \gamma$, and $p_1q_1 = \delta$:

$$\begin{aligned} P(G \otimes Q) = e^{i\varphi}(\alpha a + \beta b)|00\rangle \\ + (\alpha(-\bar{b}) + \beta(\bar{a}))|01\rangle \\ + (\gamma a + \delta b)|10\rangle \\ + (\gamma(-\bar{b}) + \delta(\bar{a}))|11\rangle \end{aligned} \tag{4.4}$$

Since (4.3) is equivalent to (4.4), the operations on a single qubit are equivalent to the linear extension applied to the system.

It is worth noting that any of the paired state amplitudes, $\alpha ... \delta$ could be zero valued, thus encompassing the Bell pair states in this definition.

## *4.2 QooSim Implementation Details*

### *4.2.1 Previous Work*

In this section, we provide a survey of existing quantum simulators. We do so in an effort to show that while there are certainly very well constructed quantum computing libraries available, two features are rarely implemented: distributed simulation across a quantum channel and behavior of entangled qubits. We further demonstrate that the union of these features, simulation of the behavior of entangled qubits across distributed systems, has not, to our knowledge, been addressed. Such a feature would enable the simulation of entanglement based quantum key distribution protocols on classical hardware. We believe this to be an important tool for validating security and efficiency claims about such protocols.

#### **Libquantum**

Libquantum is one of the most widely used and cited [58] quantum computing libraries. Indeed, the preliminary work on our quantum library was to port libquantum to C++. However, as we will later show, our work has since evolved and bares little resemblance to structure of the original.

Libquantum is written in C, and received its last major update in 2013. The library is structured around quantum registers. Quantum registers contain a set of nodes. Each node corresponds in turn to a possible value for the register and the probability of that value occurring. The authors chose this design, presumably, for greater efficiency with quantum algorithms such as Shor's and Grover's. To wit, example implementations of both algorithms are included in the library. However, a consequence of this structure is that storage of quantum registers consumes system memory at an exponential rate. Indeed, we struggled to use a quantum register longer than 22 qubits on our development machines.

Libquantum contains a number of built-in gates and operations for manipulating quantum registers, including Hadamard, sigma and rotation gates. The library also supports controlled operations on pairs of qubits, such as swap and $C_{NOT}$ gates and on 3-tuples of qubits, such as the Toffoli gate. It does not, however, have facilities for sending these registers across a network or even for serialization of these structures. Together with a limited register size, it is difficult to simulate entanglement based quantum key distribution algorithms using libquantum across a network.

Our initial efforts to extend libquantum involved lexical serialization and reassembly of data structures into raw TCP data streams. Operations were then performed by sending specialized code-word tokens, again over raw TCP data streams. We had in essence begun to design an application layer protocol for libquantum. However, this effort quickly proved cumbersome and difficult to maintain and extend. The desire to avoid creation of a network based protocol exclusively to support our simulator became one of the design goals for our library. After all, serialization is a problem which has been solved many times over and besides not within the scope of our problem.

### Q++

Q++ is a somewhat later addition to the family of quantum simulators. As the name implies, the library is written in C++. The last update to this library was in 2013. The library is designed using a series of templates, allowing the user to construct one or more quantum “simulator” objects, each with one or more quantum “register” objects associated. The internal representation of quantum information is the same as for libquantum; a list of possible states is stored in correlation with the probability of each state occurring. Q++ has a feature for defining quantum operations, or QOPs, which are in turn a construction of one or more quantum gates, similar to a quantum circuit.

We were able to compile the library, but were limited in our ability to test this library. We suspect that Q++ would have the same memory consumption issues encountered by libquantum. Q++ does not have facilities for network transmission or serialization of quantum information. While the design of Q++ shows some exciting promise with simulator objects, we believe it would be difficult to simulate entanglement based quantum key distribution protocols using Q++. We did learn from this approach that creating a control object to manage registers would be useful to our approach. We incorporated this concept into our design through the“System” object.

**Quantum::Entanglement**

Quantum simulation is not limited to strongly typed, compiled languages. The Quantum::Entanglement module aims to “port some of the functionality of the universe into Perl” [59]. The module is unique in that it allows any set of states to be represented probabilistically, including floats, integers, strings, and objects. Any numerical probability can be assigned to these states. The library will self-normalize the probabilities. Quantum::Entanglement not only allows two quantum probabilistic values to be related, but allows for the “entanglement” of classical values onto an entangled state. In doing so, users are able to change the meaning of a quantum state on the fly, and create complex classical-quantum hybrid algorithms.

Being written in Perl, native serialization libraries would certainly be capable of encoding Quantum::Entanglement variables for transmission. Indeed, native protocols would be capable of transmitting these serialized variables. However, once transmitted, the quantum variables are decoupled. The library does not include a mechanism for simulating entangled states across distributed systems.

While Quantum::Entanglement has an interesting set of features and met our requirement for serialization of data structures, we chose not to pursue it as a basis for our simulator because of the weak variable binding and lack of native support for quantum gates, such as Sigma, Hadamard, and $C_{NOT}$ gates.

We examined three other quantum simulators during our research. QCL, jQuantum, and pyQu each take a different approach to the task of quantum simulation. It should be said that all of the tools we examined are commendable for their innovation and highly recommended to anyone wanting to explore more with quantum computing algorithms and concepts. We encourage readers to test out each of these libraries for themselves. Our research has a very specific goal and intention; that no existing tool could meet this goal should not diminish these works in the least.We have simply taken aim at a different aspect of the simulation problem.

### *4.2.2 QooSim*

Research into other simulators and experience with libquantum lead first to an attempt to extend the functionality of that library. The primary goal for the simulator was to enhance the practicality and usability of the library. This goal included the conversion of the library into an object oriented paradigm as well as the addition of support for quantum networks. The object oriented paradigms of encapsulation, information hiding, and interfaces are natural tools for representing quantum data. These tools provide the mechanisms to enforce the consequences of quantum mechanics, such as the no-cloning theorem and destructive measurement. Since the original libquantum library was written in C, one of the more obvious language choices for an object oriented port was C++.

In our preliminary research using libquantum, we had created a networked pair of binaries, representing the two sides of a BB84 exchange. All communication was through the raw C socket libraries. While this approach was effective enough for implementing a simple example of a single protocol, it was lacking in extensibility. When implementing a simulator, the network layer could have been left undefined, delegating this task to users of the library, but this hardly seemed to help the usability goal. A developer working in the library should already have the tools necessary to readily send a qubit from one system to another without having to worry about the implementation details.

It would have been possible to create a networking protocol for use in the simulator, but this seemed redundant. What the simulator needed was a way to serialize objects into a transmittable data format, a conduit for sending data to a remote system, and a method of unserializing that data on the remote system. This is a problem which has been solved many times over; several libraries exist for serializing data. We selected Google Protocol Buffers because of the native support for C++ along with several other languages, the ease of data definition, and the compatibility with a transmission mechanism in the Google Remote Procedure Call (gRPC) library.

### *4.2.3 Major Structures*

The QooSim simulator itself is composed of three major components. Each of these components is encapsulated in a C++ namespace or class. The Quantum namespace is responsible for the representation of Qubits. Measurement and gate operations also take place within this namespace. This component is sufficient to simulate quantum operations not requiring transmission across a network. The Quantum namespace went through three major revisions, described further below.

Networking, serialization, and transmission all fall within the QuantumChannel namespace. This component consists of a Google .proto file, which defines the serialized data. A pair of classes, ChannelService and ChannelService_client handle the receiving and sending of data along with the unserializing and serializing data, respectively. The final class, ChannelListener, is responsible for listening for network messages and passing them to the ChannelService class for processing.

The final component is the System class. This class manages an object implementing the iRunnable interface. Developers working with QooSim define a class implementing iRunnable to express their algorithm. The System class is responsible for starting a ChannelListener, and executing the iRunnable object's Run method. The System class is implemented using the singleton pattern. It additionally tracks quantum memory instantiated in the system and queues messages. When a Qubit is received by the ChannelService, the System will add it to the memory map, say at position N and also place a message in the queue to indicate that a Qubit has been received, and is available at position N in the map. The iRunnable class implementation should then periodically query the message queue to retrieve received Qubits for processing.

Figure 4.2 demonstrates the interactions between the major components of QooSim, in the context of receiving a QuantumMessage. In earlier iterations, a QuantumMessage was some form of register object; in current iterations, it is a Qubit object.

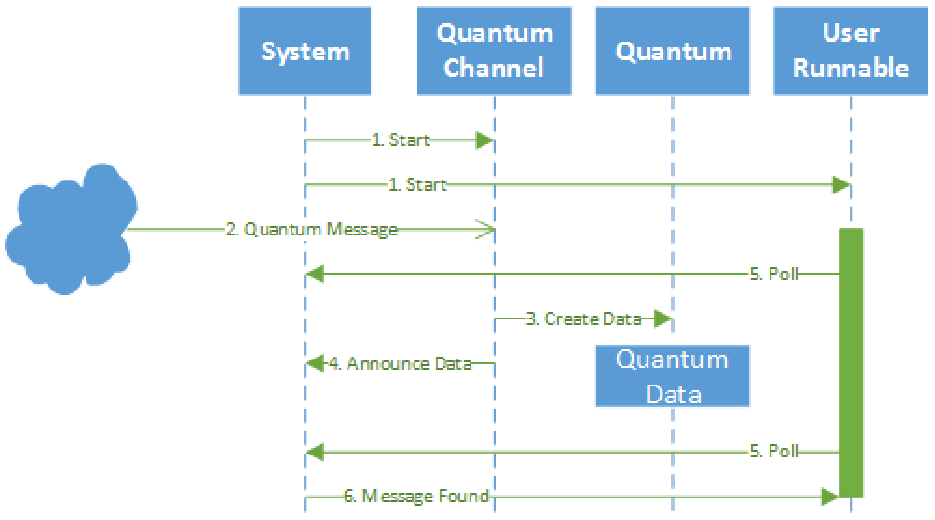

*Figure 4.2: Interaction between major components of QooSim.*

1. The system starts a thread for the ChannelListener object, which is bound to a ChannelService class, and starts a user runnable object

2. A quantum message is received from the network by the listener thread.

3. The ChannelService method associated to the quantum message type processes the message. Data is deserialized and objects are created as appropriate using classes in the Quantum namespace.

4. The ChannelService method adds a message in the System's message queue, announcing the creation of new data and providing a reference to that data

5. Meanwhile, the user runnable has been polling for messages.

6. If a message is found, the runnable is responsible for processing objects created as a result of that message.

### *4.2.4 Iteration 1 - Registers & Nodes*

The initial implementation of the Quantum component was a direct port of libquantum. The basic functional unit was defined in the Register class. The Register represents a quantum system of a specified bit-width. Internally, a Register consisted of a series of Node objects, each of which represented a possible state for the register. Each Node, in turn, had an associated state and probability. These structures are shown in figure 4.3.

The Register class provides methods for manipulating the quantum state by application of Matrix or Gate objects. Gate objects were simply predefined matrices packaged into the library, such as Sigma, Rotational, and Hadamard. Although the Register object also provided methods for converting a Register to and from a Matrix object, Matrix multiplication was not used internally in libquantum. Instead, a series of loops and conditionals were used to mimic matrix multiplication. Presumably, there was a performance gain in avoiding a pair of conversions for each operation. As such, our initial implementation followed this design.

Libquantum provides three functions for measurement, which were ported exactly. All of the measurement functions are based on generating a random number

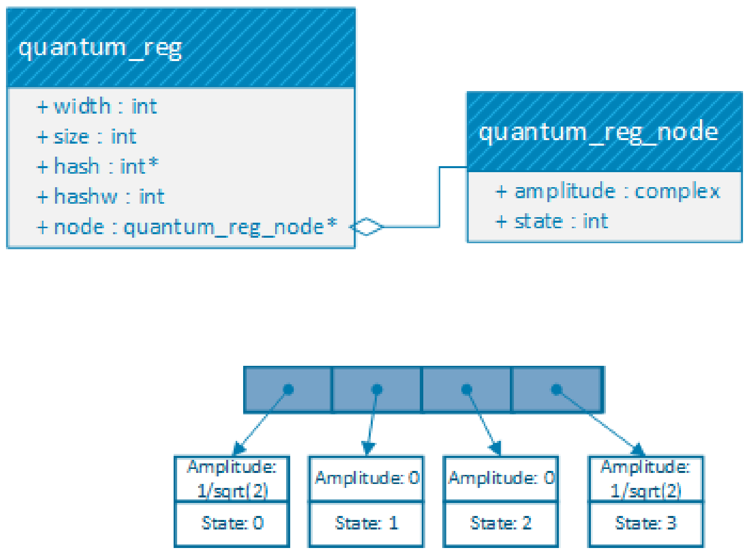

*Figure 4.3: Register & Node structures in iteration 1.*

between zero and one. The built in C rand function is used to set a threshold value. The first function performs a measurement of the entire quantum state. All node probabilities are collected and summed until the random number threshold is hit. The state contained in the node which broke the random threshold is returned as the result. All information in the entire register structure is then destroyed.

The last two measurement functions each measure only a single target bit in the register. The functions are identical, except that one of them "preserves" the quantum state of the measured bit, which is impossible in quantum mechanics. These functions collect and sum the node probabilities for which the target bit is zero. If the sum is less than the random number, the bit is determined to be a one. Otherwise, the bit is a zero. In the case of the "non-preserving" measurement function, node probabilities are rebalanced to reflect the measured bit.

While a single Register object was certainly capable of representing an entangled state locally, many of the protocols in our research involved the transmission of halves of entangled pairs. Examples of this behavior can be seen in some of the entanglement based variations to BB84 [56] and in the Ekert91 protocol [57]. Quantum teleportation relies on the ability to split up the two halves of an entangled pair. If an entangled system must be contained within a single object, and that entangled system is to be shared between two distributed nodes, then that object must be managed in a distributed fashion as well.

To support distributed entanglement, we first attempted to reframe the requirement. Rather than holding the entangled system within one object, the entanglement model split the entangled qubits into a pair of registers, and created management objects to broker the relationship between them. The maintenance of an entangled state across two distributed objects is described in section 4.1.2.

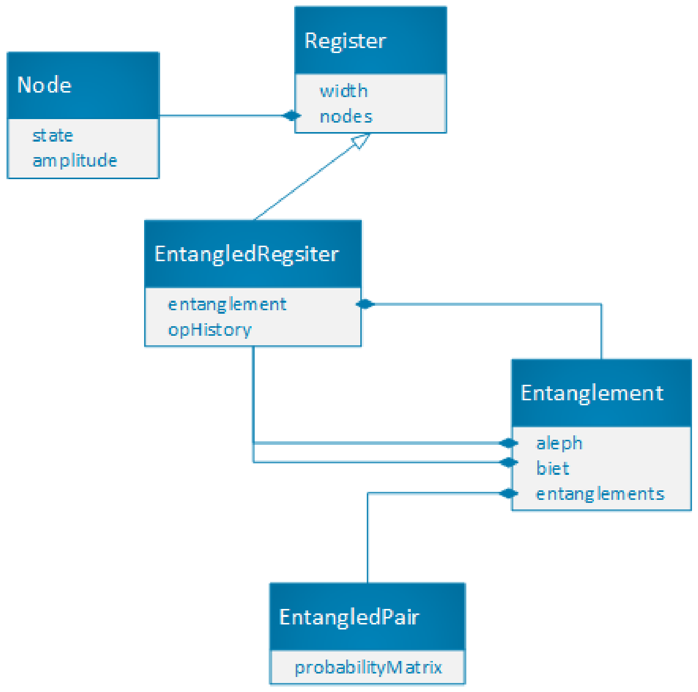

*Figure 4.4: Entanglement structures in iteration 1.*

The Register class was extended to implement the EntangledRegister class. The EntangledRegister class contained a reference to an Entanglement object. The Entanglement object, in turn, related a pair of EntangledRegisters by way of a set of EntanglementPair probability tables. The Entanglement was responsible for maintaining communication between EntangledRegisters, and processing measurement operations. Each position in the related EntanglementRegister objects requires exactly one EntanglementPair. An EntanglementPair manages four probabilities; one for each possible combination of the related bit positions.

When a qubit is sent to a remote system, it is replaced with a stub object on the local system. This stub object holds information about how to find and communicate with the remote object. When a portion of the entangled state is measured, for example, the Entanglement object calls the "notifyMeasurement" method of the stub object, which in turn uses gRPC to communicate the measurement message to the remote object.

The entanglement structures added in the first iteration were effective in enabling bi-partite entanglement of qubits, with the additional stipulation that qubits in one register only be entangled with a single qubit in the associated paired register. This was a step forward in that the initial implementation only supported entanglement within a single register. By the close of this iteration, QooSim supported network transmission of qubits, including qubits in entangled pairs. However, the library still suffered from the same memory consumption limitations as the original libquantum. For example, the BB84 implementation was limited to processing qubits in bursts of 16-bit registers.

### *4.2.5 Iteration 2 - Registers & Qubits*

The second iteration of QooSim focused on combatting the memory consumption issue by replacing the underlying structures of a Register object. In the realm of quantum key distribution and quantum communication, absent error correction mechanisms, most streams of qubits exist as unentangled entities. For example, when sending a key using the pure theoretical implementations of BB84 or Kak06 protocols, there is no reason to entangle any of the qubits [23, 24, 37, 39]. Even in protocols such as Ekert91, which use entanglement, qubits are entangled into pairs, with one half of the pair being transmitted.

The Node object structure underlying libquantum and the initial implementation of QooSim is idealized to represent a system entangled to the degree of the width of the register. In an application in which no two bit positions within the same register are entangled, only $log_2(n)$ nodes will have nonzero amplitudes, while $2^n$ nodes are held within the register. Fully $2^n - log_2(n)$ nodes have a probability value of zero. This means that under these applications, memory consumption will grow exponentially, while the actual information represented will grow linearly. As the width of the Register object grows, the efficiency of the Node representation diminishes rapidly.

To optimize the simulator for quantum communication and key distribution protocols, the Node object representation was removed in favor of a Qubit object representation. Each bit in a register was represented by a distinct Qubit object. The Qubit contained just one property; the alpha value, or probability amplitude of the zero state. Measurement and application of Matrix or Gate operations deferred to the Qubit object itself. A measurement of the entire register was reimplemented as a shorthand method for measuring each Qubit object in turn.

The efficiency gain was impressive. Instead of being limited to 22-bit wide registers as with libquantum, registers 50 times as large were easily possible. However, these improvements were not without cost in functionality. Because each bit in the register was now a distinct entity, and the combination of these entities was not possible, no longer could qubits within the same register be entangled. Additionally, the second iteration was still limited by the same conditions on entanglement as the first; only qubits belonging to a related pair of Register objects were capable of entanglement.

### 4.2.6 Iteration 3 - Qubits & State Vectors

As our research moved into the realms of quantum error correction syndrome codes and quantum repeater implementation through teleportation, it became increasingly important not only to support multipartite entanglement in a distributed environment.

The third iteration of QooSim is a dramatic departure from the libquantum roots. In fact, this iteration turns the representation of quantum states quite literally upside down. Instead of interacting with a Register object, developers working with QooSim manipulate Qubits directly. In fact, there is no register object in the third iteration. This iteration is extremely promising, offering the efficiency of the second iteration without the artificial restrictions imposed by the Entanglement object's management.

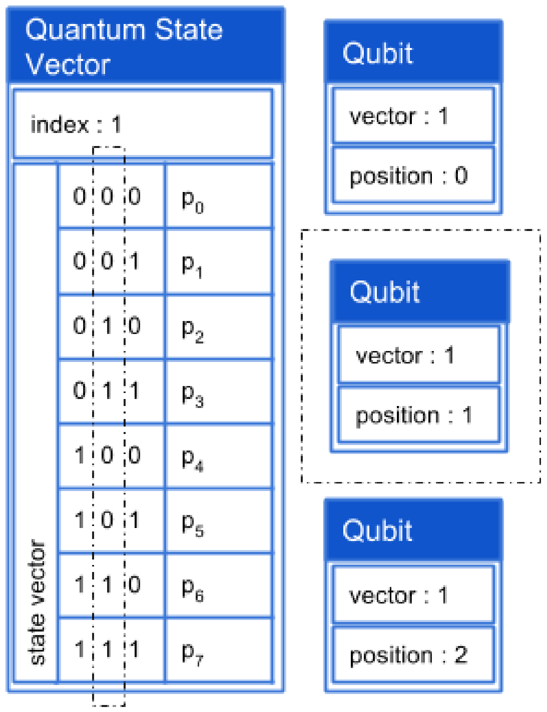

*Figure 4.5: Structure of state & vector classes in iteration 3.*

This third iteration is the closest to a theoretical model of distributed quantum

| Qubit Representation Efficiency | | | | | | |
|---|---|---|---|---|---|---|
| | Iteration 1 (register of nodes) | | Iteration 2 (register of qubits) | | Iteration 3 (state vector) | |
| | Floats | States | Floats | States | Floats | States |
| non-entangled qubits | 65536 | 2 | 16 | 32 | 16 | 32 |
| bi-partite entangled qubits | 65536 | 4 | 80 | 32 | 32 | 32 |
| maximally entangled qubits | 65536 | 65536 | N/A | N/A | 65536 | 65536 |

*Table 4.1: Efficiency of representing degrees of entanglement using structures capable of maintaining at least 16 qubits, by iteration.*

interactions to date. Distributed interactions are still based around the asynchronous model presented in section 4.1.2. However, due to greater degrees of entanglement, these interactions are not limited to only two parties. Indeed, a three party teleportation system can be demonstrated as in appendix A.2.

The greater degrees of entanglement are enabled through an abstraction layer riding below the qubits to represent system state, instead of above them. Rather than positioning the developer to interact with the system state, it is arguably more natural that they should interact with individual Qubit objects. The Qubit objects, in turn, are no longer simply holders of an alpha value, but pointers to a "position" in a StateVector object.

At first, this paradigm shift may seem to be simple semantics. However, the StateVector also implemented a degree of intelligence. Gate or matrix operations are still initiated on the individual Qubit object. The arguments to the operation, including the Qubit from which is used to initiate the operation if necessary, are passed as a vector. This call is delegated to the StateVector object. The StateVector object then determines whether or not it holds information for all Qubits passed as arguments. If it does not, it merges StateVector objects from other arguments into itself. Conversely, at the close of an operation or when processing a measurement, the StateVector will dynamically split itself if it detects unentangled bit positions.

There are two important consequences of this architecture. First, it is possible to simulate multipartite entanglement at will. Second, memory consumption is optimized so that the exponential consumption only takes place when it is required by entanglement, while linear, non-entangled consumption is the norm.

### Dynamic allocation of vector space

Vector space is dynamically increased to accommodate any operation being performed on a Qubit represented by the StateVector. By design choice, StateVector of the first Qubit argument becomes the “host” of the operation. The StateVectors of all other Qubit arguments are folded into the host’s StateVector by application of a matrix tensor product. The position values of “guest” Qubit objects are updated to reflect their new positions within the “host” StateVector, and the guest’s former StateVectors are deallocated and destroyed.

### Dynamic deallocation of vector space

At the conclusion of each quantum operation or measurement, the reduce method is called. This method searches through the StateVector for any positions which do not have an amplitude for more than one value; a Qubit which is always zero or always one. Such a Qubit can be factored out of the StateVector. The reduce method shifts positions in the StateVector so that the Qubit is in the most significant position. The reduce method allocates a new StateVector for this Qubit, and then erases it from the current StateVector. The remaining StateVector is divided in half. The top half is retained if the factored Qubit was valued one, the bottom half if it was valued zero. Positions of all remaining qubits are adjusted to reflect the change in the logical layout of the StateVector.

## *4.3 QooSim in Practice*

This section contains practical analysis of several quantum protocols using the third iteration of the QooSim system. As the system is still very new, some bugfixes were made throughout the simulations. Each simulation was re-run following bugfixes.

An outstanding problem remaining is the growth of search-space for quantum memory address management. The memory system is currently implemented as a C++ vector of memory map objects. To find a given object by state vector index and position, on average, $N$ comparisons must be executed. In a protocol such as BB84, over 2,000 qubits may be needed to generate a 1024 bit key. Certain allowances were made to compensate for these requirements over repeated runs. Where possible, protocols were executed twenty to one-hundred times by a single process. Some protocols, such as Kak06, required the use of specialized “test harness” shell scripts for these repeated runs.

The purpose of these simulations was to examine the security and efficiency of quantum cryptographic protocols under non-ideal conditions. The results of these simulations can provide useful additional information to an ongoing conversation about the limits on security of these protocols [69, 72].

### *4.3.1 BB84*

#### Eavesdropping Error Analysis

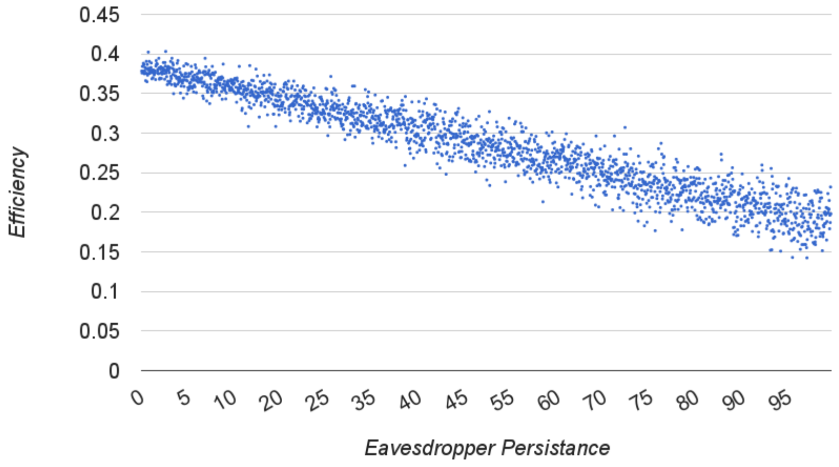

*Figure 4.6: BB84 Efficiency against an eavesdropper, where E N is the percentage of qubits the eavesdropper measures.*

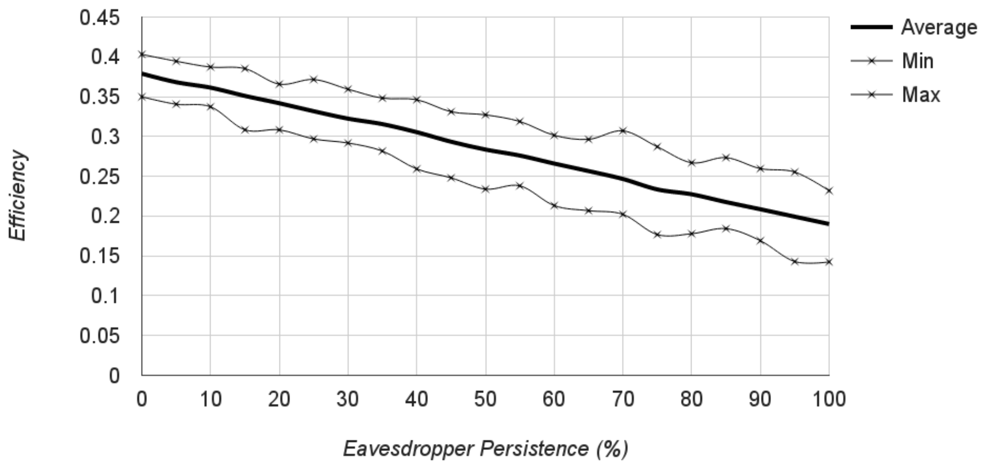

*Figure 4.7: BB84 Efficiency against an eavesdropper, lines of fit for average, minimum and maximum values at each interval.*

The classic implementation of BB84 was executed under several different conditions. The protocol was tested against a simulated eavesdropper. The eavesdropper was implemented as a channel interface object which accepted a parameter for frequency of measurement. Tests were executed at frequency intervals of five percent. The scatter plot in figure 4.6 shows the results of this simulation. Figure 4.7 shows the same data with a line of best fit for average performance, two standard deviations, and boundaries for minimum and maximum values.

The results are more or less linear, with a fanning out towards the higher error rates, which reflects more entropy in the system. Efficiency of the BB84 protocol under eavesdropping appears to be bounded by around 40%, with a worst case scenario of 10%. Based on this analysis, if Alice and Bob have a perfect channel, they may begin to suspect that an eavesdropper is present if efficiency is below 35%.

## Amplitude Damping Error Analysis

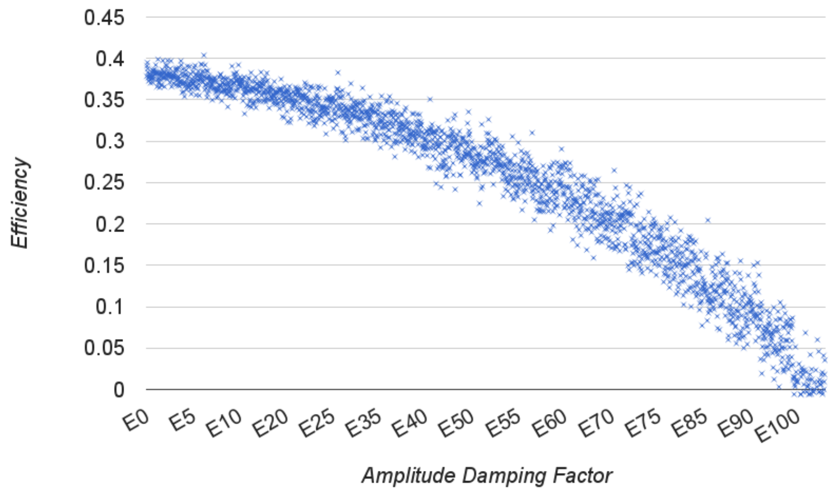

*Figure 4.8: BB84 Efficiency against amplitude damping, where the amplitude damping factor is the percentage of qubits affected by amplitude damping.*

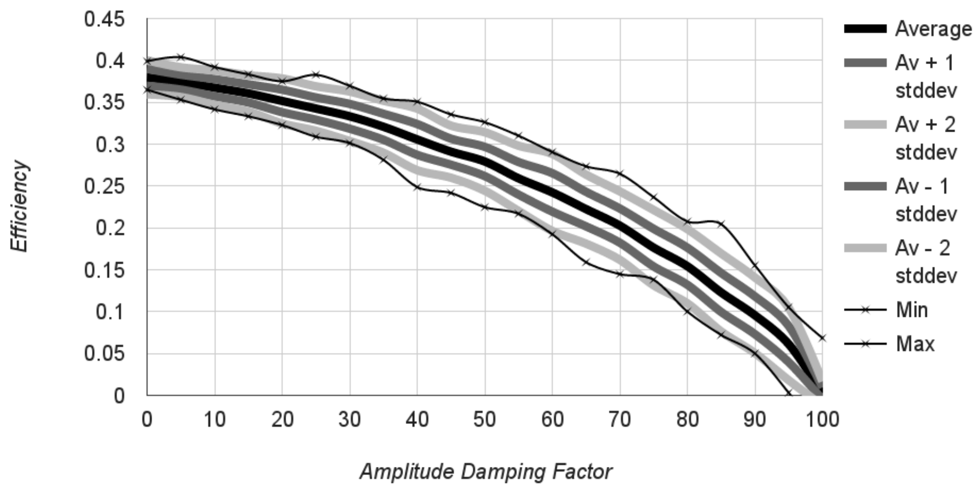

*Figure 4.9: BB84 Efficiency against an amplitude damping, lines of fit for average, minimum and maximum values at each interval.*

Next, BB84 was tested against amplitude damping (figure 4.8 and 4.9). Amplitude damping was constant across all qubits used in communication. Factors in intervals of .05 were used against the classic version of the protocol in an eavesdropper free channel. The outcome is now a gently sloping curve, terminating in zero efficiency at an amplitude damping factor of 100%. What's most useful about this scenario are the test factors .1, .9, and .99. If amplitude damping truly does grow exponentially with distance, these factors represent efficiency at distances of $d$, $2d$, and $3d$.

## Rotational Error Analysis

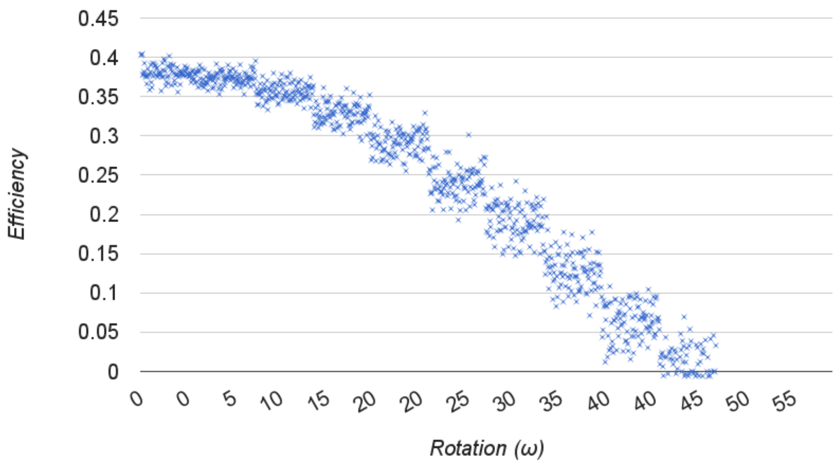

*Figure 4.10: BB84 Efficiency against rotational errors, shown as ω degrees of rotation.*

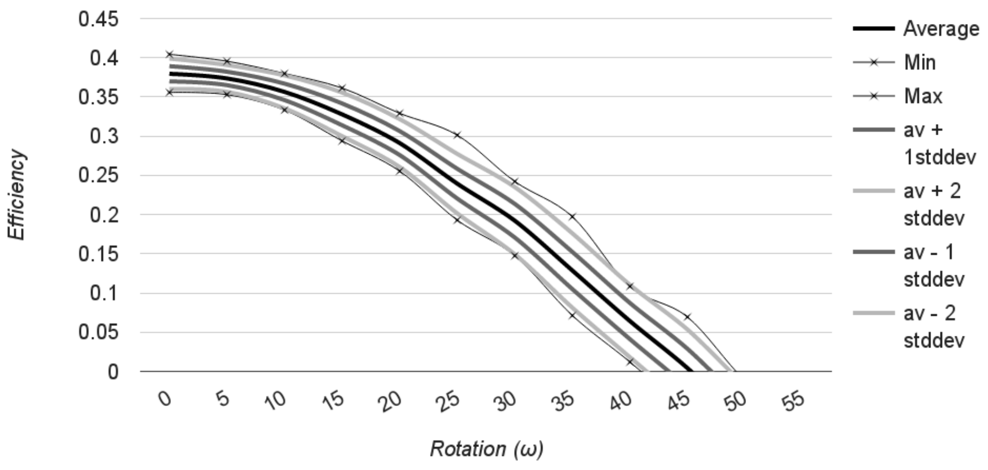

*Figure 4.11: BB84 Efficiency against rotational errors, lines of fit for average, minimum and maximum values at each interval.*

The simulated run of BB84 against rotational errors was constructed such that rotation was constant across all qubits used in communication. Rotations of 0 to 90 degrees were evaluated at intervals of 5 degrees. At the outset, BB84 appears to be affected by rotational affected by rotation in much the same way as amplitude damping and eavesdropping. However, once the rotational degree grows above 10 degrees, BB84 experiences a severe drop-off in efficiency, bottoming out at 45 degrees rotation. If satellite communications are to implement BB84, they must implement some compensation for the rotational errors which expected from these communication platforms.

## Comparative and Combined Error Condition Analysis

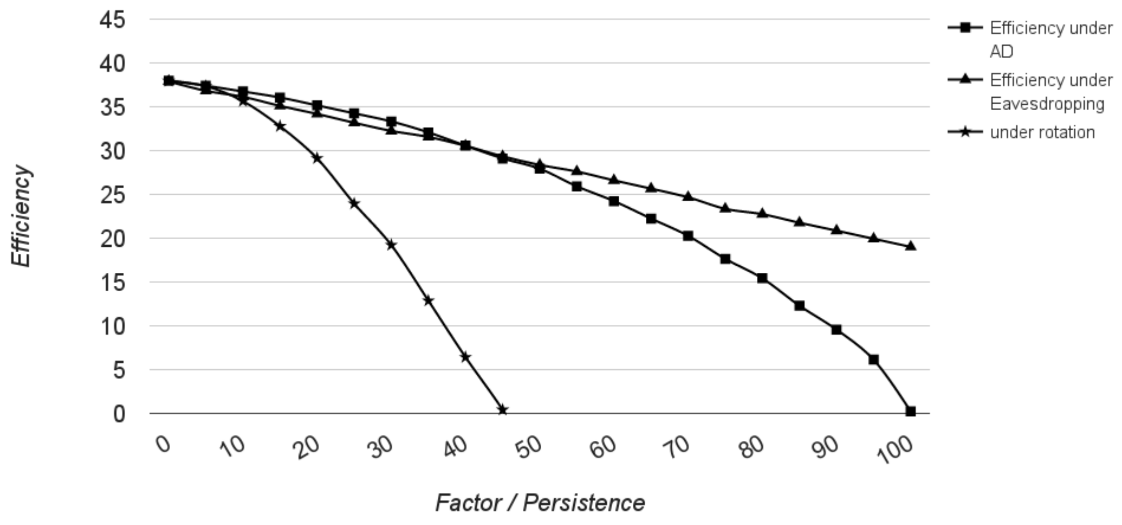

*Figure 4.12: BB84 efficiency against amplitude damping compared to eavesdropping and rotational errors.*

Figure 4.12 shows the average lines for BB84 performance against eavesdropping, amplitude damping, and rotation, separately. It is notable that efficiency suffers almost immediate drop off under rotation and a more gradual drop off under amplitude damping at 50% attenuation. Meanwhile, the efficiency drop off under eavesdropping is steady and gradual. These results imply that even slight rotational errors can have a significant impact on the security of BB84. Figure 4.13 explores the interaction of amplitude damping and eavesdropping in detail.

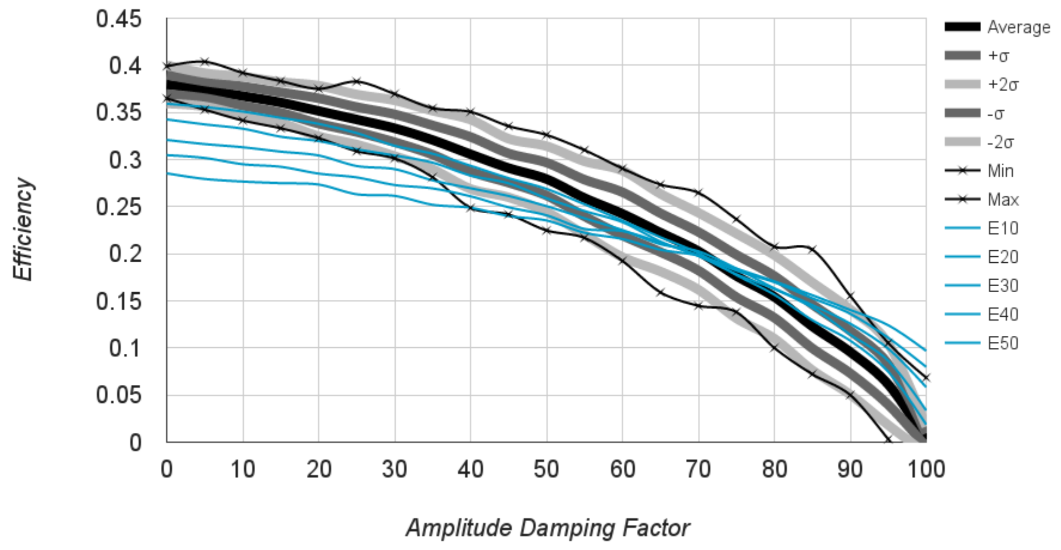

*Figure 4.13: BB84 Efficiency against amplitude damping with eavesdropping. Amplitude damping varies from 0 to 100 along the horizontal axis, while eavesdropping is held constant. The lines labeled as En mark eavesdropping at a rate of n.*

BB84 was simulated against an amplitude damping channel with an eavesdropper present. The amplitude damping factor was varied from 0 to 1 at intervals of .05, and computed for eavesdropper persistencies from 0 to .50, at intervals of .10. Two-thousand test cases were run for each permutation. For this simulator, the eavesdropper was configured to be close to the recipient's end of the communication channel.

The graph in figure 4.13 shows the average performance with amplitude damping but with 0 eavesdropper persistence as a thick, solid black line. One standard deviation from the average is plotted as a thick, solid, dark grey line, while two standard deviations from average is plotted as a thick, solid light grey line. The maximum and minimum for the cases run is plotted as a thin black line with "x" marks. Finally, the eavesdropper test cases are shown in thin, blue lines. The lines start from the left access in descending order; that is, the .10 persistence case is the top most thin blue line, and the .50 persistence case is the bottom most thin blue line.

This plot is very insightful, as it give us some idea about the security tolerance of BB84 under amplitude damping. For example, if we expect 50% signal loss due to amplitude damping, and we configure BB84 to accept two standard deviations below the average performance, we are accepting the risk of an eavesdropper intercepting up to 30% of the qubits sent. If, instead, we configure for minimum simulated performance, we are accepting the risk of an eavesdropper intercepting 50% or more of the qubits sent.

### *4.3.2 Kak06*

The Kak06 protocol is fast! Transmission rate governors were required to keep the two parties in the protocol from interrupting each other. Communication was limited to 128 qubit bursts, after which a 1 second "cooldown" was implemented. The Kak06 protocol was implemented using the $R_y(\theta)$ gate as the secret unitary transformation, with $\theta$ varying between 0 and 45 degrees with each bit transmitted. Data was assumed to be classical, and efficiency calculations are based on the outcome of a measurement operation following the bit exchange. Errors were simulated uni-directionally; that is, for only the passes from Alice to Bob.

Rotational error was simulated between 0 and 55 degrees, varying at 5 degree intervals. The results, shown in figure 4.14 and 4.15 are somewhat surprising. At first, the curve mimics the efficiency curves of BB84. However, after reaching a rotational error of 45 degrees, the efficiency surprisingly begins to rise again.

Kak06 differs from BB84 in two important ways. First, one must remember that BB84 has a built-in efficiency handicap of 50% due to base mismatching. Kak has an advantage here, in that this protocol should be at least 50% efficient, which matches the graph's lower bound. Second, Kak06 actually makes three quantum transmissions for each qubit generated. When we simulate an error rotation of 45 degrees on the channel, the qubit is actually rotated by 90 degrees off of normal, since it makes two passes through the affected channel. Therefore, assuming constant channel dynamics, the worst case for Kak06 would be rotational errors of odd multiples of 45: 45, 135, etc.

Another interesting feature of Kak06 is that the results for a fairly tightly bound band in 4.15. Presumably, if two parties knew the expected error rotation on their channel, this would make eavesdropper detection a simpler task.

Previous test cases show similar results for amplitude damping 4.16. Again, the multiple passes of the Kak06 protocol contribute to a strange "recovery" in efficiency, if not reliability. In the case of amplitude damping, this recovery is bounded by 75%, as all qubits in the damping channel are drawn towards zero. The trough in this graph, around 60%, represents the case where two passes are affected by damping, but the third is not. Again, Kak06 exhibits a very tight bounding of errors.

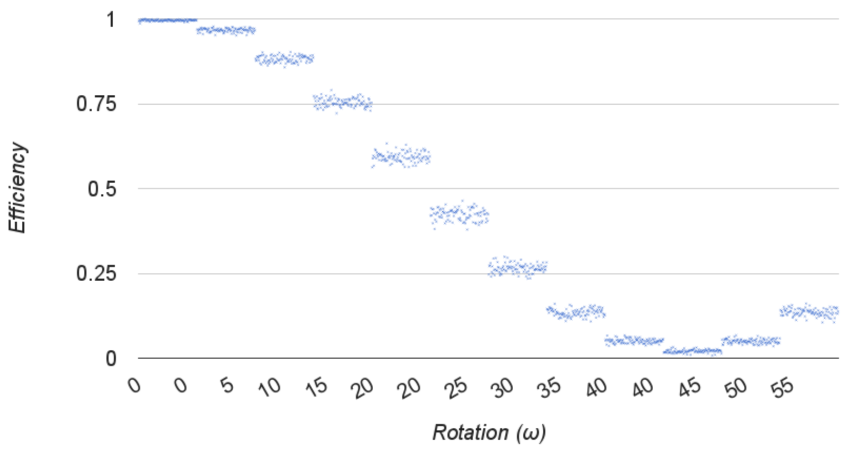

*Figure 4.14: Kak06 efficiency against rotation, scatter plot.*

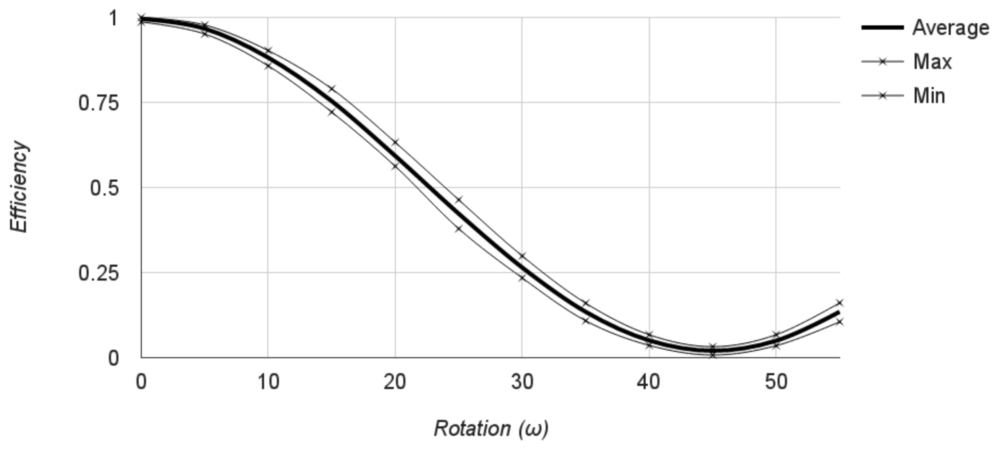

*Figure 4.15: Kak06 efficiency against rotation, average, minimum and maximum efficiency.*

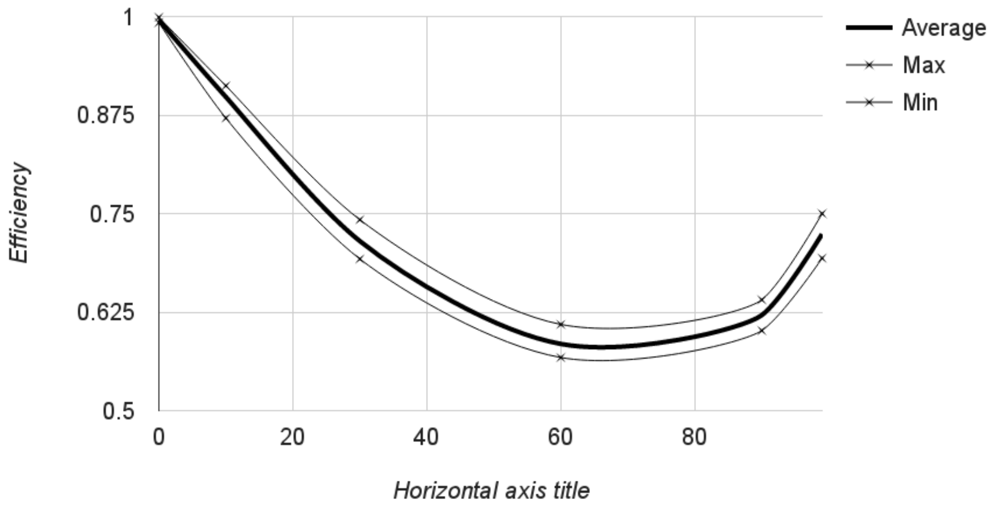

*Figure 4.16: Kak06 efficiency against amplitude damping, average, minimum and maximum efficiency.*

# Chapter 5

# Conclusions

Quantum computing is a field vastly apart from classical ideas about how information processing systems should behave. The mechanics which govern it are often counter-intuitive and without classical analogues. Even on a conceptual basis, quantum computing presents quite a challenge to those who wish to study its workings. Adding to the difficulty is a lack of available systems on which to experiment.

In this thesis, we have presented a background for understanding the basic concepts of quantum computing, including methods for representing and manipulating quantum bits. We have discussed two traits of quantum systems which are vital to the security of quantum cryptographic protocols: measurement and no-cloning, and have introduced the exotic idea of entanglement.

In our discussion of physical implementations for quantum systems, we presented several alternative mechanisms. We introduced these systems in order to provide a backdrop for discussion of quantum protocols and the types of errors to which quantum systems are subjected. We selected photonic quantum bits as the de facto implementation context for the remainder of the paper, as they are the most ideal type for communication.

Sections 3.1 and 3.2 presented the reader with a variety of well known quantum communication protocols and methods for modeling errors over quantum channels. These protocols showed methods for encoding data onto quantum channels to support key agreement or direct communication. Teleportation we presented to show a method by which entanglement and classical communications can be brought together to extend the range of a quantum network.

This factor motivated us to develop a system capable of simulating a generalized quantum computer reasonably well under a variety of circumstances. One key scenario for us was the transmission of entangled states across distributed network systems. This capability is key for simulating some of the available quantum communication protocols in and of themselves, and is universally required for the error detection or correction mechanisms examined in this thesis. In section 4.1.2, we presented an original proof of the equivalence of asynchronous operations of quantum states to traditional methods. This proof was the tool which allowed us to proceed in the development of our simulator.

Our development of QooSim is still evolving, but at this stage, we have shown it to be mature enough to simulate several of the common quantum protocols and channel conditions. Section 4.2 describes the development of the simulator from research through initial implementations to present state. The major structures of the system are outlined and their interactions detailed. In section 4.3, we demonstrate some of the algorithms QooSim is capable of simulating. Additionally, we show the effects of different types of channel errors on quantum communication protocols.

The simulator is flexible enough to allow end-users to add their own applications to the system without being concerned with the underlying simulator structures. We sincerely hope that through continued development of this simulator and the research we are able to accomplish with it that we can expose more learners and researchers to the world of quantum computing.

## *5.1 Future Work*

Future work may include the continuing to expand the usability of the simulator by streamlining the iRunnable interface. Improvement should be investigated in terms of implementing more advanced memory management strategies to support “continuously up” server models. Documentation and addition to the protocols included in this toolkit, will provide a greater breadth of learning experience. One of key goal would be to create a system or method for the visualization of quantum entangled information as it is used in algorithms and protocols across a network.

# Reference

[1] Vitanyi, Paul. "The quantum computing challenge." Informatics. Springer Berlin Heidelberg, 2001.

[2] Gershenfeld, Neil and Chuang, Isaac. "Quantum Computing with Molecules". *Scientific American* 278.6 (1998): 66-71.

[3] Shor, Peter W. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer." *SIAM Journal on Computing* 26.5 (1997): 1484-1509.

[4] C. H. Bennett and G. Brassard, "Quantum cryptography: Public-key distribution and coin tossing" *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing* , Bangalore, India, 1984, IEEE Press (1984):175–179.

[5] C.H. Bennett and G. Brassard, "Quantum public key distribution," *IBM Technical Disclosure Bulletin* 28, 3153–3163 (1985).

[6] Dieks, D. G. B. J. "Communication by EPR devices." *Physical Review A* 92.6 (1982): 271-272.

[7] M. Born (editor), *The Born-Einstein-Letters*, Macmillan, London(1971): 158.

[8] J. M. P. Armengol, H. Weinfurter, C. Barbieri, W. Leeb, J. G. Rarity, G. Baister, T. Schmitt-Manderbach, R. Ursin, T. Jennewein, M. Aspelmeyer, M. Pfennig-

bauer, L. Cacciapuoti, O. Minster, C. Matos, B. Furch, A. Zeilinger, “Quantum communications at ESA: Towards a space experiment on the ISS”, *Acta Astronautica*, 2008: 165 – 178.

[9] Ekert, Arthur. “Quantum Cryptography Based on Bell’s Theorem.” *Physical Review Letters* 67-661. Aug 1991.

[10] Branciard, Cyril; Gisin, Nicolas; Kraus, Barbara; Scarani, Valerio. “Security of Two Quantum Cryptography Protocol Using the Same Four Qubit States”. *Physical Review A* 72-3. Sept 2005.

[11] Bennett, Charles H. and Wiesner, Stephen J. “Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states.” *Physical Review Letters* 69-20. Nov 1992.

[12] Braunstein, Samuel L. “Quantum error correction for communication with linear optics.” *Nature* 394.6688 (1998): 47-49.

[13] Reed, M. D., et al. “Realization of three-qubit quantum error correction with superconducting circuits.” *Nature* 482.7385 (2012): 382-385.

[14] Chuang, Isaac L., and Yoshihisa Yamamoto. “Quantum bit regeneration.” *Physical Review Letters* 76.22 (1996): 4281.

[15] Briegel, H-J., et al. “Quantum repeaters: The role of imperfect local operations in quantum communication.” *Physical Review Letters* 81.26 (1998): 5932.

[16] Zhao, Zhi, et al. “Experimental realization of entanglement concentration and a quantum repeater.” *Physical Review Letters* 90.20 (2003): 207901.

[17] Gottesman, Daniel. “Class of quantum error-correcting codes saturating the quantum Hamming bound.” *Physical Review A* 54.3 (1996): 1862.

[18] Davies, Paul Charles William, and Julian R. Brown. *The ghost in the atom: a discussion of the mysteries of quantum physics.* Cambridge University Press, 1993.

[19] Scarani, Valerio, et al."Quantum cryptography protocols robust against photon number splitting attacks for weak laser pulse implementations." *Physical Review Letters* 92.5 (2004): 057901.

[20] Jaeger, Gregg. *Quantum Information.* Springer New York, 2007.

[21] Faye, Jan,"Copenhagen Interpretation of Quantum Mechanics", *The Stanford Encyclopedia of Philosophy* (Fall 2014 Edition), Edward N. Zalta (ed.), URL = http://plato.stanford.edu/archives/fall2014/entries/qm-copenhagen/.

[22] Horodecki, Ryszard, et al."Quantum Entanglement." *Reviews of Modern Physics* 81.2 (2009): 865.

[23] Desurvire, Emmanuel. *Classical and Quantum Information Theory.* Cambridge University Press, 1st Edition. 2009.

[24] Nielsen, Michael A and Chuang, Isaac L. *Quantum Computation and Quantum Information.* Cambridge University Press 10th Edition. 2010.

[25] Ladd, Thaddeus D., et al."Quantum Computing." *arXiv* preprint arXiv:1009.2267 (2010).

[26] Gao, W. B., et al."Observation of entanglement between a quantum dot spin and a single photon." *Nature* 491.7424 (2012): 426-430.

[27] Morton, John JL, et al."Solid-state quantum memory using the 31P nuclear spin." *Nature* 455.7216 (2008): 1085-1088.

[28] Saeedi, Kamyar, et al."Room-temperature quantum bit storage exceeding 39 minutes using ionized donors in silicon-28." *Science* 342.6160 (2013): 830-833.

[29] Menicucci, N. C., and Carlton M. Caves. "Local realistic model for the dynamics of bulk-ensemble NMR information processing." *Physical Review Letters* 88.16 (2002): 167901.

[30] Ma, Xiao-Song, et al. "Quantum teleportation over 143 kilometres using active feed-forward." *Nature* 489.7415 (2012): 269-273.

[31] Stucki, Damien, et al. "High rate, long-distance quantum key distribution over 250 km of ultra low loss fibres." *New Journal of Physics* 11.7 (2009): 075003.

[32] Gisin, Nicolas. "How far can one send a photon?" *Frontiers of Physics* 10.6 (2015): 1-8.

[33] Comandar, L. C., et al. "Room temperature single-photon detectors for high bit rate quantum key distribution." *Applied Physics Letters* 104.2 (2014): 021101.

[34] Elliott, Chip, et al. "Current status of the DARPA quantum network." *Defense and Security*. International Society for Optics and Photonics, 2005.

[35] Elliott, Chip. "The DARPA quantum network." *Quantum Communications and cryptography* (2006): 83-102.

[36] C. H. Bennett and G. Brassard, "Quantum cryptography: Public-key distribution and coin tossing" in *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing*, Bangalore, India, 1984, (IEEE Press, 1984), pp. 175–179.

[37] Gisin, Nicolas, et al. "Quantum cryptography." *Reviews of Modern Physics* 74.1 (2002): 145.

[38] Ilic, Nikolina. "The Ekert protocol." *Journal of Phy334* 1 (2007): 22.

[39] Kak, Subhash. "A three-stage quantum cryptography protocol." *Foundations of Physics Letters* 19.3 (2006): 293-296.

[40] Wu, Linsen, and Yuhua Chen.“Three-Stage Quantum Cryptography Protocol under Collective-Rotation Noise.” *Entropy* 17.5 (2015): 2919-2931.

[41] Devitt, Simon J., William J. Munro, and Kae Nemoto.“Quantum error correction for beginners.” *Reports on Progress in Physics* 76.7 (2013): 076001.

[42] Sharma, Rishi Dutt, et al.“Which verification qubits perform best for secure communication in noisy channel?.” *Quantum Information Processing* (2015): 1-16.

[43] R. B. Griffiths, “Quantum Channels, Kraus Operators, POVMs,” Quantum Computation and Quantum Information Theory Course Notes, Carnegie Mellon University (2010).

[44] Qin, SuJuan, et al.“Quantum secure direct communication over the collective amplitude damping channel.” *Science in China Series G: Physics, Mechanics and Astronomy* 52.8 (2009): 1208-1212.

[45] Whaley, Brigitta *Mixed States and Density Matrix, Entanglement measures.* Internet: https://inst.eecs.berkeley.edu/ cs191/fa09/lectures/lecture13_fa09.pdf [27 Mar 2016]

[46] Yanofsky, Noson S, Mannucci, Mirco A.“Quantum Computing for Computer Scientists.” Cambridge University Press, 2013.

[47] Hughes, Richard J., et al.“Practical free-space quantum key distribution over 10 km in daylight and at night.” *New Journal of Physics* 4.1 (2002): 43.

[48] C. Bonato, M. Aspelmeyer, T. Jennewein, C. Pernechele, P. Villoresi, and A. Zeilinger, ”Influence of satellite motion on polarization qubits in a Space-Earth quantum communication link,” *Opt. Express* 14, 10050- 10059 (2006).

[49] Wang, Gang, et al. "Polarization tracking for quantum satellite communications." *SPIE Defense+ Security.* International Society for Optics and Photonics, 2014.

[50] M. Aspelmeyer, T. Jennewein, M. Pfennigbauer, W. R. Leeb, A. Zeilinger, "Long distance quantum communication with entangled photons using satellites", *IEEE J. Sel. Top. Quantum Electron.* 9, 1541 (2003)

[51] Van Meter, Rodney, et al. "System design for a long-line quantum repeater." *IEEE/ACM Transactions on Networking (TON)* 17.3 (2009): 1002-1013.

[52] Azuma, Koji, Kiyoshi Tamaki, and Hoi-Kwong Lo. "All-photonic quantum repeaters." *Nature Communications* 6 (2015).

[53] Kwiat, Paul G., et al. "Ultrabright source of polarization-entangled photons." *Physical Review A* 60.2 (1999): R773.

[54] Vazirani, Umesh. *Lecture 2: Quantum Algorithms.* Internet: http://www.cs.berkeley.edu/ṽazirani/s09quantum/notes/lecture2.pdf [20 Jul 2015].

[55] Moore, Michael G. *Lecture 24: Tensor Product States.* Internet: http://www.pa.msu.edu mmoore/Lect24_TensorProduct.pdf [23 Jul 2015].

[56] C. H. Bennett, G. Brassard, and N. D. Mermin. "Quantum cryptography without bell's theorem." *Physical Review Letters.* 68:557–559, Feb 1992.

[57] A. K. Ekert." Quantum cryptography based on bell's theorem." *Physical Review Letters.* 67:661–663, Aug 1991.

[58] Bjorn Butscher, Hendrik Weimer (2015), http://www.libquantum.de/bibliography

[59] Gough, Alex. *Quantum::Entanglement.* Internet: http://www.perl.com/pub/2001/08/08/quantum.html [10 oct 2015]

[60] Brassard, Gilles, and Louis Salvail. “Secret-key reconciliation by public discussion.” *Advances in Cryptology—EUROCRYPT’93.* Springer Berlin Heidelberg, 1993.

[61] Fiat, Amos, and Adi Shamir. “How to prove yourself: Practical solutions to identification and signature problems.” *Advances in Cryptology—CRYPTO’86.* Springer Berlin Heidelberg, 1986.

[62] Parakh, A. (2016) Quantum Cryptography: Overview. To appear in Encyclopedia of Information Assurance, Taylor and Francis.

[63] Subramaniam, P. and Parakh, A. (2016) A quantum Diffie-Hellman protocol, International Journal of Security and Networks, Inderscience, In Press.

[64] Parakh, A. and vanBrandwijk, J. (2016) Correcting Rotational Errors in Three Stage QKD, 23rd IEEE International Conference on Telecommunication (ICT 2016), Thessaloniki, Greece, May 16-18, 2016.

[65] Parakh, A. and vanBrandwijk, J. (2016) Rotational Error Correction in Three Stage QKD, IEEE Communication Theory Workshop (CTW 2016), Nafplio, Greece, May 15-18, 2016.

[66] Abhishek Parakh and Pramode Verma and Mahadevan Subramaniam. “Improving efficiency of quantum key distribution with probabilistic measurements.” *International Journal of Security and Networks* 11 1-2. (2016): 37-47.

[67] Abhishek Parakh. “A probabilistic quantum key transfer protocol.” *Security and Communication Networks* 6-11. (2013): 1389-1395.

[68] Abhishek Parakh. “New protocol for quantum public key cryptography.” *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS).* (2015): 1-3.

[69] Abhishek Parakh. “Quantifying the security of a QKD protocol” *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*. (2015): 1-3.

[70] Parakh, Abhishek. “Quantum teleportation for keyless cryptography.” *SPIE Sensing Technology + Applications.* International Society for Optics and Photonics, 2015.

[71] Mandal, Sayonnha, and Abhishek Parakh. “Implementing Diffie-Hellman key exchange using quantum EPR pairs.”*SPIE Sensing Technology+ Applications.* International Society for Optics and Photonics, 2015.

[72] Subramaniam, Pranav, and Abhishek Parakh. “Limits on detecting eavesdropper in QKD protocols.” *Advanced Networks and Telecommuncations Systems (ANTS), 2014 IEEE International Conference on.* IEEE, 2014.

[73] Subramaniam, Pranav, and Abhishek Parakh. “A quantum Diffie-Hellman protocol using commuting transformations.” *Advanced Networks and Telecommuncations Systems (ANTS), 2014 IEEE International Conference on.* IEEE, 2014.

[74] Parakh, Abhishek, and Pulkit Verma. “Improving the efficiency of entanglement based quantum key exchange.” *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on.* IEEE, 2014.

[75] Parakh, Abhishek. “A quantum oblivious transfer protocol.” *SPIE Optical Engineering+ Applications.* International Society for Optics and Photonics, 2013.

# Appendix A

# Source Code Listings

The listed source code is intended to enhance the reader's understanding of critical portions of the QooSim library. Full source code can be found in the github project located at https://github.com/vanbrandwijk/libquantum-oo in the multiqubitops branch, tagged as 0.3.2-alpha as of the release of this thesis. The StateVector class is provided in this document, as it is the "heart and soul" of the Quantum namespace in iteration 3. This class is responsible for the matrix representation of quantum states. Example runnables for teleportation and Kak06 protocol are also provided, in order to demonstrate the ease of implementation with this library.

## *A.1 StateVector*

```
/*
 * stateVector.cpp
 */
//STD C++ includes
#include <algorithm>
#include <complex.h>
#include <math.h>
#include <stdlib.h>
#include <vector>

//QooSim Includes
#include "channelService_client.h"
#include "matrix.h"
#include "qubit.h"
#include "qubitMap.h"
#include "remotePeer.h"
```

```
#include "stateVector.h"
#include "stateVectorOperation.h"

//External Includes
#include "externals/jacobi_eigenvalue.hpp"

using namespace std;

namespace Quantum {
/*
 * StateVector
 * Parameterless constructor, creates a StateVector of width 1
 */
StateVector::StateVector() : StateVector(1) {
}

/*
 * StateVector
 * @param bitWidth the width of the vector
 * Constructor, creates a vector with width equal to bitWidth
 */
StateVector::StateVector(int bitWidth)
        : qsv(1, pow(2, bitWidth)), remoteQubits(bitWidth) {
        this->qsv.set(0, 0, 1);
}

/*
 * StateVector
 * @param m a matrix object
 * Constructor, creates a vector from the matrix object
 */
StateVector::StateVector(Matrix m)
        : qsv(m), remoteQubits(log(m.getRows())/log(2)) {
}

/*
 * setIndex
 * @param newIndex the input index
 * Sets the memory map index of this vector
 */
void StateVector::setIndex(int newIndex) {
        this->index = newIndex;
}

/*
 * getIndex
 * @return the memory map index of this vector
 */
int StateVector::getIndex() {
        return this->index;
}

/*
 * resize
 * @param newSize
 * Add space the the remoteQubits vector
 */
void StateVector::resize(int newSize) {
        this->remoteQubits.resize(newSize);
}

/*
 * applyOperation
 * @param m operation a matrix operator
 * @param input1 the position of the qubit to be operated on
 * Shortcut method to apply an operation to one qubit already in
 * this vector
 */
```

```
void StateVector::applyOperation(Matrix operation, int input1) {
        vector<int> inputs(1);
        inputs.at(0) = input1;
        this->applyOperation(operation, inputs);
}

/*
 * applyOperation
 * @param m operation a matrix operator
 * @param input1 the position of a qubit to be operated on
 * @param input2 the position of a qubit to be operated on
 * Shortcut method to apply an operation to two qubits already in
 * this vector
 */
void StateVector::applyOperation(Matrix operation, int input1, int input2) {
        vector<int> inputs(2);
        inputs.at(0) = input1;
        inputs.at(1) = input2;
        this->applyOperation(operation, inputs);
}

/*
 * applyOperation
 * @param m operation a matrix operator
 * @param inputs the positions of qubits to be operated on
 * Apply an operation to an arbitrary number of qubits in this vector
 */
void StateVector::applyOperation(Matrix operation, vector<int> inputs) {
        this->applyOperation(operation, inputs, true);
}

/*
 * applyOperation
 * @param operation a matrix operator
 * @param inputs the positions of qubits to be operated on
 * @param addToHistory whether or not to push this operation onto the
 * vector's history
 * Apply an operation to an arbitrary number of qubits in this vector
 */
void StateVector::applyOperation(Matrix operation, vector<int> inputs,
        bool addToHistory) {
        Matrix expandedOperation = operation;
        unsigned int i;
        vector<int> rowMap = generateRowMap(inputs);
        Matrix scratch( this->qsv.getCols(), this->qsv.getRows() );

        for ( i = 0; i < rowMap.size(); i++ ) {
                scratch.set(0, rowMap.at(i), this->qsv.get(0, i));
        }

        for ( i = inputs.size(); i < this->getWidth(); i++ ) {
                expandedOperation = Matrix::matrixTensor(Matrix::identity(),
                         expandedOperation);
        }

        scratch = Matrix::matrixMultiply(expandedOperation, scratch);

        for ( i = 0; i < rowMap.size(); i++ ) {
                this->qsv.set(0, i, scratch.get(0, rowMap.at(i)));
        }

        if ( addToHistory ) {
                this->reduce();
                this->opHistory.push_back(
                        StateVectorOperation(operation, inputs));
        }
}
```

```
/*
 * applyOperation
 * @param operation a matrix operator
 * @param inputs the qubit objects on which to apply the operation
 * Apply an operation to an arbitrary number of qubits, consolidating them
 * into this vector first
 */
void StateVector::applyOperation(Matrix operation,
        vector< shared_ptr<Qubit> > inputs) {
        unsigned int i, j;
        vector<int> inputPositions(inputs.size());

        QubitMap* m = QubitMap::getInstance();

        //first, collect all the qubit state vectors into this vector
        for ( i = 0; i < inputs.size(); i++ ) {
                int inputIndex = inputs.at(i)->v->index;
                if ( this->index != inputIndex ) {
                        Matrix temp = Matrix::matrixTensor(
                                        inputs.at(i)->v->qsv, this->qsv);

                        for ( j = 0; j < m->numQubits(); j++ ) {
                                if ( inputIndex
                                        == m->getQubit(j)->v->index ) {
                                        m->getQubit(j)->position =+
                                                this->getWidth();
                                        m->getQubit(j)->v =
                                                shared_from_this();
                                }
                        }
                        this->qsv = temp;
                }
        }

        //next, convert the vector of Qubits to a vector of ints and process
        for ( i = 0; i < inputs.size(); i++ ) {
                inputPositions.at(i) = inputs.at(i)->position;
        }
        this->applyOperation(operation, inputPositions);
        this->resize(this->getWidth());
}

/*
 * generateRowMap
 * @param inputs the inputs to generate a map for
 * @return integer vector of the rowmap
 * Generate a mapping of the rows in this vector in order to place the
 * inputs provided in the left most positions
 */
vector<int> StateVector::generateRowMap(vector<int> inputs) {
        vector<int> positionMap(this->getWidth(),-1);
        vector<int> rowMap(this->qsv.getRows(),0);
        int i, j, zWidth;

        zWidth = this->getWidth() - 1;

        for ( i = 0; i < inputs.size(); i++ ) {
                positionMap.at(inputs.at(i)) = i;
        }

        for ( j = 0; j < positionMap.size(); j++ ) {
                if ( positionMap.at(j) == -1 ) {
                        positionMap.at(j) = i;
                        i++;
                }
        }

        for ( i = 0; i < this->qsv.getRows(); i++ ) {
                rowMap.at(i) = 0;
                for ( j = 0; j < positionMap.size(); j++ ) {
                        rowMap.at(i) += ((( i >> (zWidth - j) ) % 2)
                                        << (zWidth - positionMap.at(j)));
                }
        }

        return rowMap;
}

/*
 * getWidth
 * @returns the bit width of this vector
 */
int StateVector::getWidth() {
        return log(this->qsv.getRows()) / log(2);
}

/*
 * print
 * Prints this vector
 */
void StateVector::print() {
        printf("Index: %i\r\n", this->index);
        this->qsv.print();
}

/*
 * reduce
 * Removed unentanged qubits from this vector, splitting them off
 * into their own vectors
 */
void StateVector::reduce() {
        int i, j, k, firstValueFound, currentValue;
        double DOUBLE_ZERO = .000001;
        int zWidth = this->getWidth() - 1;
        bool isBitEntangled;

        QubitMap* m = QubitMap::getInstance();

        if ( this->getWidth() == 1 ) {
                return;
        }

        for ( i = this->getWidth() - 1; i >= 0; i-- ) {
                firstValueFound = -1;
                isBitEntangled = false;
                int zI = this->getWidth() - 1 - i;
                for (j = 0; j < this->qsv.getRows(); j++ ) {
                        currentValue = StateVector::isBitSet(j, zI);
                        if ( abs(this->qsv.get(0, j)) > DOUBLE_ZERO ) {
                                if ( firstValueFound == -1 ) {
                                        firstValueFound = currentValue;
                                } else {
                                        if ( currentValue != firstValueFound ){
                                                isBitEntangled = true;
                                        }
                                }
                        }
                }
                if ( !isBitEntangled && this->getWidth() > 1
                        && firstValueFound != -1 ) {
                        Matrix scratch( this->qsv.getCols(),
                                this->qsv.getRows() / 2 );

                        j = 0;
                        for ( k = 0; k < this->qsv.getRows(); k++ ) {
                                if (
                                (firstValueFound == 1
                                        && StateVector::isBitSet(k, zI) )
                                || (firstValueFound == 0
                                        && !StateVector::isBitSet(k, zI) )
                                ) {
                                        scratch.set(0, j, this->qsv.get(0, k));
                                        j++;
                                }
                        }
                        this->qsv = scratch;

                        for ( j = 0; j < m->numQubits(); j++ ) {
                                if ( m->getQubit(j)->v->index
                                        == this->index &&
                                        m->getQubit(j)->position == i ) {
                                        m->getQubit(j)->init();

                                        if ( firstValueFound == 1 ) {
                                                Matrix sigmaX = Matrix(2,2);
                                                sigmaX.set(0, 1, 1);
                                                sigmaX.set(1, 0, 1);
                                                m->getQubit(j)->applyMatrix(sigmaX);
                                        }
                                }
                                if ( m->getQubit(j)->v->index
                                        == this->index &&
                                        m->getQubit(j)->position > i ) {
                                        m->getQubit(j)->position--;
                                }
                        }
                }
        }
}

/*
 * isBitSet
 * @param index The index in the state vector
 * @param position The bit position
 * @return bool whether or not the bit position is set to zero or one
 */
bool StateVector::isBitSet(int index, int position) {
        return (int ( index / pow(2, (position)) )
                        % 2 == 1 );
}

/*
 * getAlpha
 * @param position
 * @returns alpha value for this bit
 * Determine the overall alpha (0) probability amplitude of the
 * bit position in this vector
 */
double StateVector::getAlpha(int position) {
        double alpha = 0.0;
        int i;

        for ( i = 0; i < this->qsv.getRows(); i++ ) {
                if ( !isBitSet(i, position) ) {
                        alpha += pow(real(qsv.get(0, i)), 2) +
                                pow(imag(qsv.get(0, i)), 2);
                }

        }

        return alpha;
}

/*
```

```
 * getBeta
 * @param position
 * @returns beta value for this bit
 * Determine the overall beta (1) probability amplitude of the
 * bit position in this vector
 */
double StateVector::getBeta(int position) {
        double beta = 0.0;
        int i;

        for ( i = 0; i < this->qsv.getRows(); i++ ) {
                if ( isBitSet(i, position) ) {
                        beta += pow(real(qsv.get(0, i)), 2) +
                                pow(imag(qsv.get(0, i)), 2);
                }
        }

        return sqrt(beta);
}

/*
 * measure
 * @param position the bit to measure
 * @return the measured value
 * Perform a pseudo random measurement of the bit given
 */
int StateVector::measure(int position) {
        int value = 0;
        double measurement = rand() / (float)RAND_MAX;

        if ( this->getAlpha(position) < measurement ) {
                value = 1;
        }

        return this->measure(position, value);
}

/*
 * measure
 * @param position the bit to measure
 * @param forceResult
 * @return the measured value
 * Perform a forced measurement of the bit given
 */
int StateVector::measure(int position, int forceResult) {
        this->measure(position, forceResult, true);
}

/*
 * measure
 * @param position the bit to measure
 * @param forceResult
 * @param propagate
 * @return the measured value
 * Perform a forced measurement of the bit given,
 * propagating the measuredment to any remote peers
 */
int StateVector::measure(int position, int forceResult,
        bool propagate) {
        int i;
        vector<int> peersNotified;
        Matrix initialState(this->qsv);

        int zPosition = this->getWidth() - 1 - position;

        if ( forceResult != 0 ) {
                forceResult = 1;
        }
        for ( i = 0; i < this->qsv.getRows(); i++ ) {
                if ( forceResult == 0 && this->isBitSet(i, zPosition) ) {
                        this->qsv.set(0, i, 0);
                }
                if ( forceResult == 1 && !this->isBitSet(i, zPosition) ) {
                        this->qsv.set(0, i, 0);
                }
        }

        if ( propagate ) {
                for ( i = 0; i < this->getWidth(); i++ ) {
                        if ( this->remoteQubits.at(i).remoteSystem
                                != "" ) {
                                QuantumChannel::ChannelService_client
                                csc(this->remoteQubits.at(i)
                                        .remoteSystem,
                                        this->remoteQubits.at(i)
                                        .remotePort);

                                csc.SendMeasurementMessage(
                                        this->getIndex(),
                                        position, initialState, forceResult);
                        }
                }
        }

        this->reduce();
        this->normalize();
        return forceResult;
}

/*
 * fromDensity
 * @param rho the density matrix
 * Convert a density matrix to a state vector
 */
void StateVector::fromDensity(Matrix rho) {
        int i, j;
        int N = rho.getRows();
        double A[N*N];
        double V[N*N];
        double D[N];
        int it_num;
        int rot_num;

        if ( rho.getRows() != rho.getCols()
                || rho.getRows() != this->qsv.getRows() ) {
                return;
        }

        for ( i = 0; i < this->qsv.getRows(); i++ ) {
                this->qsv.set(0, i, 0);
        }

        for ( i = 0; i < N; i++ ) {
                for ( j = 0; j < N; j++ ) {
                        A[i*N + j] = real(rho.get(i, j));
                }
        }
        jacobi_eigenvalue(N, A, 20, V, D, it_num, rot_num);
        for ( i = 0; i < N; i++ ) {
                for ( j = 0; j < N; j++ ) {
                        complex<double> temp = this->qsv.get(0, j);
                        temp += D[i] * V[i*N+j];
                        this->qsv.set(0, j, temp);
                }
        }
```

```
}

/*
 * toDensity
 * @return the density matrix
 * Convert this state vector to a density matrix
 */
Matrix StateVector::toDensity() {
        int i, j;
        Matrix rho(this->qsv.getRows(), this->qsv.getRows());

        for ( i = 0; i < this->qsv.getRows(); i++ ) {
                for ( j = 0; j < this->qsv.getRows(); j++ ) {
                        complex<double> temp = rho.get(i, j);
                        temp += this->qsv.get(i, 0) * this->qsv.get(j, 0);
                        rho.set(i, j, temp);
                }
        }
        return rho;
}

/*
 * normalize
 * Normalize probability amplitudes across the state vector
 */
void StateVector::normalize() {
        int i;
        double realPart = 0.0;
        double imagPart = 0.0;

        double total = 0.0;

        for ( i = 0; i < this->qsv.getRows(); i++ ) {
                realPart = real(this->qsv.get(0, i));
                imagPart = imag(this->qsv.get(0, i));
                total += realPart*realPart + imagPart*imagPart;
        }

        for ( i = 0; i < this->qsv.getRows(); i++ ) {
                realPart = real(this->qsv.get(0, i));
                imagPart = imag(this->qsv.get(0, i));

                this->qsv.set(0, i,
                        sqrt((realPart*realPart + imagPart*imagPart) / total));
        }
}

void sync() {

}

/*
 * toMatrix
 * @return this vector as a matrix
 */
Matrix StateVector::toMatrix() {
        return this->qsv;
}

/*
 * replay
 * Replay this state vector's history
 */
void StateVector::replay() {
        int i = 0;

        for ( i = 0; i < this->opHistory.size(); i++ ) {
                this->applyOperation(
                    this->opHistory.at(i).getOperation(),
                    this->opHistory.at(i).getArgs(),
                    false
                );
            }
            this->opHistory.clear();
        }

    }
```

## *A.2 Teleportation Runnable*

```
/*
 * teleportClientRunnable.cpp
 */
//STD C++ includes
#include <memory>
#include <stdio.h>
#include <string>
#include <utility>
#include <unistd.h>
#include <vector>

//Qoosim includes
#include "channelService_client.h"
#include "gates.h"
#include "system.h"
#include "qubit.h"

//Runnable header
#include "teleportClientRunnable.h"

using namespace std;
namespace Quantum {
/*
 * Run
 * @param none
 * @returns none
 * @sides teleports a qubit to a remote peer
 */
void TeleportClientRunnable::Run() {
        //Create three qubits in initial state |0>
        shared_ptr<Qubit> q1 = Qubit::create();
        shared_ptr<Qubit> q2 = Qubit::create();
        shared_ptr<Qubit> psi = Qubit::create();

        //Create a hadamard gate
        Hadamard h;
        //Create a y-rotational gate
        Ry r(M_PI/3);
        //Create a Cnot gate
        CNot cn;

        //Get the system object
        System* sys = System::getInstance();
        //Create a connection to the server
        QuantumChannel::ChannelService_client csc(
                this->serverIP,
                this->serverPort);

        //Rotate the payload qubit (this could be any operation)
        psi->applyMatrix(r);

        //Violate the laws of physics and print psi
        psi->print();

        //Apply a hadamard to q1, placing the qubit in a superposition
        q1->applyMatrix(h);

        //Entangle q1 and q2 by applying a Cnot gate
        vector< shared_ptr<Qubit> > inputs;
        inputs.push_back(q1);
        inputs.push_back(q2);
        q1->v->applyOperation(cn, inputs);

        //Send q2 to the server
        csc.SendQubit(q2);

        //Wait two seconds, for the server to get the qubit
        sleep(2);

        //Add the payload qubit to the entanglement by using Cnot
        inputs.at(0) = psi;
        inputs.at(1) = q1;
        q1->v->applyOperation(cn, inputs);

        //Apply a hadamard gate to the payload qubit
        psi->applyMatrix(h);

        //Measure q1 and psi
        int q1_result = q1->measure();
        int psi_result = psi->measure();
        printf("q1: %i, psi: %i\r\n", q1_result, psi_result);

        /*
         * Normally, here, we would send q1_result and psi_result to the server
         * so that she could perform the encoded post-teleportation operation
         */

        sys->stopServer();
}
}
```

```
/*
 * teleportServerRunnable.cpp
 */
//STD C++ includes
#include <memory>
#include <unistd.h>
#include <utility>
#include <stdio.h>

//Qoosim includes
#include "gates.h"
#include "qubit.h"
#include "qubitMap.h"
#include "system.h"

//runnable header
#include "teleportServerRunnable.h"

using namespace std;
namespace Quantum {
/*
 * Run
 * @param none
 * @return none
 */
void TeleportServerRunnable::Run() {
        //Get the system object
        System* sys = System::getInstance();

        //Get the memory map object
        QubitMap* qm = QubitMap::getInstance();

        //Wait for a message to come in
        while ( sys->isMessageQueueEmpty() ) {
        }

        //If the message is a quantum message, we assume it's our half of q1q2
        if ( sys->nextMessageType()
                == SystemMessage::QUANTUM_DATA_RECEIVED ) {
                //Get the address of the received qubit
                int address = sys->nextMessage();
                //Retrieve the qubit from the address
                shared_ptr<Qubit> q2 = qm->getQubit(address);

                //Wait 5 seconds for the client to do her operations
                sleep(5);

                //Violate the laws of physics and print the state of the qubit
                q2->print();
        }

        sys->stopServer();
}
}
```

## A.3 Kak06 Runnable

```
/*
 * kakinitiator_runnable.cpp
 */
#include <unistd.h>
#include <time.h>
#include <math.h>
#include <stdio.h>
#include <string>
#include <vector>

#include "system.h"
#include "qubit.h"
#include "qubitMap.h"
#include "gates.h"
#include "channelService_client.h"
#include "kakinitiator_runnable.h"
#include "kakresponder_runnable.h"

using namespace std;

namespace Quantum {
void KakInitiator_Runnable::Run() {
        int i;
        //count of errors (bits mismatched)
        int errors = 0;
        //vector for storing bits we sent
        vector<int> bits;
        //vector for storing rotational angles used to encode bits
        vector<float> rotations;

        //vector for storing qubits (scratch space)
        vector< shared_ptr<Qubit> > q;
        //quantum Sigma X gate
        SigmaX sx;

        //seed random, unique to this process
        srand(time(NULL) / getpid());

        //get a reference to the system and qubit map
        System* sys = System::getInstance();
        QubitMap* qm = QubitMap::getInstance();

        //set up connection to the server
        QuantumChannel::ChannelService_client csc(this->serverIP,
                this->serverPort);

        //this implementation sends a key of fixed size, with all
        //key-bits sent in serial
        for ( i = 0; i < KAK_KEY_SIZE; i++ ) {
                //generate a random bit value {0, 1}
                bits.push_back(round(rand() / (float)RAND_MAX));
                //generate a random rotation value 0 < rotation <= 1
                rotations.push_back((rand() / (float)RAND_MAX) * M_PI/8.0);

                //create a new qubit (initial value |0>)
                q.push_back(Qubit::create());
                //if the bit is to be a 1, apply a sigma-x gate
                if ( bits.at(i) == 1 ) {
                        q.at(i)->applyMatrix(sx);
                }

                //create a rotational gate
                Ry r = Ry(rotations.at(i));
                //apply the rotational gate
                q.at(i)->applyMatrix(r);

                //send the qubit
                csc.SendQubit(q.at(i));
        }

        //receive returned qubits
        for ( i = 0; i < KAK_KEY_SIZE; i++ ) {
                //wait for the system to receive the qubit and put
                //a message on the queue
                while ( sys->isMessageQueueEmpty() ) {
                }
                //get the map address of the received qubit
                int address = sys->nextMessage();
                //store the received qubit in the scratch vector
                q.at(i) = qm->getQubit(address);
        }

        //counter-rotate the returned qubits and resend
        for ( i = 0; i < KAK_KEY_SIZE; i++ ) {
                //create the counter-rotational gate for this qubit
                Ry r = Ry(-1.0*rotations.at(i));
                //apply the counter-rotational gate
                q.at(i)->applyMatrix(r);
                //return the qubit
                csc.SendQubit(q.at(i));
        }

        //this part is outside of Kak's protocol
        //we're going to receive the results from our peer
        //and count how many errors were introduced
        for ( i = 0; i < KAK_KEY_SIZE; i++ ) {
                //wait for a message on the queue
                while ( sys->isMessageQueueEmpty() ) {
                }

                //get the map address of the received bit
                int address = sys->nextMessage();
                //return the data stored at the given address
                string data = sys->getClassicData(address);
                //if the data received doesn't match our original
                //bit vector, count it as an error
                if ( data == to_string(bits.at(i)) ) {
                        errors++;
                }
        }

        //output error statistics
        printf("%i, %i\r\n", KAK_KEY_SIZE, errors);
        //stop the server thread
        sys->stopServer();
}
}
```

```
/*
 * kakresponder_runnable.cpp
 */
#include <unistd.h>
#include <time.h>
#include <math.h>
#include <stdio.h>
#include <string>
#include <vector>

#include "system.h"
#include "qubit.h"
#include "qubitMap.h"
#include "gates.h"
#include "channelService_client.h"
#include "kakinitiator_runnable.h"
#include "kakresponder_runnable.h"

using namespace std;

namespace Quantum {
void KakResponder_Runnable::Run() {
        int i;
        //to store the client's ip/port once they connect
        string peerIP = "";
        int peerServicePort = 0;

        //vector for storing the bits we decode
        vector<int> keyMaterial;
        //vector for storing rotation bits we use to encode
        vector<float> rotations;
        //vector for storing qubits (scratch space)
        vector< shared_ptr<Qubit> > q;

        //seed random, unique to this process
        srand(time(NULL) / getpid());

        //get a reference to the system and qubit map
        System* sys = System::getInstance();
        QubitMap* qm = QubitMap::getInstance();
        //declared for client callback connection
        QuantumChannel::ChannelService_client* csc;

        //receive qubits rotated by our peer
        for ( i = 0; i < KAK_KEY_SIZE; i++ ) {
                //generate a random rotation value 0 < rotation <= 1
                rotations.push_back((rand() / (float)RAND_MAX) * M_PI / 8.0);
                //wait for the system to receive the qubit and put
                //a message on the queue
                while ( sys->isMessageQueueEmpty() ) {
                }
                //get the map address of the received qubit
                int address = sys->nextMessage();
                //store the received qubit in the scratch vector
                q.push_back(qm->getQubit(address));

                //if we haven't yet established a callback connection to the
                //client, do so now
                if ( peerIP == "" ) {
                        peerIP = q.at(i)->origin.peerIP;
                        peerServicePort = q.at(i)->origin.peerServicePort;
                        csc = new QuantumChannel::ChannelService_client(
                                        peerIP, peerServicePort);
                }
        }

        //rotate the returned qubits and send back
        for ( i = 0; i < KAK_KEY_SIZE; i++ ) {
                //create a rotational gate
                Ry r = Ry(rotations.at(i));
                //apply the rotational gate
                q.at(i)->applyMatrix(r);
                //send the qubit
                csc->SendQubit(q.at(i));
        }

        //receive qubits which our peer has counter-rotated
        for ( i = 0; i < KAK_KEY_SIZE; i++ ) {
                //wait for the system to receive the qubit and put
                //a message on the queue
                while ( sys->isMessageQueueEmpty() ) {
                }
                //get the map address of the received qubit
                int address = sys->nextMessage();
                //store the received qubit in the scratch vector
                q.at(i) = qm->getQubit(address);
        }

        //counter-rotate the returned qubits and measure
        for ( i = 0; i < KAK_KEY_SIZE; i++ ) {
                //create the counter-rotational gate for this qubit
                Ry r = Ry(-1 * rotations.at(i));
                //apply the counter-rotational gate
                q.at(i)->applyMatrix(r);
                //measure the qubit
                string data = to_string(q.at(i)->measure());

                //this part is outside of Kak's protocol
                //we're going to send the results to our peer
                //and they'll count how many errors were introduced
                csc->SendClassicData(data);
        }

        //stop the server thread
        sys->stopServer();
}
}
```