Will Bidle
Physics 488
April 30, 2020

Quantum Teleportation

1 INTRODUCTION

By far one of the most interesting and mysterious concepts to come from quantum computing is the quantum teleportation algorithm. The algorithm itself is remarkably simple, yet it does something extremely profound in the sense that it transfers information from one place to another without ever exchanging information between the two locations. Even though nothing is physically 'teleported,' we are still able to send qubits without access to a quantum channel. We can extend this concept to teleport entangled bits and create a chain of teleportation's to separate entangled pairs over very large distances. This allows for some pretty amazing real-world applications, from ultra-precise clocks, to uncrackable encryption codes. I analyze the overall concept in a few ways, starting with a simple teleportation of pure and mixed states, and then move toward teleporting entangled qubits. Each circuit is constructed in Python by utilizing IBM's Qiskit software, and a supplementary Jupyter Notebook file is attached with the code used to get my results. The written algorithms were run through both a classical simulator as well as an IBM quantum computer to compare the respective performances.

2 TELEPORTING QUBITS

2.1 METHODOLOGY

Teleporting the state of a qubit to another without actually making observations amounts to using three qubits, two of which are entangled while the other is the state to be teleported. The state we desire to teleport will be designated as qubit A, while the entangled qubits will be referred to as B and C. By measuring the 'sameness' of qubits A and B (which amounts to measurement in the Bell basis), we can then perform corresponding operation on qubit C to make sure the correct state was transferred. When running the data through both the classical simulator and quantum computers, I used 8192 (2^{13}) trials. This number was chosen because it is high enough to reduce errors that come with statistical measurements. The statistics of large numbers tell us that a high enough number of measurements causes the uncertainty of our results to go to zero, leaving us with clean teleportation's with small errors. I expect this to be the case in the classical simulator, since classical systems behave predictably, but not the case in the quantum simulator, since quantum systems behave unpredictably and the quantum computers to be used are not perfect.

2.2 CODE AND DESIGN

The design of the teleportation circuit is fairly straightforward and can be broken down into three sections (separated by barriers), as seen in Figure 1. The first section deals with initializing qubit A to the state that is to be teleported, as well as creating an entangled pair $|\Phi_+\rangle_{bc}$[1] between qubits B and C. To do this, qubit B is hit with a Hadamard gate, then a CNOT is applied to qubits B (control) and C (target). The second section deals with measuring qubits A and B in the Bell basis. This amounts to using a CNOT gate on qubits A (control) and B (target), and then using a Hadamard on qubit A. Qubits A and B are then measured to see what Bell state they are in, and the result is used to perform operations on qubit C. The third and final segment deals with performing unitary operations on qubit C, depending on the results of the Bell basis measurement. If we measured that qubit B was $|1\rangle$, we apply Z to qubit C, and if we measured that qubit A was $|1\rangle$, we apply X to qubit C. Assuming everything was done correctly, then when we measure qubit C, we should expect to see the state that qubit A was in.

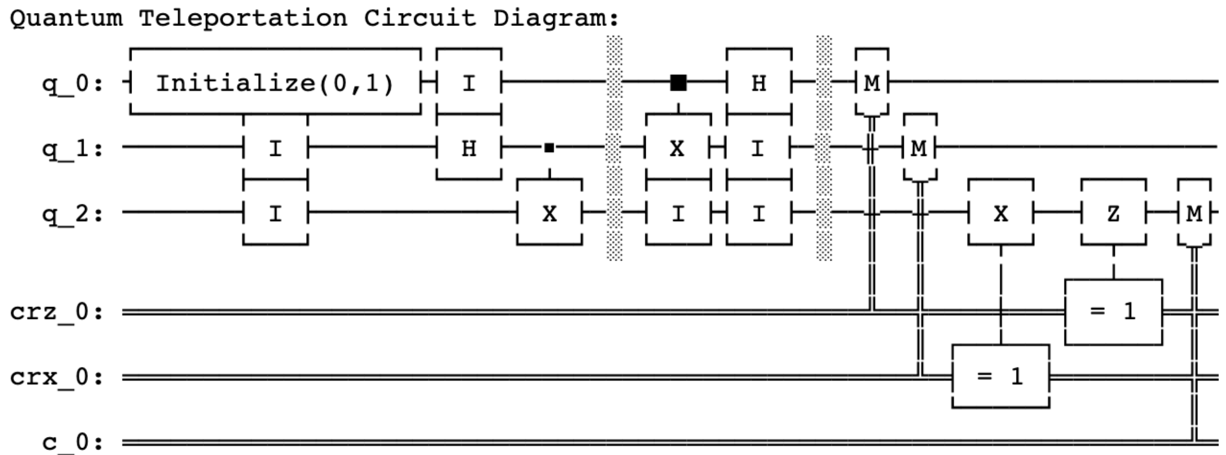

Figure 1. Circuit Diagram for Teleportation Qubits

2.3 CLASSICAL RESULTS

For a simple state for qubit A, such $|1\rangle$, we expect to see 100% of our results come out with $|1\rangle$ for qubit C, since we initially have zero chance of seeing $|0\rangle$ in qubit A. The initial state $|1\rangle$ was then run through the circuit and measured for the number of trials specified, and the results can be seen in Figure 2 below. Qubit C is represented by the bottommost number in each result, and we can easily see that for each result, $|1\rangle$ was successfully teleported to C every time. We end up seeing something similar if we had started with $|0\rangle$ for qubit A and can conclude that the teleportation protocol works[2] for pure states.

[1] $|\Phi_+\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$

[2] At least in our classical simulation.

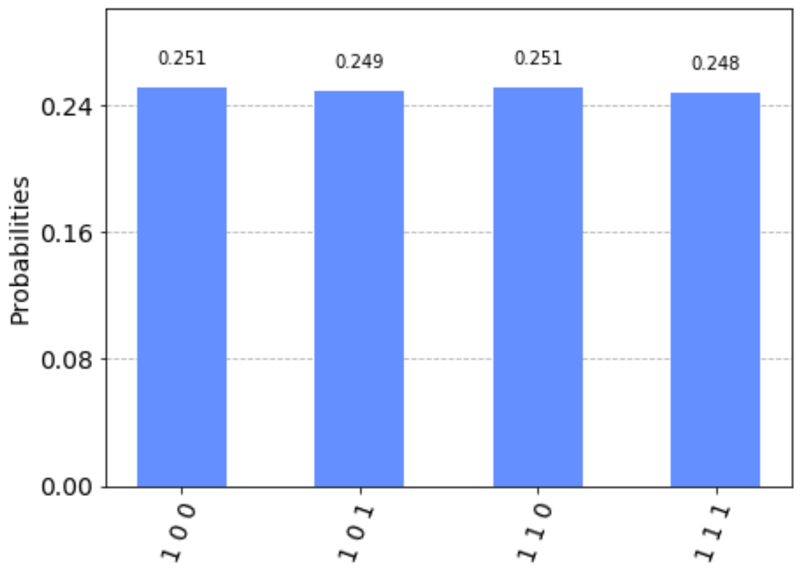

Figure 2. Classical results for teleporting the state $|1\rangle$

What about for a general mixed state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$? To test this, we will use the $|+\rangle$ state, where $|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. In this case, we should expect to see half the results be $|0\rangle_c$ and the other half be $|1\rangle_c$. There may be some statistical error in our results due to the way we are simulating a distribution of results, however we can expect the deviation to be small since the number of measurements we are running is high. Figure 3 below shows the results for when the algorithm is run with the initial mixed state. At first glance we can see that, on average, we result with an equal number of $|0\rangle_c$ and $|1\rangle_c$ as predicted[3]. Counting up the exact numbers reveals that $|+\rangle$ was teleported with less than 1% error to qubit C, which is exactly what we hoped for.

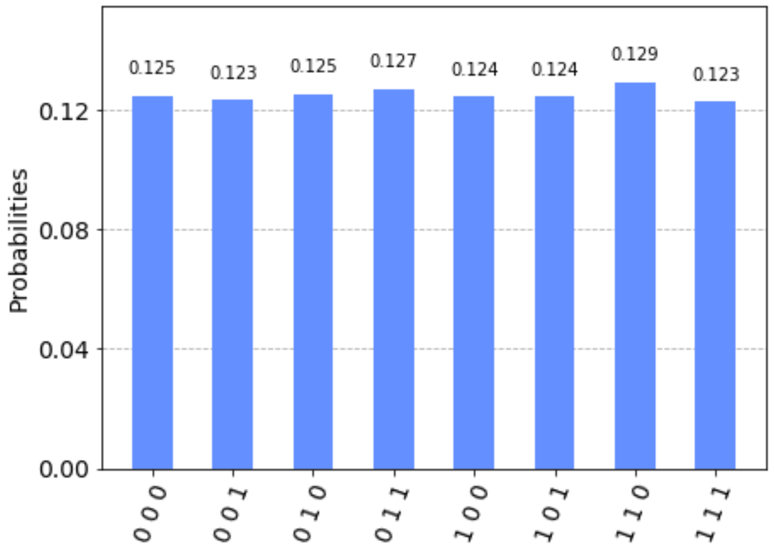

Figure 3. Classical results for teleporting the state $|+\rangle$

[3] Again, the bottommost number in the figure represents qubit C.

2.4 QUANTUM RESULTS

The results when we run the initial states of $|1\rangle$ and $|+\rangle$ through a quantum computer can be seen in Figures 4 and 5 respectively. The data shows the probability of measuring $|0\rangle$ or $|1\rangle$ in qubit C, essentially telling us how well the process worked. As expected, the quantum computer does not give us the perfect results achieved in the classical simulation for the same number of trials. Even for the simple $|1\rangle$, state, our quantum computer gives us a nonzero probability of seeing $|0\rangle$, indicating some definite error. The difference here is that no matter how many times we run our simulation, we will always find some error in the quantum computers, due to inaccuracies in the gates and detectors.

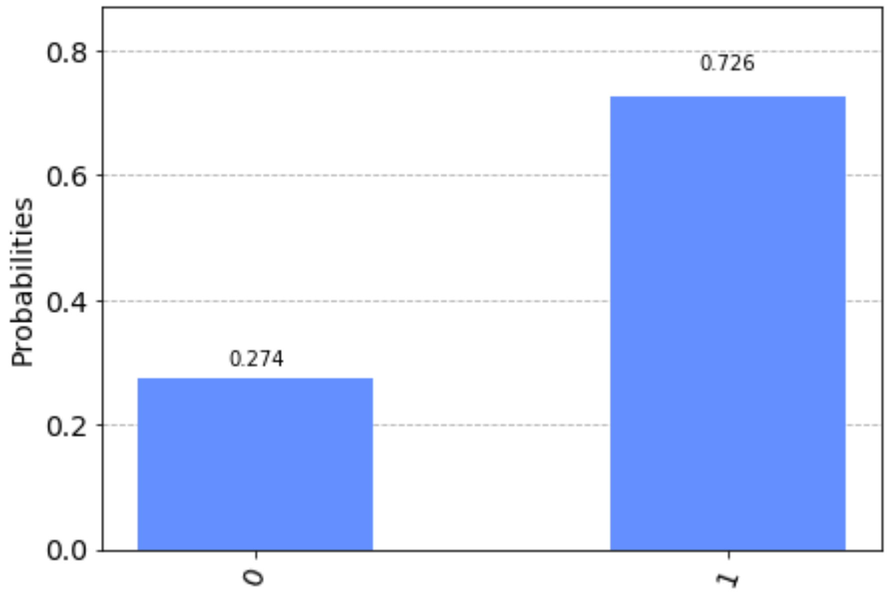

Figure 4. Quantum results for teleporting the state $|1\rangle$

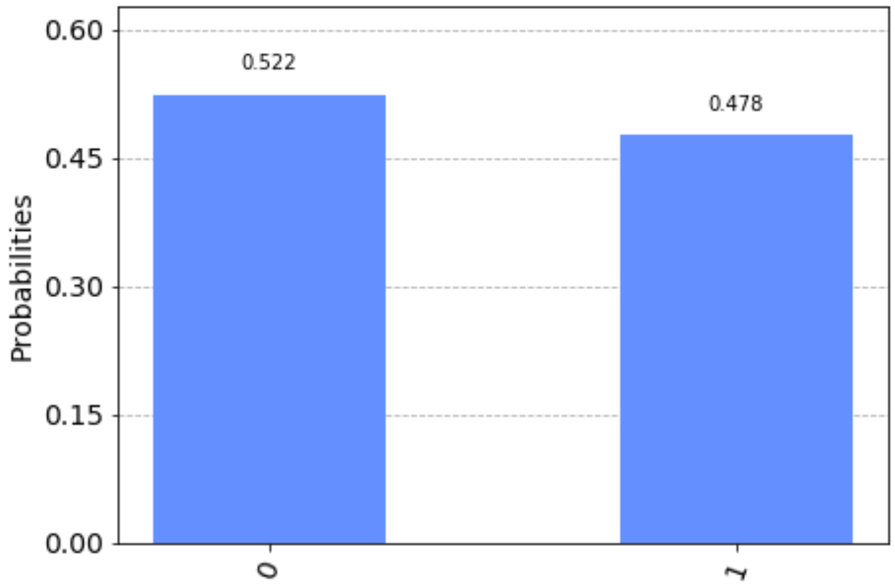

Figure 5. Quantum results for teleporting the state $|+\rangle$

3 TELEPORTING ENTANGLED QUBITS

3.1 METHODOLOGY

Teleporting the state of entangled qubits amounts to using two entangled pairs of qubits. By performing specific operations on the first entangled pair, containing qubits A and B, and the second entangled pair, containing qubits C and D, we can teleport A to D, entangling this new pair together with the initial state that A and B had. Similar to that of the single qubit teleportation algorithm, we measure the ‘sameness’ of qubits B and C (again amounting to a measurement in the Bell basis), and then can perform corresponding operation on qubit D to make sure the correct state was transferred. The same 8192 (2^{13}) trials were used for the same reason as explain in section 2.1. The number chosen because is high enough to reduce errors that come with statistical measurements. Much like in the previous experiment, I expected to see almost perfect results for the classical simulations and expected to see error for the quantum computer.

3.2 CODE AND DESIGN

The design of the entanglement teleportation circuit is actually very similar to the simple teleportation circuit, with a few minor tweaks (see Figure 6). The initial state we start with is $|\Psi_-\rangle_{ab} \otimes |\Phi_+\rangle_{cd}$[4], where the goal is to transfer the entanglement between qubits A and B to qubits A and D, i.e. result with $|\Psi_-\rangle_{ad}$. Unlike the single qubit teleportation, we will end up with an entangled state as our final result. The first segment deals with preparing the two entangled states $|\Psi_-\rangle_{ab}$ and $|\Phi_+\rangle_{cd}$. Qubits C and D are entangled together to make the $|\Phi_+\rangle_{cd}$ Bell state by sending qubit C through a Hadamard gate, and then applying a CNOT gate to qubits C (control) and D (target). The same thing is done for qubits A and B, however in order to turn $|\Phi_+\rangle_{ab}$ into $|\Psi_-\rangle_{ab}$, a Z and X gate are used on qubits A and B respectively. The second segment deals with measuring qubits B and C in the Bell basis. This amounts to using a CNOT gate on qubits B (control) and C (target), and then sending B through a Hadamard. Qubits B and C are then measured to see what Bell state they are in, and the result is used to perform operations on qubit D. The third and final segment deals with performing unitary operations on qubit D, depending on the results of the Bell basis measurement. If we measured that qubit C was $|1\rangle$, we apply Z to qubit D, and if we measured that qubit B was $|1\rangle$, we apply X to qubit D. Once the operations are performed, we measure the result of qubits A and D to see the results. If the job was done correctly, then we should expect to see $|\Psi_-\rangle_{ad}$.

[4] $|\Psi_-\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$ and $|\Phi_+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$

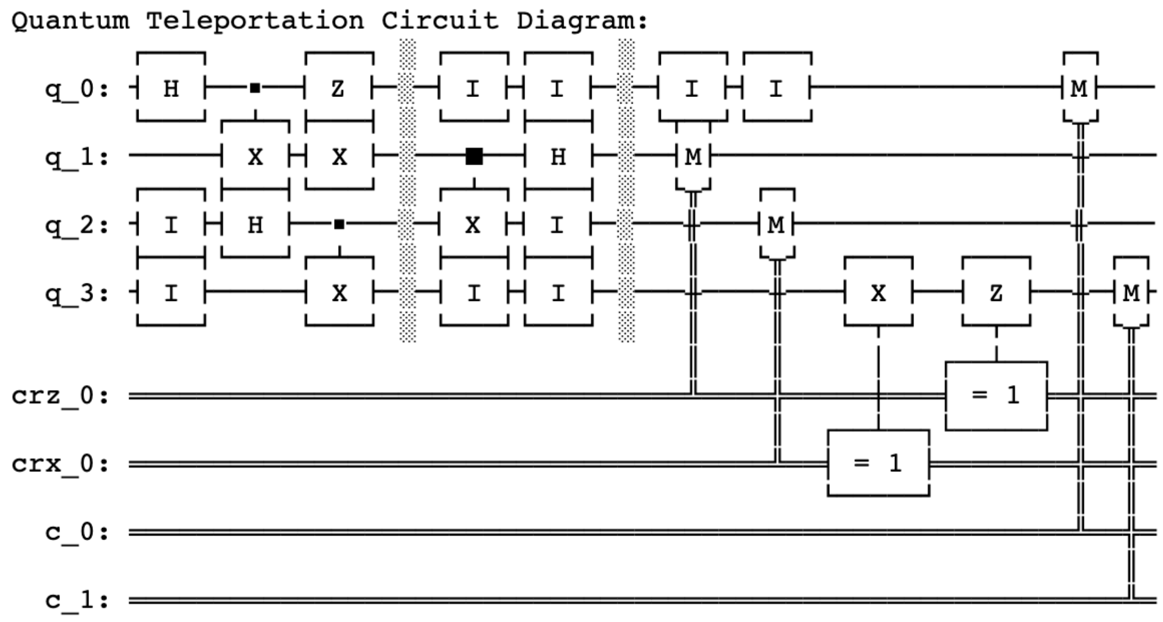

Figure 6. Circuit Diagram for Teleporting Entangled Qubits

3.3 CLASSICAL RESULTS

As explained, we set up the state $|\Psi_{-}\rangle_{ab} \otimes |\Phi_{+}\rangle_{cd}$, and measurement the results of qubit A and qubit D. Since $|\Psi_{-}\rangle = \frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|10\rangle$ and we are looking to find $|\Psi_{-}\rangle_{ad}$, we should expect to see half the results be $|0\rangle_a|1\rangle_d$ and the other half be $|1\rangle_a|0\rangle_d$. Figure 7 below shows the results for when the state $|\Psi_{-}\rangle_{ab} \otimes |\Phi_{+}\rangle_{cd}$ is run through the algorithm on a classical simulator. At first glance we can see that, on average, we result with an equal number of $|0\rangle_a|1\rangle_d$ and $|1\rangle_a|0\rangle_d$. The exact results tell us that $|\Psi_{-}\rangle_{ab}$ was teleported to A and D with less than 1% error, similar to when we did the mixed state $|+\rangle$. The small amount of error again comes from simulating statistical data.

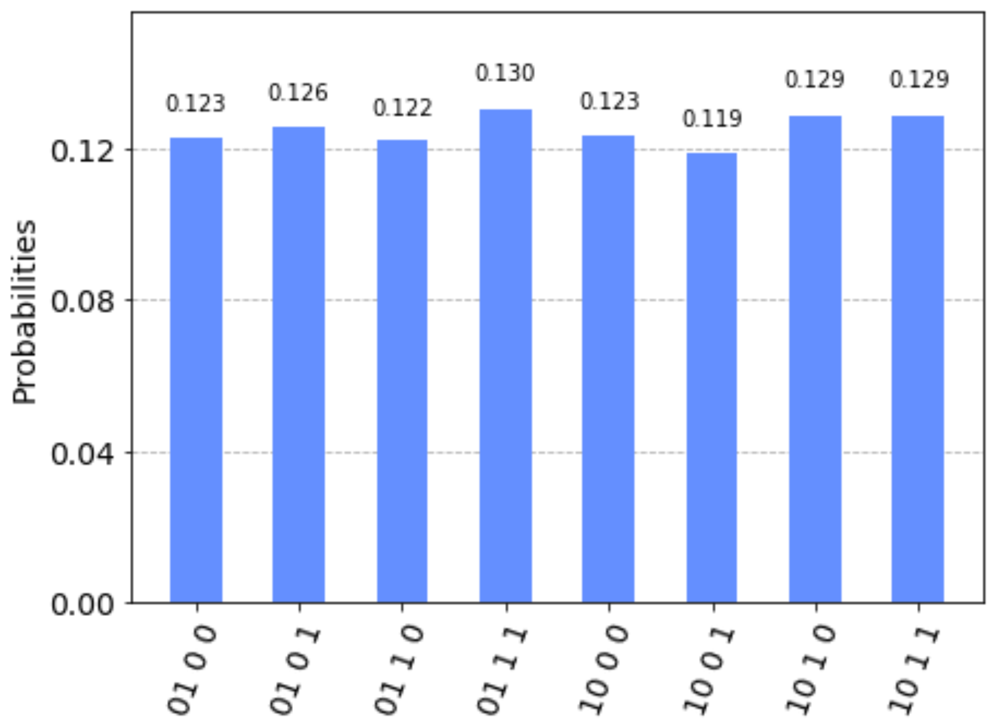

Figure 7. Classical results for teleporting the state $|\Psi_{-}\rangle$

3.4 QUANTUM RESULTS

The state $|\Psi_{-}\rangle_{ab} \otimes |\Phi_{+}\rangle_{cd}$ was then run through a quantum computer, and the results can be seen in Figure 8. Again, as expected, the quantum computer does not give us the perfect results achieved in the classical simulation for the same number of trials. In general, it actually gives us a majority of $|0\rangle_a|1\rangle_d$ and $|1\rangle_a|0\rangle_d$, signifying a successful teleportation of $|\Psi_{-}\rangle$. However we can see that there are some cases where we result with $|0\rangle_a|0\rangle_d$ or $|1\rangle_a|1\rangle_d$, which are not possible states present in $|\Psi_{-}\rangle$. These scenarios correspond to errors within the quantum computer, whether through an imperfect application of a gate or through interaction with the environment.

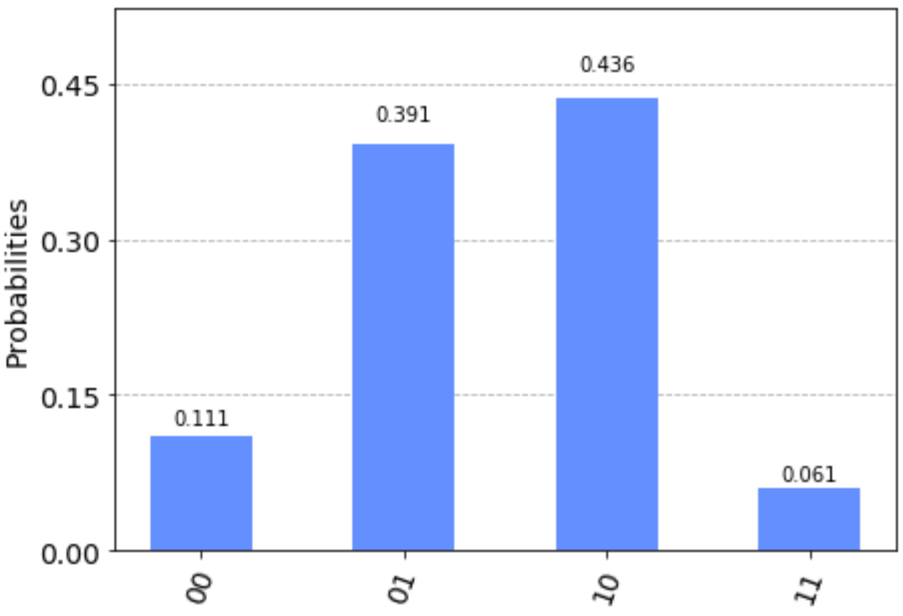

Figure 8. Quantum results for teleporting the state $|\Psi\_\rangle$

4 TELEPORTING TWICE

4.1 METHODOLOGY

What if we were to teleport a qubit as in section 2, and then teleport it again? The methodology is identical to teleporting a qubit once, with the subtle difference of doing it all again immediately after. Again, I expected the results to be fairly accurate for the classical case, and this time to have introduced much more error in the quantum computation. Since I am essentially just doing the same process twice, I expected to have about twice the error as doing it just once. Figure 9 shows the updated circuit diagram.

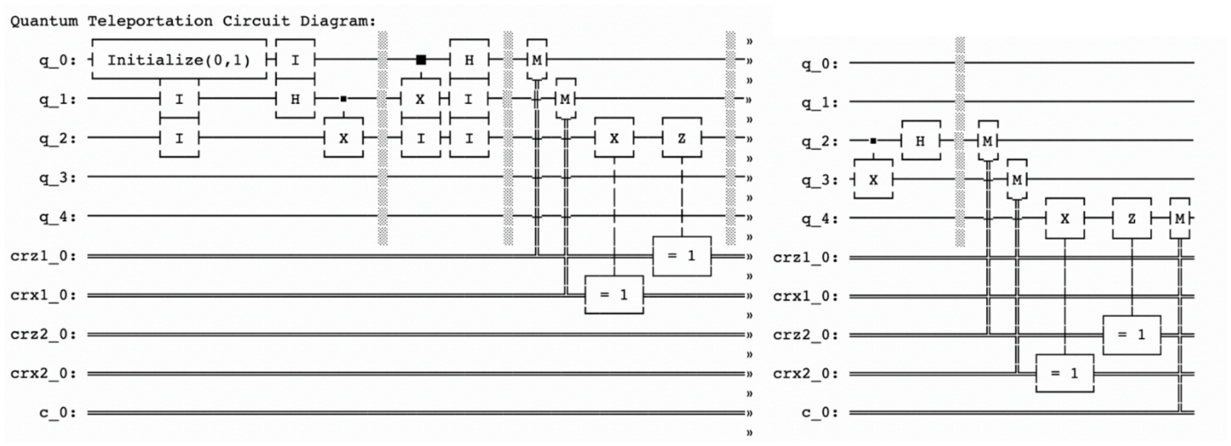

Figure 9. Circuit Diagram for Twice

4.2 CLASSICAL RESULTS

Using the same $|+\rangle$ test state as used in section 2, we now hope to teleport across two entangled pairs instead of one. Once again, we should expect to see half the results be $|0\rangle$ and the other half be $|1\rangle$. Figure 10 below shows the results for when the algorithm is run with the initial mixed state. Perhaps unsurprisingly we can see that, on average, we result with an equal number of $|0\rangle$ and $|1\rangle$ as predicted when we look at the bottom most number in each result. Counting up the exact probabilities reveals that $|+\rangle$ was again teleported with less than 1% error, indicating a successful double teleportation.

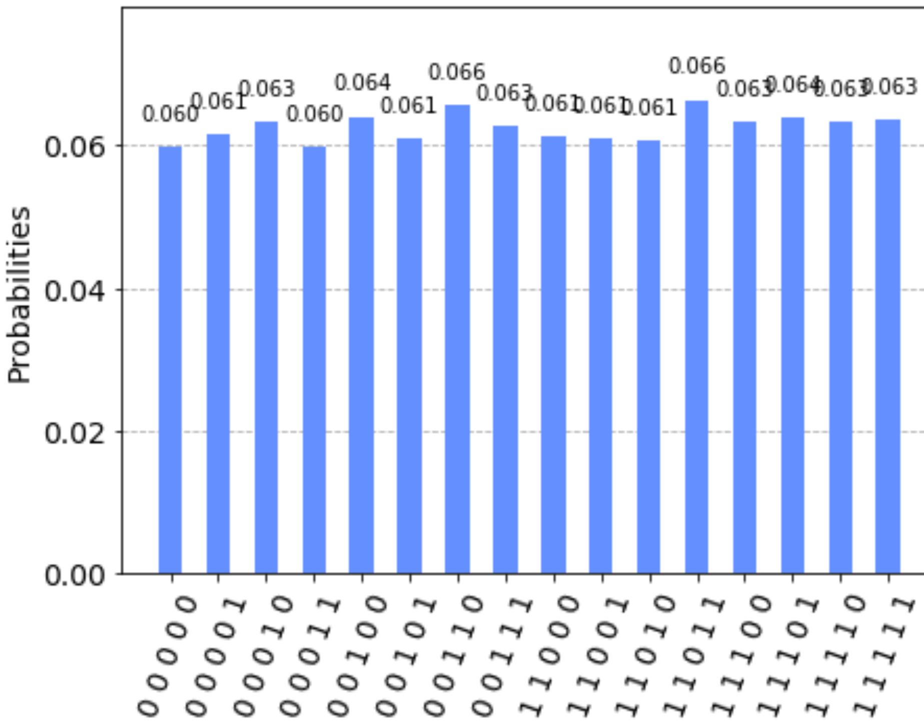

Figure 10. Classical results for teleporting the state $|+\rangle$

4.3 QUANTUM RESULTS

Quite surprisingly, when the circuit was run on a quantum computer, the error in the final results was not as bad as I thought it would be. It can be seen in Figure 11 that there is almost as good a split of our data as there was in Figure 5. Since we would essentially be doing the operation twice, I had anticipated that there would be about twice the error in our data, whether it was a skew towards $|0\rangle$ or towards $|1\rangle$. Despite a more complex system, the quantum computer was able to perform the operation fairly accurately.

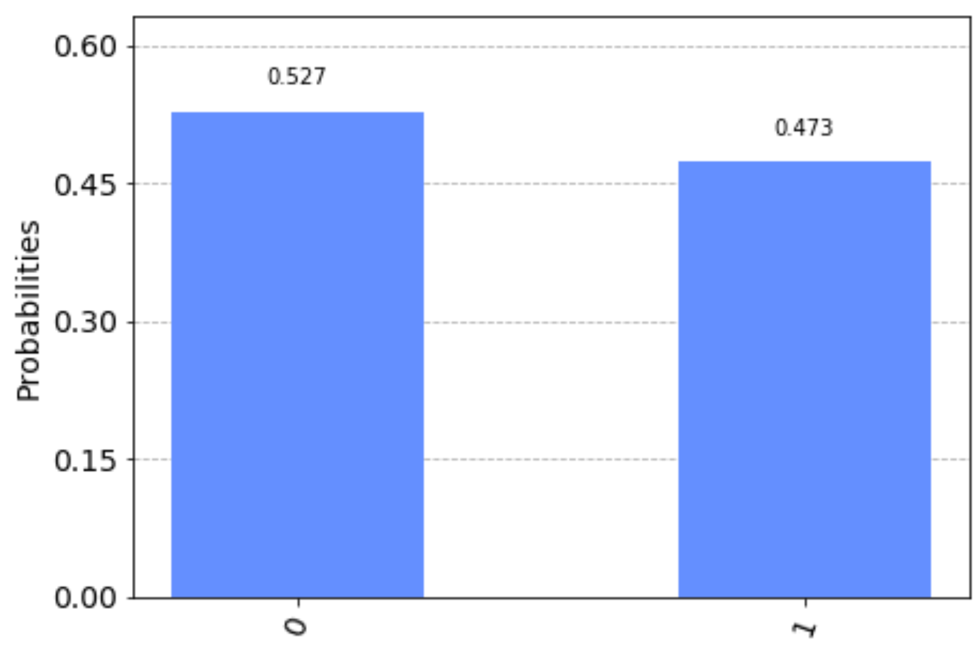

Figure 11. Quantum results for teleporting the state $|+\rangle$

5 CONCLUSIONS

Throughout these experiments, it is evident that the quantum computers used do not produce the ideal results we desired. Even though in theory our protocols should work perfectly, it is very difficult to have an ideal situation. Designing and creating a quantum computer that makes perfect measurements is extremely difficult, and qubits subject to even the smallest interaction with the environment can lead to drastic end results. As seen in Figure 8 of section 3.4, the $|0\rangle_a|0\rangle_d$ and $|1\rangle_a|1\rangle_d$ states are not present in the initial state $|\Psi_-\rangle_{ab}$, however somewhere along the experiment some error was introduced to the qubits, leading to these final states. It is worth noting, however, that while the results were not perfect, the quantum computers correctly performed the operation a majority of the time. There is definitely a lot of work to be done on the accuracy of these computers, but simple results such as these show the potential for such a powerful communication system.

In [ ]:
```
from qiskit import ClassicalRegister, QuantumRegister, QuantumCircuit, e
xecute, Aer
from qiskit import IBMQ
from qiskit.tools.visualization import circuit_drawer
from qiskit.visualization import plot_histogram
from qiskit.providers.ibmq import least_busy
import numpy as np
import random

IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
```

Teleporting Qubits

```
In [ ]: def QuantumTeleport(initial_state):
            # declare the 3 qubits; one to teleport and the other two to entangl
        e
            q = QuantumRegister(3, name = 'q')

            # initialize the classical registers

            # crz checks whether to operate Z on the 3rd qubit after the gates
            crz = ClassicalRegister(1, name = 'crz')
            # crx check whether to operate X on the 3rd qubit after the gates
            crx = ClassicalRegister(1, name = 'crx')
            # c is the final measurement of the 3rd quibit
            c = ClassicalRegister(1, name = 'c')

            # initialize our quantum circuit
            qc = QuantumCircuit(q,crz,crx,c, name ='qc')

            # store the initial state into qubit 1, which will be teleported to
         qubit 3
            qc.initialize(initial_state, 0)
            qc.i(q[1]) # makes the visuals line up
            qc.i(q[2]) # makes the visuals line up

            # create a bell pair
            qc.i(q[0]) # makes the visuals line up
            qc.h(q[1]) # applies a hadamard to the second qubit
            qc.cx(q[1],q[2]) # applies a CNOT to the second(control) and third(t
        arget) qubits to entangle them together
            qc.barrier() # separates the stages of the circuit

            # prepare to teleport
            qc.cx(q[0],q[1]) # applies a CNOT to the first(control) and second(t
        arget) qubits
            qc.h(q[0]) # applies a hadamard to the first qubit
            qc.i(q[1]) # makes the visuals line up
            qc.i(q[2]) # makes the visuals line up
            qc.i(q[2]) # makes the visuals line up
            qc.barrier() # separates the stages of the circuit

            # perform measurement on the first and second qubits
            qc.measure(q[0],crz) # measure qubit 1
            qc.measure(q[1],crx) # measure qubit 2

            # make necessary changes to the third qubit depending on the state o
        f the first two
            qc.x(q[2]).c_if(crx, 1) # Apply X gate to qubit 3 if qubit 2 is in s
        tate |1>
            qc.z(q[2]).c_if(crz, 1) # Apply Z gate to qubit 3 if qubit 1 is in s
        tate |1>

            # perform measurement on the third qubit to see if we teleported cor
        rectly
            qc.measure(q[2],c) # measure qubit 3

            # use the classical simulator
            backend = Aer.get_backend('qasm_simulator') # classical simulator
```

```
    counts = execute(qc, backend, shots=16384).result().get_counts() # s
imulate a large set of data

    # draw the circuit
    print('Quantum Teleportation Circuit Diagram:')
    print(circuit_drawer(qc)) # ascii write out

    # calculate the probability amplitudes we result with on average
    total_0 = 0 # stores the total prob we find |0>
    total_1 = 0 # stores the total prob we find |1>
    for item in counts:
        if item[0] == '0': # check if we measured a |0> for the teleport
ed qubit
            total_0 += counts[item]/16384 # get the probability and add
 to total number of |0>'s
        else: # check if we measured a |1> for the teleported qubit
            total_1 += counts[item]/16384 # get the probability and add
 to total number of |1>'s

    total_0 = np.sqrt(total_0) # turn into probability amplitude
    total_1 = np.sqrt(total_1) # turn into probability amplitude

    print('The state we are trying to send is: %s|0>' %str(initial_state
[0]), '+ %s|1>' % str(initial_state[1]))
    print('The state we resulted with is: %s|0>' %str(total_0), '+ %s|1
>' % str(total_1))
    print()

    # calculate the % error of our teleportation
    if initial_state[0] == 0: # making sure we don't divide by zero if o
ur initial state was just |1>
        print('The error in |0> is:', 0.0, '%') # in this case there is
 zero error no matter what
    else:
        print('The error in |0> is:', abs(1 - (total_0/initial_state[0
]))*100, '%')

    if initial_state[1] == 0: # making sure we don't divide by zero if o
ur initial state was just |0>
        print('The error in |1> is:', 0.0, '%') # in this case there is
 zero error no matter what
    else:
        print('The error in |1> is:', abs(1 - (total_1/initial_state[1
]))*100, '%')

    return qc, counts
```

Pure State |1>:

In [ ]:
```
initial_state = [0,1] # teleport the state |1>
qc, counts = QuantumTeleport(initial_state) # collect data
plot_histogram(counts) # plot the results
```

```
In [ ]: # Test on a quantum computer

        qc = QuantumCircuit(3, 1)

        qc.barrier()

        # create a bell pair
        qc.x(q[0])
        qc.h(q[1]) # applies a hadamard to the second qubit
        qc.cx(q[1],q[2]) # applies a CNOT to the second(control) and third(targe
        t) qubits to entangle them together

        # prepare to teleport
        qc.cx(q[0],q[1]) # applies a CNOT to the first(control) and second(targe
        t) qubits
        qc.h(q[0]) # applies a hadamard to the first qubit

        # make necessary changes to the third qubit depending on the state of th
        e first two
        qc.cz(0,2) # Apply Z gate to qubit 3 if qubit 1 is in state |1>
        qc.cx(1,2) # Apply X gate to qubit 3 if qubit 2 is in state |1>

        qc.measure(2, 0) # measure qubit 3

        # The following was used from https://qiskit.org/textbook/ch-algorithms/
        teleportation.html
        backend = least_busy(provider.backends(filters=lambda b: b.configuration
        ().n_qubits >= 3 and
                                           not b.configuration().simulator and b
        .status().operational==True))
        job_exp = execute(qc, backend=backend, shots=8192)
        exp_result = job_exp.result()
        exp_measurement_result = exp_result.get_counts(qc)
        plot_histogram(exp_measurement_result)
```

Mixed State |+>:

```
In [ ]: initial_state = [1/np.sqrt(2),1/np.sqrt(2)] # prepare the |+> state
        qc, counts = QuantumTeleport(initial_state) # collect data
        plot_histogram(counts) # plot the results
```

```
In [ ]: # Test on a quantum computer

        qc = QuantumCircuit(3, 1)

        # create a bell pair
        qc.h(q[0])
        qc.h(q[1]) # applies a hadamard to the second qubit
        qc.cx(q[1],q[2]) # applies a CNOT to the second(control) and third(targe
        t) qubits to entangle them together

        # prepare to teleport
        qc.cx(q[0],q[1]) # applies a CNOT to the first(control) and second(targe
        t) qubits
        qc.h(q[0]) # applies a hadamard to the first qubit

        # make necessary changes to the third qubit depending on the state of th
        e first two
        qc.cz(0,2) # Apply Z gate to qubit 3 if qubit 1 is in state |1>
        qc.cx(1,2) # Apply X gate to qubit 3 if qubit 2 is in state |1>

        qc.measure(2, 0) # measure the 3rd qubit

        # The following was used from https://qiskit.org/textbook/ch-algorithms/
        teleportation.html
        backend = least_busy(provider.backends(filters=lambda b: b.configuration
        ().n_qubits >= 3 and
                                           not b.configuration().simulator and b
        .status().operational==True))
        job_exp = execute(qc, backend=backend, shots=8192)
        exp_result = job_exp.result()
        exp_measurement_result = exp_result.get_counts(qc)
        plot_histogram(exp_measurement_result)
```

Teleporting Entangled Qubits

```
In [ ]: def QuantumTeleportEntangled():
            # declare the 4 qubits
            q = QuantumRegister(4, name = 'q')

            # classical registers to check if the teleportation was successful
            c = ClassicalRegister(2, name = 'c')

            # crz checks whether to operate Z on the 3rd qubit after the gates
            crz = ClassicalRegister(1, name = 'crz')
            # crx check whether to operate X on the 3rd qubit after the gates
            crx = ClassicalRegister(1, name = 'crx')

            # initialize our quantum circuit
            qc = QuantumCircuit(q,crz,crx,c,name ='qc')

            # create our first bell pair |Ψ−⟩ab to be teleported
            qc.h(q[0]) # applies a hadamard to the first qubit
            qc.cx(q[0],q[1]) # applies a CNOT to the first(control) and second(t
        arget) qubits to entangle them together
            qc.z(q[0]) # applies a Z gate to the first qubit
            qc.x(q[1]) # applies an X gate to the second qubit

            # create our second bell pair |Φ+⟩cd
            qc.i(q[2]) # makes the visuals line up
            qc.i(q[3]) # makes the visuals line up
            qc.h(q[2]) # applies a hadamard to the third qubit
            qc.cx(q[2],q[3]) # applies a CNOT to the third(control) and fourth(t
        arget) qubits to entangle them together
            qc.barrier() # separates the stages of the circuit

            #prepare to teleport
            qc.cx(q[1], q[2]) # applies a CNOT to the second(control) and third
        (target) qubits
            qc.h(q[1]) # applies a hadamard to the second qubit
            qc.i(q[0]) # makes the visuals line up
            qc.i(q[0]) # makes the visuals line up
            qc.i(q[2]) # makes the visuals line up
            qc.i(q[3]) # makes the visuals line up
            qc.i(q[3]) # makes the visuals line up
            qc.barrier() # separates the stages of the circuit

            # perform measurement on the second and third qubits
            qc.measure(q[1],crz) # measure qubit 2
            qc.measure(q[2],crx) # measure qubit 3

            qc.x(q[3]).c_if(crx, 1) # Apply X gate to qubit 4 if qubit 3 is in s
        tate |1>
            qc.z(q[3]).c_if(crz, 1) # Apply Z gate to qubit 4 if qubit 2 is in s
        tate |1>

            # perform measurement on the third qubit to see if we teleported cor
        rectly
            qc.i(q[0]) # makes the visuals line up
            qc.i(q[0]) # makes the visuals line up
            qc.measure(q[0],c[0]) # measure qubit 1
            qc.measure(q[3],c[1]) # measure qubit 4
```

```
    # use the classical simulator
    backend = Aer.get_backend('qasm_simulator') # classical simulator
    counts = execute(qc, backend, shots=16384).result().get_counts() # s
imulate a large set of data

    # draw the circuit
    print('Quantum Teleportation Circuit Diagram:')
    print(circuit_drawer(qc)) # ascii write out

    # calculate the probability amplitudes we result with on average
    total_01 = 0 # stores the total prob we find |01>
    total_10 = 0 # stores the total prob we find |10>
    for item in counts:
        if item [0:2] == '01': # check if we measured a |01> for the new
entangled pair
            total_01 += counts[item]/16384 # get the probability and add
to total number of |01>'s
        if item[0:2] == '10': # check if we measured a |10> for the new
 entangled pair
            total_10 += counts[item]/16384 # get the probability and add
to total number of |10>'s

    total_01 = np.sqrt(total_01) # turn into probability amplitude
    total_10 = np.sqrt(total_10) # turn into probability amplitude

    print('The state we are trying to send is: %s|01>' %str(1/np.sqrt
(2)), '- %s|10>' % str(1/np.sqrt(2)))
    print('The state we resulted with is: %s|01>' %str(total_01), '- %s|
10>' % str(total_10))
    print()

    # calculate the % error of our teleportation
    print('The error in |01> is:', abs(1 - (total_01*np.sqrt(2)))*100,
'%')
    print('The error in |10> is:', abs(1 - (total_10*np.sqrt(2)))*100,
'%')

    return counts
```

Example with |Ψ−⟩ab ⊗ |Φ+⟩cd:

In [ ]:
```
counts = QuantumTeleportEntangled()
plot_histogram(counts) # plot the results
```

In [ ]:
```
# Test on a quantum computer

qc = QuantumCircuit(4, 2)

# create a bell pair
qc.h(0) # applies a hadamard to the first qubit
qc.cx(0,1) # applies a CNOT to the first(control) and second(target) qub
its to entangle them together
qc.z(0) # apply a Z gate to qubit 1
qc.x(1) # apply an X gate to qubit 1

# create a bell pair
qc.h(2) # applies a hadamard to the third qubit
qc.cx(2,3) # applies a CNOT to the third(control) and fourth(target) qub
its to entangle them together

# prepare to teleport
qc.cx(1,2) # applies a CNOT to the second(control) and third(target) qub
its
qc.h(1) # applies a hadamard to the first qubit

# make necessary changes to the fourth qubit depending on the state of t
he middle two
qc.cz(1,3) # Apply Z gate to qubit 4 if qubit 2 is in state |1>
qc.cx(2,3) # Apply X gate to qubit 4 if qubit 3 is in state |1>

qc.measure(0, 0)
qc.measure(3, 1)

# The following was used from https://qiskit.org/textbook/ch-algorithms/
teleportation.html
backend = least_busy(provider.backends(filters=lambda b: b.configuration
().n_qubits >= 3 and
                                   not b.configuration().simulator and b
.status().operational==True))
job_exp = execute(qc, backend=backend, shots=8192)
exp_result = job_exp.result()
exp_measurement_result = exp_result.get_counts(qc)
plot_histogram(exp_measurement_result)
```

Teleporting Twice

```
In [ ]: def TeleportTwice(initial_state):
            # declare the 3 qubits; one to teleport and the other two to entangl
        e
            q = QuantumRegister(5, name = 'q')

            # initialize the classical registers

            # crz checks whether to operate Z on the 3rd qubit after the gates
            crz1 = ClassicalRegister(1, name = 'crz1')
            # crx check whether to operate X on the 3rd qubit after the gates
            crx1 = ClassicalRegister(1, name = 'crx1')
            # crz checks whether to operate Z on the 5th qubit after the gates
            crz2 = ClassicalRegister(1, name = 'crz2')
            # crx check whether to operate X on the 5th qubit after the gates
            crx2 = ClassicalRegister(1, name = 'crx2')
            # c is the final measurement of the 5th quibit
            c = ClassicalRegister(1, name = 'c')

            # initialize our quantum circuit
            qc = QuantumCircuit(q,crz1,crx1,crz2,crx2,c, name ='qc')

            # store the initial state into qubit 1, which will be teleported to
         qubit 3
            qc.initialize(initial_state, 0)
            qc.i(q[1]) # makes the visuals line up
            qc.i(q[2]) # makes the visuals line up

            # create a bell pair
            qc.i(q[0]) # makes the visuals line up
            qc.h(q[1]) # applies a hadamard to the second qubit
            qc.cx(q[1],q[2]) # applies a CNOT to the second(control) and third(t
        arget) qubits to entangle them together
            qc.barrier() # separates the stages of the circuit

            # prepare to teleport the first time
            qc.cx(q[0],q[1]) # applies a CNOT to the first(control) and second(t
        arget) qubits
            qc.h(q[0]) # applies a hadamard to the first qubit
            qc.i(q[1]) # makes the visuals line up
            qc.i(q[2]) # makes the visuals line up
            qc.i(q[2]) # makes the visuals line up
            qc.barrier() # separates the stages of the circuit

            # perform measurement on the first and second qubits
            qc.measure(q[0],crz1) # measure qubit 1
            qc.measure(q[1],crx1) # measure qubit 2

            # make necessary changes to the third qubit depending on the state o
        f the first two
            qc.x(q[2]).c_if(crx1, 1) # Apply X gate to qubit 3 if qubit 2 is in
         state |1>
            qc.z(q[2]).c_if(crz1, 1) # Apply Z gate to qubit 3 if qubit 1 is in
         state |1>
            qc.barrier()

            # prepare to teleport the second time
```

```
    qc.cx(q[2],q[3]) # applies a CNOT to the third(control) and fourth(t
arget) qubits
    qc.h(q[2]) # applies a hadamard to the third qubit
    qc.barrier() # separates the stages of the circuit

    # perform measurement on the third and fourth qubits
    qc.measure(q[2],crz2) # measure qubit 3
    qc.measure(q[3],crx2) # measure qubit 4

    # make necessary changes to the third qubit depending on the state o
f the previous two
    qc.x(q[4]).c_if(crx2, 1) # Apply X gate to qubit 5 if qubit 3 is in
 state |1>
    qc.z(q[4]).c_if(crz2, 1) # Apply Z gate to qubit 5 if qubit 2 is in
 state |1>

    # perform measurement on the fifth qubit to see if we teleported cor
rectly
    qc.measure(q[4],c) # measure qubit 5

    # use the classical simulator
    backend = Aer.get_backend('qasm_simulator') # classical simulator
    counts = execute(qc, backend, shots=16384).result().get_counts() # s
imulate a large set of data

    # draw the circuit
    print('Quantum Teleportation Circuit Diagram:')
    print(circuit_drawer(qc)) # ascii write out

    # calculate the probability amplitudes we result with on average
    total_0 = 0 # stores the total prob we find |0>
    total_1 = 0 # stores the total prob we find |1>
    for item in counts:
        if item[0] == '0': # check if we measured a |0> for the teleport
ed qubit
            total_0 += counts[item]/16384 # get the probability and add
 to total number of |0>'s
        else: # check if we measured a |1> for the teleported qubit
            total_1 += counts[item]/16384 # get the probability and add
 to total number of |1>'s

    total_0 = np.sqrt(total_0) # turn into probability amplitude
    total_1 = np.sqrt(total_1) # turn into probability amplitude

    print('The state we are trying to send is: %s|0>' %str(initial_state
[0]), '+ %s|1>' % str(initial_state[1]))
    print('The state we resulted with is: %s|0>' %str(total_0), '+ %s|1
>' % str(total_1))
    print()

    # calculate the % error of our teleportation
    if initial_state[0] == 0: # making sure we don't divide by zero if o
ur initial state was just |1>
        print('The error in |0> is:', 0.0, '%') # in this case there is
 zero error no matter what
    else:
        print('The error in |0> is:', abs(1 - (total_0/initial_state[0
]))*100, '%')

    if initial_state[1] == 0: # making sure we don't divide by zero if o
ur initial state was just |0>
        print('The error in |1> is:', 0.0, '%') # in this case there is
 zero error no matter what
    else:
        print('The error in |1> is:', abs(1 - (total_1/initial_state[1
]))*100, '%')

    return qc, counts
```

Example with |+>:

In [ ]:
```
initial_state = [1/np.sqrt(2),1/np.sqrt(2)] # prepare the |+> state
qc, counts = TeleportTwice(initial_state) # collect data
plot_histogram(counts) # plot the results
```

In [ ]:
```
# Test on a quantum computer

qc = QuantumCircuit(5, 1)

qc.barrier()

# create a bell pair
qc.h(0) # make the |+> state
qc.h(1) # applies a hadamard to the second qubit
qc.cx(1,2) # applies a CNOT to the second(control) and third(target) qub
its to entangle them together

# prepare to teleport the first time
qc.cx(0,1) # applies a CNOT to the first(control) and second(target) qub
its
qc.h(0) # applies a hadamard to the first qubit

# make necessary changes to the third qubit depending on the state of th
e first two
qc.cz(0,2) # Apply Z gate to qubit 3 if qubit 2 is in state |1>
qc.cx(1,2) # Apply X gate to qubit 3 if qubit 1 is in state |1>

# prepare to teleport the second time
qc.cx(2,3) # applies a CNOT to the third(control) and fourth(target) qub
its
qc.h(2) # applies a hadamard to the third qubit

# make necessary changes to the fifth qubit depending on the state of th
e previous two
qc.cz(2,4) # Apply Z gate to qubit 5 if qubit 3 is in state |1>
qc.cx(3,4) # Apply X gate to qubit 5 if qubit 4 is in state |1>

qc.measure(4, 0)

# The following was used from https://qiskit.org/textbook/ch-algorithms/
teleportation.html
backend = least_busy(provider.backends(filters=lambda b: b.configuration
().n_qubits >= 3 and
                                   not b.configuration().simulator and b
.status().operational==True))
job_exp = execute(qc, backend=backend, shots=8192)
exp_result = job_exp.result()
exp_measurement_result = exp_result.get_counts(qc)
plot_histogram(exp_measurement_result)
```