4 Testing

*Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, power system, or software.*

The testing plan should connect the requirements and the design to the adopting test strategy and instruments. In this overarching introduction, given an overview of the testing strategy. Emphasize any unique challenges to testing for your system/design.

4.1 Unit Testing

What units are being tested? How? Tools?

For our project, we have two big parts that need to be tested. The two parts are 1) classical networks which will be used for communication between router and nodes, and 2) quantum communication lines which will be essential to make our network maintain integrity of quantum information as its being transported. For each part, there are small units to be worked on to make our network work in an appropriate and reliable manner.

In this part, we're talking about small units for network functions. Units for interface tracking, and reliability tracking, such as logger will be discussed in later chapters.

Classical networks

Our classical networks will be coded in python, and it will be needed to make nodes communicate with each other inside of our simulated network. To make this possible, our network will consist of below units and will be tested same as below description.

1) Router

Router is the main component of our classical networks and the "big brain" of our classical network. Likewise with the analogy of "big brain", it handles important jobs concerning our network. It will identify jobs and distribute jobs to entire nodes, and communicate with each other to determine which nodes finish their job, and summarize outputs to create one final answer. It is more like a container which has components below and works as a single entity inside of our network. This component will be tested consistently while we build each small subunit same as below description a)~c) and it will be integrated when we can ensure each subunit works perfectly.

a) Job identification unit

This unit is taking a job from the user e.g.) calculating energy bands of Nitrogen atoms. And identify the user's job and ready before this job will be distributed to each node. In the ideal case, it also can interpret users' jobs to quantum computers since that's what the final version of quantum networks will do 20 years later, but for this project and based on our advisors' needs, it will be simply taking simple jobs from users and preparing to send them to each node. It will be constructed first and will be tested first. It will be a module in python, so it will be tested whether it takes jobs from users and recognize our test job which is very simple, such as adding up from 1 to 1000. We can test it in a python interpreter.

b) Communication unit with each classical nodes

This unit is communicating with each classical node in our quantum network. Again, in the future each node also will be quantum, but now, we're using classical nodes. This unit will be used to communicate with router and classical nodes to make sure the router knows which nodes finish their job or not. It will be a module in python, so it will be tested whether it sends a flag signal well or not. We can test it in a python interpreter.

c) Output unit

This unit summarizes answers from each classical node and makes one final answer to users who input corresponding works. It might not be required for our current project since at this stage we're using classical nodes, but we're planning to make this module since it will be essential for quantum nodes. Likewise other router components, It will be a module in python and we can test it in a python interpreter.

Quantum communication part

Our quantum communication line will be also made by python, especially using Qiskit. This communication line will consist of below units and will be tested same as below description.

1) Bell State Measurement

Bell State measurement is required to make two different q-bits entangled. This function will be simulated in Qiskit and tested in Qiskit which is a python extension for quantum computer simulation. Tests about our BSM will be done entirely in the Qiskit environment and will be done by giving the function which performs this measurements some values and looking to see if it matches the expected outcomes within a given percentage.

2) EPR pair

An EPR pair is needed to connect Router's Qbit and node's Qbit and communicate close to speed of light. EPR pair function will be simulated in Qiskit and tested in Qiskit. Tests about our EPR pair in our quantum communication will be done entirely in the Qiskit environment using a similar approach as the BSM.

3) Cluster states for each data

This is simply a bunch of BSM and EPR pairs for implementing error correction features in our quantum network. Like what we will do in part 1) and 2), We're using Qiskit. Tests about our cluster states in our quantum communication will be done entirely in the Qiskit environment.

4.2 Interface Testing

What are the interfaces in your design? Discuss how the composition of two or more units (interfaces) are being tested. Tools?

Main interfaces of our simulated network will be the interface between the classical networks part and quantum communication part. We're planning to use python for our classical networks part and also use python, especially Qiskit. Since we're using the same platformI(python interpreter) for each different two parts, our interface testing will be easily done by making a logger for each part. By using loggers, we will be able to track the interface between classical networks and quantum communication parts.

4.3 Integration Testing

What are the critical integration paths in your design? Justification for criticality may come from your requirements. How will they be tested? Tools?

The critical integration paths for our project would be anywhere that the classical and quantum parts of our network intertwine and must communicate and or interact with each other. Although we're making two parts independently, we're discussing communication rules between each other parts to make sure that everything works well at the final integration section. This will be moderated by Derrick. We think this workflow is best for our team since we have two team members who specialize in classical computer communication and network, one member who specializes in quantum circuit, and one member who specializes in system engineering. We will coordinate with each other to establish a baseline of how the two should work and communicate together and develop tests around this. Every time changes are made, they will be run against these tests.

4.4 System Testing

Describe system level testing strategy. What set of unit tests, interface tests, and integration tests suffice for system level testing? This should be closely tied to the requirements. Tools?

We're planning to use a dummy job (e.g. add numbers from 1 to 10000, calculate light path in uniaxial dielectric material etc.) for our network. We can test our system to input our dummy job to our network, and prove the functionality and performance of our system. For each classical node, likewise a dummy job, we're planning to simulate dummy nodes which will be recognized as separate computers(nodes) in our simulation, but actually just imaginary nodes.

4.5 Regression Testing

How are you ensuring that any new additions do not break the old functionality? What implemented critical features do you need to ensure do not break? Is it driven by requirements? Tools?

We're planning to use github for our project for our work history tracking. Unexpected errors could be reverted using this work history tracking software. Also, for advanced functions such as error corrections and QKD functions regarding quantum communication parts, we will build a very rudimentary quantum communication structure at first, and based on that, try to add functions on top of it. If some functionality weren't working appropriately, we can take it apart from our network and fix it partially.

4.6 Acceptance Testing

How will you demonstrate that the design requirements, both functional and non-functional are being met? How would you involve your client in the acceptance testing?

Our client's ultimate goal is to use our network for their research and development of an actual quantum cluster computing network. For initiation of their research, making a simulated quantum cluster network and running a dummy job will be enough. Even Though dummy jobs will be done in classical computers, but from the fact that we can't have quantum nodes at this stage and this is their initiating procedure, running a dummy job and presenting results using a classical computer will be sufficient to present to our client since they will get a conceptual network for quantum cluster computing and they will initiate their study based on our work.
However, since our project is research intensive and our project is using agile and waterfall methodologies, our team and clients have agreed that these requirements could be modulated by time.

It is further important to note that we do have access to some limited quantum computing resources online via IBM. This means that we will have some capability to see if this network simulation runs appropriately on / with the hardware it was meant to.

4.7 Security Testing (if applicable)

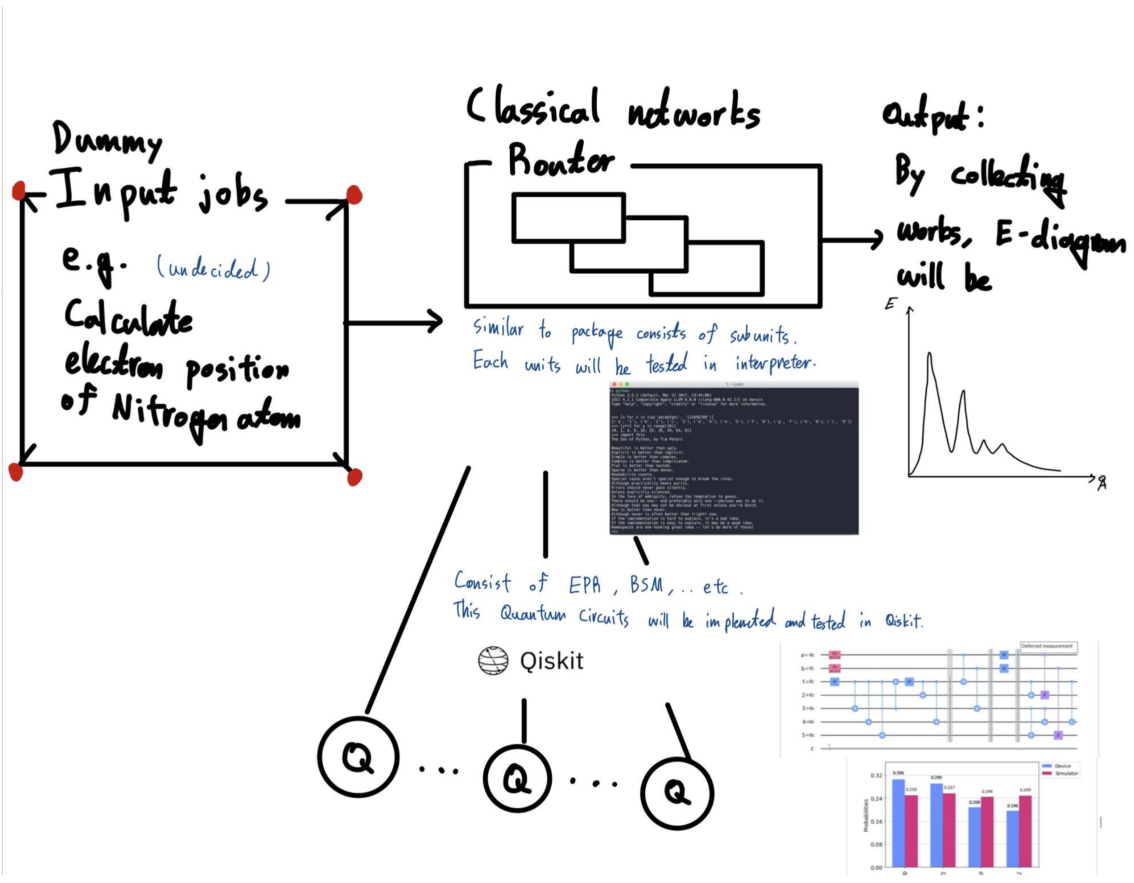
This will be directly connected to the Testing of Cluster States what we already described in section 4-1. That's because enabling Cluster States in our network says that we can implement advanced error-correction functions, but it also means that we can implement Quantum Key distribution functions in our network which ensure security in our network.

4.8 Results

What are the results of your testing? How do they ensure compliance with the requirements? Include figures and tables to explain your testing process better. A summary narrative concluding that your design is as intended is useful.

Our results from testing will be the correct output from a list of dummy jobs we gave it from the beginning. Knowing these, we will be looking for the correct response from the output which can be in a few different formats depending on the input data. In the graphic below, you can see we are inputting a list of jobs into our classical network router which is then splitting them up via the quantum channel to the nodes. Then, our router will need to collect the information from those nodes and output the answer. If it does all of these things then it will result in a working router.

Our testing results will be easily explained from the below graphic.


Dummy
Input jobs
e.g. (undecided)
Calculate electron position of Nitrogen atom
Classical networks
Router
Output:
By collecting works, E-diagram will be
E
Å
Similar to package consists of subunits.
Each units will be tested in interpreter.
Consist of EPA, BSM, .. etc.
This Quantum Circuits will be implemented and tested in Qiskit.
Qiskit
Q
Q
Q
Deferred measurement
Probabilities
Device
Simulator
0.306
0.250
0.290
0.257
0.208
0.244
0.196
0.249

```
> python
Python 3.5.2 (default, Mar 21 2017, 23:44:08)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.


>>> [x for x in zip('abcdefghi', '123456789')]
[('a', '1'), ('b', '2'), ('c', '3'), ('d', '4'), ('e', '5'), ('f', '6'), ('g', '7'), ('h', '8'), ('i', '9')]
>>> [y**2 for y in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

figure : description of our testing graphic. expected results are described graphically (classical network: logger in python interpreter. quantum communication : quantum gate and probability graph as an output.)