EE/ CprE/ SE 492 - sddec23-17

Simulated Design of Quantum Networks

Biweekly Status Report

September 28 - October 13

Client: Dr. Durga Paudyal

Faculty Advisor: Dr. Durga Paudyal

Team Members:

Benjamin Amick - Network security engineer

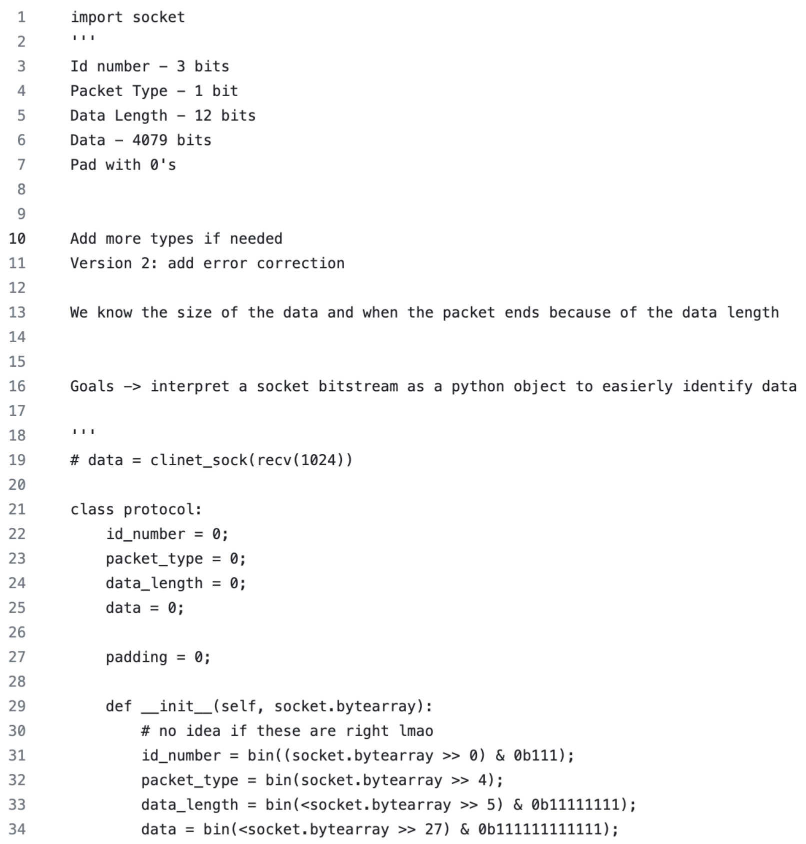
Derrick Wright - System integration engineer

Ohik Kwon- System component designer

Steven Tompary- Network engineer

Past Week Accomplishments

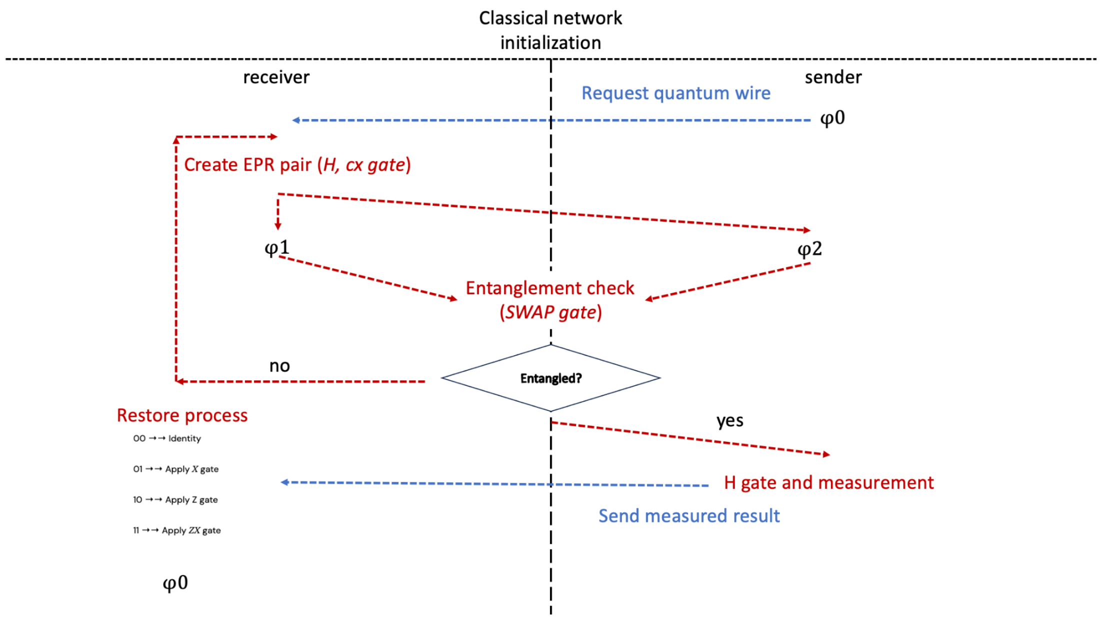
We have begun to build our first iteration. This version includes the creation of both quantum connections and classical connections. We have spent this week beginning to create skeleton code for our routers and our nodes. We have also implemented a basic classical protocol to be implemented by these nodes. As we are currently troubleshooting this, we are hoping to have working communication between them where they can communicate with each other in both quantum and classical channels. Below images are screenshots of our classical part. We can also share to the public on our github repository.



```
import socket
'''
Id number - 3 bits
Packet Type - 1 bit
Data Length - 12 bits
Data - 4079 bits
Pad with 0's


Add more types if needed
Version 2: add error correction

We know the size of the data and when the packet ends because of the data length


Goals -> interpret a socket bitstream as a python object to easierly identify data

'''
# data = clinet_sock(recv(1024))

class protocol:
    id_number = 0;
    packet_type = 0;
    data_length = 0;
    data = 0;

    padding = 0;

    def __init__(self, socket.bytearray):
        # no idea if these are right lmao
        id_number = bin((socket.bytearray >> 0) & 0b111);
        packet_type = bin(socket.bytearray >> 4);
        data_length = bin(<socket.bytearray >> 5) & 0b11111111);
        data = bin(<socket.bytearray >> 27) & 0b111111111111);
```

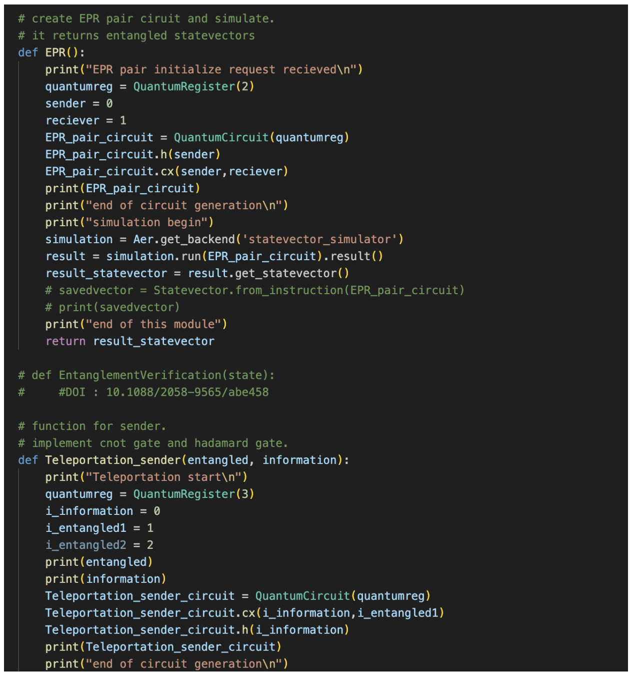
[figure: Part of classical network router code]

For a quantum part, we re-establish the protocol with more detailed description to make it easier to communicate with classical parts.

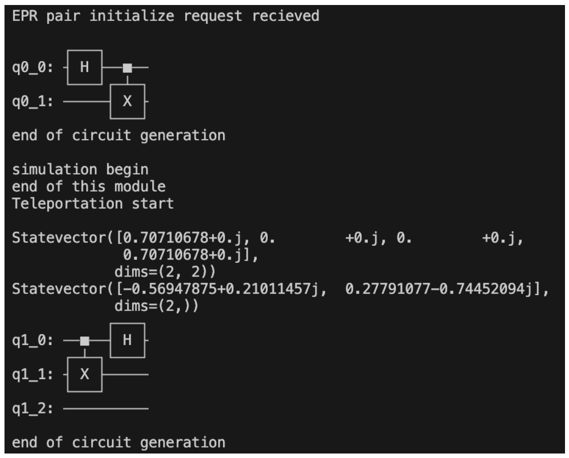


[figure: Quantum network protocol]

Then, based on our basic quantum teleportation code, we have started to modularize for calling inside of our network. This represents each different quantum device in the real world.



```
# create EPR pair ciruit and simulate.
# it returns entangled statevectors
def EPR():
    print("EPR pair initialize request recieved\n")
    quantumreg = QuantumRegister(2)
    sender = 0
    reciever = 1
    EPR_pair_circuit = QuantumCircuit(quantumreg)
    EPR_pair_circuit.h(sender)
    EPR_pair_circuit.cx(sender,reciever)
    print(EPR_pair_circuit)
    print("end of circuit generation\n")
    print("simulation begin")
    simulation = Aer.get_backend('statevector_simulator')
    result = simulation.run(EPR_pair_circuit).result()
    result_statevector = result.get_statevector()
    # savedvector = Statevector.from_instruction(EPR_pair_circuit)
    # print(savedvector)
    print("end of this module")
    return result_statevector

# def EntanglementVerification(state):
#     #DOI : 10.1088/2058-9565/abe458

# function for sender.
# implement cnot gate and hadamard gate.
def Teleportation_sender(entangled, information):
    print("Teleportation start\n")
    quantumreg = QuantumRegister(3)
    i_information = 0
    i_entangled1 = 1
    i_entangled2 = 2
    print(entangled)
    print(information)
    Teleportation_sender_circuit = QuantumCircuit(quantumreg)
    Teleportation_sender_circuit.cx(i_information,i_entangled1)
    Teleportation_sender_circuit.h(i_information)
    print(Teleportation_sender_circuit)
    print("end of circuit generation\n")
```



```
EPR pair initialize request recieved

        ┌───┐     
q0_0: ┤ H ├──■──
        └───┘┌─┴─┐
q0_1: ─────┤ X ├
             └───┘
end of circuit generation

simulation begin
end of this module
Teleportation start

Statevector([0.70710678+0.j, 0.        +0.j, 0.        +0.j,
             0.70710678+0.j],
            dims=(2, 2))
Statevector([-0.56947875+0.21011457j,  0.27791077-0.74452094j],
            dims=(2,))
                   ┌───┐
q1_0: ──■──┤ H ├
        ┌─┴─┐└───┘
q1_1: ┤ X ├─────
        └───┘
q1_2: ──────────

end of circuit generation
```

[figure: Quantum part modularization]

- **Ben** - Worked with Steven to create outline of classical network protocol and began implementing them with the router and nodes

- **Ohik** - Worked with Derrick to ensure that the router and nodes will be able to communicate quantum instructions.

Modularize quantum circuits by parts so we can call up corresponding functions when needed.

- **Steven** - Worked with Ben to create outline of classical network protocol and began implementing them with the router and nodes

- **Derrick** - Developed base code for the router and nodes so that they will have basic communication

Resources

Our git repository
https://github.com/Kcops11/SeniorDesignQuantum17

Books we are reading

- Quantum Computation and Quantum Information, Michael A. Nielson

Articles we found this week and reading

- Github Qiskit Community Tutorials
- When Entanglement meets Classical Communications: Quantum Teleportation for the Quantum Internet, IEEE Transactions on Communication, 2020, 10.1109/TCOMM.2020.2978071
- The controlled SWAP test for determining quantum entanglement, Quantum Science and Technology, 2021, *https://doi.org/10.1088/2058-9565/abe458*

Pending Issues

- There are no pending issues for this week since we all agreed on detailed functionalities of our first iteration network.

Individual Contributions

Team Member	Contribution	Weekly Hours	Total Hours
Benjamin Amick	Work on cluster computing architecture	9	60
Derrick Wright	Made module for quantum teleportation	9	60
Ohik Kwon	Found systematic way to verify entanglement state	9	60
Steven Tompary	Work on cluster computing architecture	9	60

Plans for Coming Week

- Continue to work on the communication between router and nodes then ensure that they can communicate quantum instructions