Extended Quantum Network Project

Quantum Computers Systems Simulation UROP Research Intern
for Computer Systems Institute (SYS), USI - Università della Svizzera italiana

Lorenzo Zaniol (lorenzo.zaniol@usi.ch, lorenzo.zaniol@alumni.usi.ch)

Project Repository August 29, 2024

1. Project Description

We implemented a *Quantum Network* (QN) using the simulator Netsquid that follows the star topology, one of the most common network topologies, shown in Figure 1. The center of the network is a node with only a Quantum Source as component. All the other nodes are composed of quantum processors, and are connected to the source node by means of quantum connections. One of the points of the network is a repeater connected to another node. This work is based on the previous project, done by Edoardo Riggio (edoardo.riggio@usi.ch), Işın Su Ecevit (isin.su.ecevit@usi.ch) and myself, available at this GitHub repository. The method *entangle_nodes* connects two nodes (porting of the old project) and method *protocol_a* connects three nodes following the example with the same name at page 4 of the scientific paper called "An Algebraic Language for Specifying Quantum Networks" [1] (done by my two supervisors, Anita Buckley and Pavel Chuprikov), published by the prestigious ACM (Association for Computing Machinery) available here.

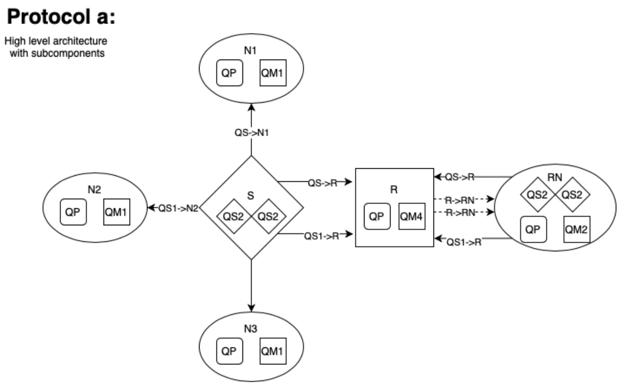


Figure 1: Diagram of the generated QN

1.1. Key features

The key features of the structure we created are:

- *Quantum Networks* (QNs) composed of directly connected nodes (called *Quantum Node* (N_i), where i is the respective index) as well as a indirectly connected node (called *Remote Node i* (RN_i)) connected using a *Quantum Repeater* (R).
- Generating QNs dynamically with n nodes, from which the following are created: [(n-2) $\times$ N_i], [1 $\times$ R], [1 $\times$ RN_i] and [1 $\times$ *Quantum Source* (S)].
- The ability to have one entangled Qbit each, between any two (with method *entangle_nodes*) or three nodes (with method *protocol_a*).
- Automatic statistics data export and summary plot of multiple experiments, over different channel length.
- The option to enable/disable the Quantum Channel (QC) models applied everywhere.

2. Implementation

2.1. Network Components

The nodes in the network contain the following components: Quantum Source (Q. Source), which generates a Bell pairs `|00> + |11>` each, since there are 2 in the Source node (S) and Remote Node (RN_i); Quantum Processor (QP), present in all the nodes apart from the Source node (S); Quantum Memory with 1 memory position (QM1), in all normal nodes (N_i), 2 memory positions (QM2), in the Remote Node (RN_i), and 4 memory positions (QM4), in the repeater node (R). The distribution of the components can be found in Table 1. Both types of channels (classical and quantum) used in the network are unidirectional.

	Q. Source	Q. Processor	Q. Memory	Q. Memory	Q. Memory
	2	1	1	2	4
S	✓				
R		✓			✓
N_i		✓	✓		
RN_i	✓	✓		✓	

Table 1: Distribution of the Quantum components to the nodes.

2.2. *Quantum Channel* (QC) models

At runtime, by passing the first argument, called *models_name*, by using *empty* all loss, noise and delay are disabled in all Quantum Channels (QC) in the Quantum Network (QN). If instead the keyword *combined* is passed, the following well established techniques with (mean) parameters observed in real QNs are added as models to make the simulations on the QN realistic:

Fibre loss: Model for exponential photon loss on fibre optic channels. Uses length of transmitting channel to sample an exponential loss probability.

T1T2 noise: Commonly used phenomenological noise model based on T1 and T2, where T1 time dictates amplitude damping component and T2 time (called T2 Hahn) dictates the dephasing component.

Fibre delay: Transmission delay model based on constant speed of photons through fibre. The travel distance is given by the length of channel. The default parameter, c refers to the speed of light in a F1 fibre cable (the best available), which is 200,000 km/s, 30% slower than the speed of light in a vacuum.

2.3. Challenges

Most of the challenges we faced during the implementation were mainly caused by Netsquid not being widely used, and support for the tool not being enough, which affected the installation and implementation processes greatly. The lack of documentation as well as false or incomplete information on the tool resulted in occasional misunderstandings of the components we were able/allowed to utilize. We had problem with the Quantum Source component that supports the generation of 4 Qbits simultaneously, however not 2 Bell pairs `|00> + |11>`, that based on the documentation should be possible, but on changing the format of them, we encounter an undocumented error `NotImplemented`. In order to get around this problem, I had to double all the Quantum Source component.

Another struggle was adapting the existing automating routing with multiple channels (connected to the repeater) and especially do the correct port mapping between the starting port, ending port and storing the Qbit (if received in the correct memory slot, otherwise it will destroy the currently stored Qbit as well as loosing track of where it came from/entangled with) in a Quantum Memory (QM). To ease the task, we created a table that maps everything as well as a architecture diagram with all sub-components with all port mapping using the default arguments of the program; resulting in the very detailed Figure 2.

3. Program Usage

The usage is available in the readme file in the GitHub repository of the project as well as running the following command in the terminal `python3 main.py help`. The program uses the following command structure:

```
python3 main.py <models_name> <method_name>
<nodes> <debug> <experiment_num>
```

where:

- *models_name* can be either: *combined* or *empty*, with the default value set to *empty*, that enables/disables the quantum: loss, noise and delay models.
- *method_name* can be either: *protocol_a* or *entangle_nodes*, with the default value set to *protocol_a*, that selects the method to use to send the qubits.
- *nodes* is a comma separated list of numbers between 1 and 4 (both included), with 2 or 3 numbers (2 for using *entangle_nodes* and 3 for using *protocol_a*), with the default value set to *1,2,4*, that represents the nodes to connect with entangled qubits.
- *debug* can be either: *False* or *True*, with the default value set to *True*, that enables/disables the print of additional debug information, in the standard output (in the terminal).
- *experiment_num* is a positive number (0 included), with the default value set to 0, that enables the execution of the program via the experiment wrapper (when not 0) that executes the program that amounts of times for every different channel length (to get an approximated statistical probability of the execution/operations) as well as collecting data (as *csv* file format) and generating the corresponding plots about the fidelity of the qubits over different channel length (as *png* file format) automatically (in the folder called *out*) with name of the inputted values; otherwise (with 0) it does not collect and generates anything (by not using the experiment wrapper) and simply displays the results in the standard output (in the terminal).

4. Examples

Default, an unrealistic (almost perfect system) simulation for *protocol_a*:

```
python3 main.py
```

is equivalent to the following:

```
python3 main.py empty protocol_a 1,2,4 True
0
```

Realistic simulation for *protocol_a* with 1 execution and no data collection and plot generation:

```
python3 main.py combined protocol_a 1,2,4
True 0
```

Realistic simulation for *protocol_a* with 100 execution and data collection and plot generation:

```
python3 main.py combined protocol_a 1,2,4
True 100
```

Unrealistic (almost perfect system) simulation for *entangle_nodes*:

```
python3 main.py empty entangle_nodes 1,4
True 0
```

Realistic simulation for *entangle_nodes* with 1 execution and no data collection and plot generation:

```
python3 main.py combined entangle_nodes 1,4
True 0
```

Realistic simulation for *entangle_nodes* with 100 execution and data collection and plot generation:

```
python3 main.py combined entangle_nodes 1,4
True 100
```

5. Conclusion

This project accomplished its aim of develop a working example of *protocol_a*, that it is a possible way to fully-functional quantum star network capable of establishing concurrent connections, between nodes. We integrated/ported/built on top of the previous project (done 2 years prior), to reuse the utilities/helper method, in particular the generate entanglement as well as the automatic extraction and collection of data (by re-running) and plot generation of different channel length over the fidelity, called experiment. The addition of the command line arguments, was done in order to modify the behaviour/-parameter without modifying the program (how the previous project required) and be easily extendable, in particular for new examples of Qbit connections, that don't require another Quantum Network (QN) architecture, and Quantum Channel (QC) models.

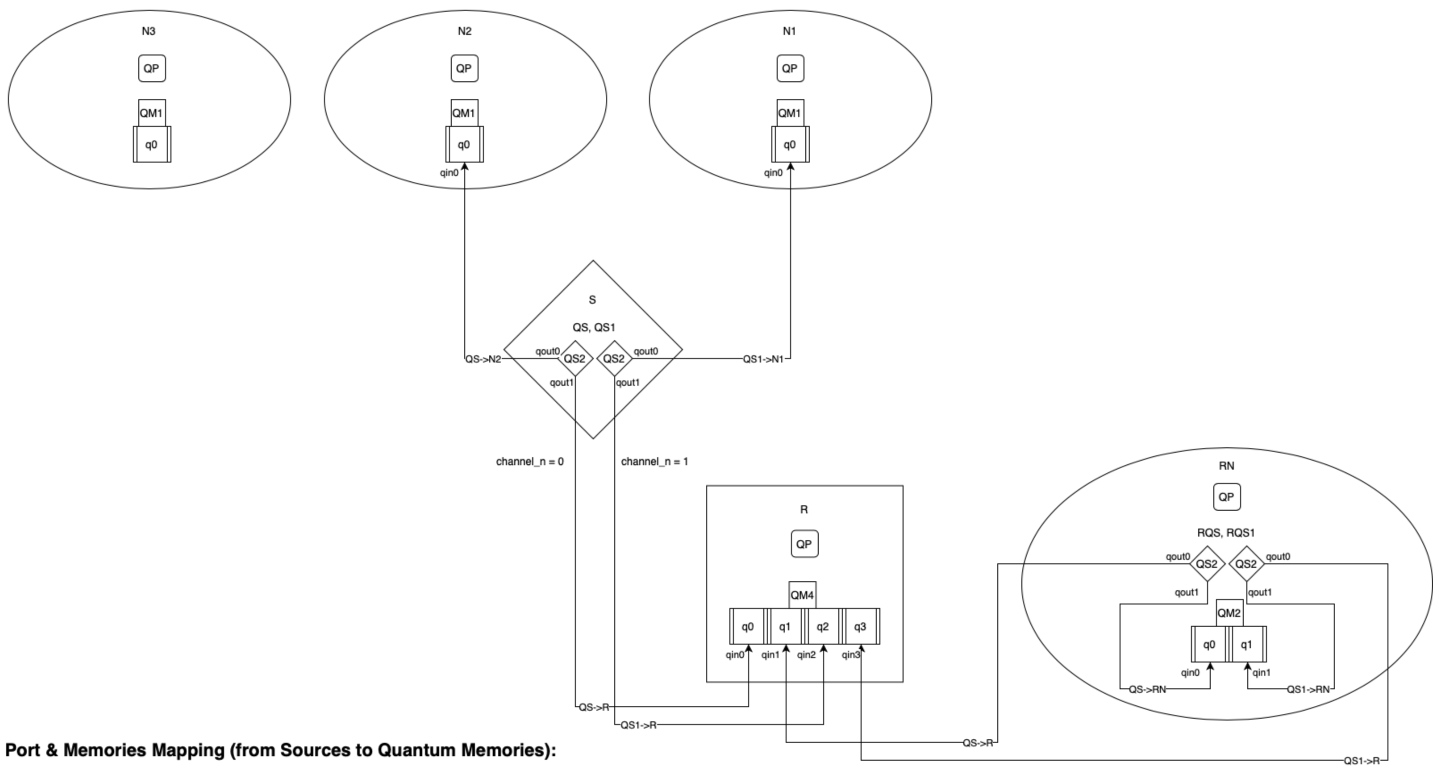
References

[1] Anita Buckley et al. "An Algebraic Language for Specifying Quantum Networks". In: *Proc. ACM Program. Lang.* 8.PLDI (June 2024). DOI: 10.1145/3656430. URL: https://doi.org/10.1145/3656430.

Protocol a:

High level architecture with:
subcomponents
sources names
ports mapping
memories

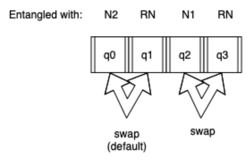
Protocol a with the following parameters/values:
nodes: 1,2,4
method_name protocol_a
and rest are not relevant (here)



Port & Memories Mapping (from Sources to Quantum Memories):

QS Node	QS Name/label	Channel_n	QS Port Name	Node Port Name	Node Name with memory	Memory	Memory Details
S	Quantum Source	0	qout0	qin0	N2	Memory q0	(forced)
S	Quantum Source	0	qout1	qin0	R	Memory q0	(default)
S	Quantum Source1	1	qout0	qin0	N1	Memory q0	(forced)
S	Quantum Source1	1	qout1	qin2	R	Memory q2	
RN	Quantum Source	0	qout0	qin1	R	Memory q1	(default)
RN	Quantum Source	0	qout1	qin0	RN	Memory q0	(default)
RN	Quantum Source1	1	qout0	qin3	R	Memory q3	
RN	Quantum Source1	1	qout1	qin1	RN	Memory q1	

Perform 2 swapping in R:



Swaps are done manually with individual gates

Legend:

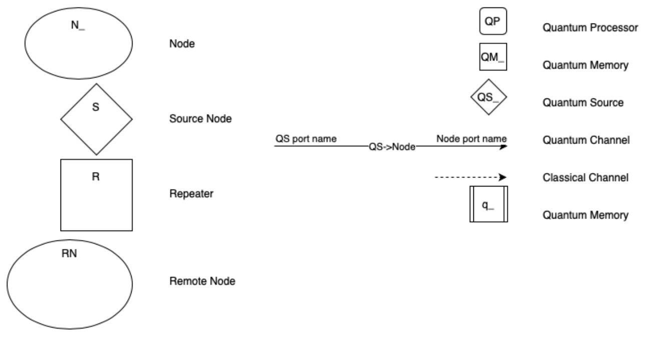


Figure 2: Detailed Architecture diagram of default protocol a