Technische Universität München
TUM School of Computation, Information and Technology

Designs, Protocols, and Software Tools for Quantum Enhanced Networks

Stephen Di Adamo

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz: Prof. Dr. Wolfgang Kellerer

Prüfer*innen der Dissertation:

1. TUM Junior Fellow Dr. Janis Nötzel
2. Prof. Elham Kashefi, Ph.D.
3. Assistant Prof. Dr. David Elkouss

Die Dissertation wurde am 08.06.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 28.11.2022 angenommen.

LEHRSTUHL FÜR THEORETISCHE INFORMATIONSTECHNIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Doktor der Ingenieurwissenschaften (Dr. Ing.)

Designs, Protocols, and Software Tools for Quantum Enhanced Networks

Stephen DiAdamo

Supervisor:	**Dr. Janis Nötzel**
Advisor:	**Dr. Elham Kashefi**
Submission Date:	**April 26th, 2022**

I confirm that this dissertation is my own work and I have documented all sources and material used.

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Technical University of Munich's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to `http://www.ieee.org/publications_standards/publications/rights/rights_link.html` to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Munich, April 26th, 2022 Stephen DiAdamo

List of Articles in This Thesis

8. **S. DiAdamo**, J. Nötzel, "The Impact of Quantum Memory Quality on Distributed Quantum-Accelerated Computation", Publication in progress.

7. **S. DiAdamo**, C. O'Meara, G. Cortiana, J. Bernabé-Moreno, © 2022 IEEE. Reprinted, with permission, from "Practical Quantum K-Means Clustering: Performance Analysis and Applications in Energy Grid Classification", in IEEE Transactions on Quantum Engineering, 2022.

6. **S. DiAdamo**, J. Nötzel, S. Sekavçnik, R. Bassoli, R. Ferrara, C. Deppe, F. Fitzek, H. Boche, © 2022 IEEE. Reprinted, with permission, from "Integrating Quantum Simulation for Quantum-Enhanced Classical Network Emulation", in IEEE Communications Letters, 2021, 10.1109/LCOMM.2021.3115982.

5. R. Parekh, A. Ricciardi, A. Darwish, **S. DiAdamo**. © 2022 IEEE. Reprinted, with permission, from "Quantum Algorithms and Simulation for Parallel and Distributed Quantum Computing", in 2021 SC21 International Workshop on Quantum Computing Software, 2021, 10.1109/QCS54837.2021.00005.

4. **S. DiAdamo**, J. Nötzel, B. Zanger and M. M. Beşe, © 2022 IEEE. Reprinted, with permission, from "QuNetSim: A Software Framework for Quantum Networks", in IEEE Transactions on Quantum Engineering, 2021, 10.1109/TQE.2021.3092395.

3. **S. DiAdamo**, J. Nötzel. "Undoing Causal Effects of a Causal Broadcast Channel with Cooperating Receivers using Entanglement Resources", arXiv preprint arXiv:2102.07427, 2021.

2. **S. DiAdamo**, M. Ghibaudi and J. Cruise, © 2022 IEEE. Reprinted, with permission, from "Distributed Quantum Computing and Network Control for Accelerated VQE", in IEEE Transactions on Quantum Engineering, 2020, 10.1109/TQE.2021.3057908.

1. J. Nötzel and **S. DiAdamo**, © 2022 IEEE. Reprinted, with permission, from "Entanglement-Enhanced Communication Networks," in 2020 IEEE International Conference on Quantum Computing and Engineering (QCE), 2020, 10.1109/QCE49297.2020.00038.

Abstract

Throughout the last six decades, since the invention of the Internet, the use of communication networks has been critical and revolutionary for many fields, from controlling robot fleets, to secure and covert communications for the military, to deep space communication with satellites orbiting Saturn. The development of communication networks is a marvel of human ability. Today, a new networking paradigm is emerging. We are shifting from communicating with information mediums that behave according to classical mechanics to those which behave quantumly. Indeed many obstacles arise when developing quantum networks and the networking theory developed over these last decades is not directly applicable. Quantum systems are much more complex to work with and many challenges arise when designing future quantum networks. By overcoming these challenges, quantum networks will bring applications that cannot possibly be achieved via classical networks alone, but still, the full suite of applications for quantum networks is not yet known. In this thesis, we, therefore, aim to bring to light more potential applications of quantum networks.

Compared to the information mediums that behave classically, various properties of quantum systems allow for these further applications. One such property is called quantum entanglement. Independent quantum systems can be correlated in a particular way such that regardless of the distance separating the systems, by observing one of the systems, the other entangled systems are instantaneously affected. Entanglement, therefore, becomes a valuable commodity when dealing with multiparty communication and can even be used to accelerate transmission rates across quantum communication channels. In practice, distributing entanglement in a quantum network, because of the challenges of distributing quantum systems, is yet a more challenging problem. A large part of the thesis, therefore, explores the process of distributing entanglement efficiently and then consuming it so that transmission rates over quantum channels are, on average, increased. We study the effects of entanglement for accelerating communication rates from a physical, link, and network layer perspective, designing protocols for entanglement-assisted communication and methods for optimally moving entanglement around in an entanglement-enhanced network. In a more theoretical framework, we explore a communication scenario in which many users cooperate to decode messages. By using entanglement, they can decode incoming messages with an advantage over a purely classical strategy.

Another highly important application of quantum networks next to communication is the ability to connect quantum computers to perform distributed quantum computing. Indeed connecting quantum computers can be done in a variety of ways, using various approaches and physical mediums. In this thesis, we first explore the state-of-the-art in terms of accessing quantum computers via remote, cloud services to perform quantum algorithms over a network. We benchmark these remote quantum computers for their ability to perform quantum clustering algorithms. Next, we propose an architecture of parallel and distributed quantum computing, proposing circuit decomposition algorithms and qubit allocation and program scheduling over a network of quantum computers. Such distributed quantum networks require novel control systems and we

also propose such an approach for one. We apply these methods in an explicit use-case, namely the accelerated variation quantum eigensolver algorithm, in-depth.

When designing novel applications of quantum networks, because access to physical hardware is rare, it is important to have the ability to easily simulate such environments. Quantum networking is an emerging field and therefore the scope of available simulation tools is presently narrow. Contributing to the available simulation tools, we develop a quantum network simulation platform to target the application layer of quantum networks. We use the platform to demonstrate various quantum networking protocols and further use the platform to integrate a classical network emulation tool with our novel quantum network simulator. We demonstrate a prototype link-layer protocol for a quantum-enhanced communication network using the tool. In terms of distributed quantum computing, for validating circuit decomposition algorithms, program scheduling, and network control, we develop a framework built on our network simulator that simulates a network of quantum computers.

Quantum networks have the potential to bring an abundance of novel applications to communication networks and distributed computing, leading to more secure communication, accelerated communication rates, and more powerful computing methods. With this thesis, we uncover and explore such applications. Overcoming the hurdles that make reliable and robust quantum communication so challenging will be a problem to face for the coming decades, but with these problems comes the possibility to invent, recreate, rethink, and redesign, bringing on a tremendously exciting period in quantum science and networking in general.

Acknowledgments

The road from the start to the end of a doctorate is never traveled alone and indeed, I am no exception. I am grateful to many people for their guidance, patience, advice, support, and interaction during my studies. The first of which to mention, and of most importance, is Dr. Janis Nötzel, my supervisor. From the beginning to the end of my studies, Janis has provided challenging research topics, the freedom to explore my path, and a great deal of support and opportunity throughout. My time in his group has allowed me to become a far better scientist than when I began, and for that I am grateful. The next two people I mention are Dr. Marco Ghibaudi and Dr. Corey O'Meara whom I worked with during my time at Riverlane and E. On respectively. During these two independent internships, Marco and Corey both provided project supervision, and in the process much creative freedom to explore the topics I am passionate about with careful but crucial guidance to produce the best possible outputs. From these experiences, I have learned, by example, a great deal about both science and strong leadership, and for that am grateful. Next, I mention Dr. Elham Kashefi who has on many occasions welcomed me into her group to visit, present, and discuss. Her openness and creativity have helped me to broaden my scope of the field of quantum networking and my career in general, and for that I am grateful.

From my personal relationships, I first mention my partner Muneeza Qureshi who has provided an endless supply of love, support, and patience, without which my doctoral studies would be incomplete. Throughout the last three years, she has supported me in more ways than she knows, constantly making me a better person, and for that, I am both grateful and forever indebted. Next to mention is my sister Christina DiAdamo, who has, since I moved to Germany six years ago, continuously shown support and care while I have been away. Knowing I have not lost my close family connection after being away for so long has kept me at peace and for that I am grateful. Lastly, I mention my parents Pia and Joe DiAdamo, who have always allowed me to be myself and take risks even when I know it would cause them to worry and stress. The level of support they have given me and the selflessness they have shown me in my life has allowed me to make every decision on my own without sacrifice, always with their full support, which is something I can never repay. For that, I am truly grateful.

Contents

1. Introduction

In the last two to three decades, the level of interest in quantum science and technology has been steadily increasing, and especially in the last five years, the resources allocated to quantum information processing have grown very rapidly [1]. The level of interest and investment does not go unjustified, as the possible applications of quantum technologies apply to many fields. The applications for quantum technologies that are of high interest are in quantum computing, secret key distribution, and most related to this thesis, quantum communication. The development of these technologies brings with them a vast number of novel applications that are either not possible to achieve or are difficult to achieve without the integration of quantum technology, what is usually referred to as the "classical" approach.

Within these fields are specific use cases that can fundamentally change the way we approach current computation and communication challenges. Quantum computing for example uses properties of quantum entanglement and superposition to execute algorithms that can take a challenging problem such as prime factorization, the method on which most Internet encryption is based on and which classical computing methods struggle to perform and reduce the complexity dramatically [2]. Quantum Key Distribution (QKD) is another application that uses the property of unclonability of quantum systems to distribute secret keys among communicating parties. QKD is a large driver for the industry development of quantum networks and it promises communication networks with a higher level of privacy and security than what can currently be achieved [3, 4]. With regards to quantum communication, it has been shown theoretically that when two parties share entangled quantum states, subsequent classical communication over a quantum channel can have a vastly increased communication rate [5].

Although the many applications that come with the integration and development of quantum technologies into our current computing and communication systems can be highly beneficial, the challenges in which to overcome to do so are vast. Quantum systems are very fragile and many environmental factors can influence and destroy the information contained in a quantum system. To perform complex quantum algorithms, methods of quantum error correction will be needed, potentially increasing the number of required qubits and control instructions drastically [6, 7]. Transmitting single quantum states over a fiber-optic cable causes loss and error scaling poorly with the length of the connection, making long-distance quantum key distribution challenging without quantum repeater technologies [8]. To perform entanglement-assisted communication requires that quantum entanglement can be distributed and stored for periods, requiring quantum memory technologies [9].

The promise of the vast applications that come with quantum technologies, therefore, goes hand-in-hand with the difficult engineering challenges that come when developing the technology to perform the applications. Nonetheless, the history of human nature has shown repeatedly with the advent of many technologies that we can persevere and overcome difficulties. With this thesis, we explore various use cases and implementations to further develop the number of applications for quantum technologies, benchmark the current state-of-the-art quantum technology, and develop architectural

plans for the construction of quantum networks. In this introductory section, we explore what differs quantum systems from classical systems. We review the mathematical models for quantum systems and quantum channels and the fundamentals of quantum computing. The following chapters of this thesis go on to develop and analyze protocols for entanglement-assisted communication networks, propose an architecture for a distributed quantum computing network, and finally develop various simulation platforms for the further study of quantum networks. Overall, the strong drive we are currently witnessing in both academia and industry to further develop quantum technology shows strong promise that the challenges faced when building quantum technologies will be overcome, and with this thesis, we aim to further this development along.

1.1. Modeling and Manipulating Quantum Systems

Classical computing and communication technology are based on the fundamental unit of information called the "bit". A bit is an abstraction of objects that can contain one piece of information. Usually, we represent a bit by a 0 or a 1, up or down, or true and false. Bits can be implemented using a vast number of physical objects, but in our commonly used technology, the most important is the transistor. Transistors have been miniaturized to a scale that millions of them are integrated into commonly available computer chips, and even so far that the effects of quantum mechanics are a real concern, as the effect known as quantum tunneling starts to occur.

Bits, albeit the most common way to encode information, are not the only basis for computing or communication technologies. Indeed, the concept of quantum computing and quantum information theory have become successful fields of research, making use of the additional properties of another medium for information storage, namely the "qubit". In the quantum setting, a *qubit* is an analogous unit of information with two states, that can contain one unit of quantum information. Mathematically, a qubit is represented by a complex vector in a Hilbert space. The key difference between a bit and a qubit is that a qubit can be in a superposition of the two states and the quantum superposition is usually represented as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbb{C}$, such that $|\alpha| + |\beta| = 1$. Using the basis vectors $|0\rangle = (1,0)^T, |1\rangle = (0,1)^T$, the vector $|\psi\rangle$ is a vector in the Hilbert space $\mathbb{C}^2$. These types of quantum systems that can be written in this vector form are quantum systems that are in what is called *pure* quantum states, where those that cannot are considered in a mixed state.

Although we can—in theory—*encode* an infinite amount of information into a single qubit using rotational degrees of freedom, *decoding* the qubit to recover all the information is not possible. Qubits exist in a probabilistic mixture of states, and because measuring qubits forces the qubit into one state or the other, a second measurement will not help in determining the probabilistic mixture. Indeed, many copies of the encoded qubit would be necessary to recover, to a finite precision, an arbitrary choice of α and β. This leads to the next step, which is representing multiple qubits. Mathematically, this is achieved using a tensor product. For qubits $|\psi\rangle$ and $|\phi\rangle$, we can create a single quantum state $|\Psi\rangle = |\psi\rangle \otimes |\phi\rangle$. Using this approach, we can grow the quantum system

infinitely large, allowing now for both encoding and decoding approaches.

A feature of quantum systems that do not exist in purely classical systems is that multiple quantum systems can be entangled between themselves. Entanglement is a type of correlation that can— regardless of physical distance—create a type of shared randomness between multiple parties. Shared randomness can be achieved using purely classical techniques, but in 1964, John Bell derived an upper bound on the amount of shared randomness that can be produced with classical systems, and further showed that particular entangled quantum states violate this classical upper bound [10]. Using Dirac notation, we sometimes write a class of important two-qubit entangled states which are known as Bell Pairs or Einstein-Podolsky-Rosen (EPR) pairs as,

$$\left|\Phi^+\right\rangle = \frac{1}{\sqrt{2}}\left|00\right\rangle + \left|11\right\rangle, \tag{1.1.1}$$

$$\left|\Phi^-\right\rangle = \frac{1}{\sqrt{2}}\left|00\right\rangle - \left|11\right\rangle, \tag{1.1.2}$$

$$\left|\Psi^+\right\rangle = \frac{1}{\sqrt{2}}\left|01\right\rangle + \left|10\right\rangle, \text{ and} \tag{1.1.3}$$

$$\left|\Psi^-\right\rangle = \frac{1}{\sqrt{2}}\left|01\right\rangle - \left|10\right\rangle. \tag{1.1.4}$$

Using these entangled states, we can devise communication protocols that consume entanglement such as quantum teleportation and super-dense coding, creating applications beyond shared randomness, that are unique applications of quantum communication. Entanglement is a property that will appear repeatedly in this thesis and is invaluable when deriving protocols that outperform classical solutions.

To create quantum communication protocols or develop algorithms for quantum computers, it is necessary to be able to encode, manipulate, and read out quantum information. To manipulate quantum systems in a mathematical sense, we use the theory of linear operators, specifically linear operators that are also unitary operators. This means that the operators, when applied to the quantum system, maintain the mathematical properties of the quantum system and it is possible to recover the original system by reversing the operator since unitary operators have an inverse. Measuring quantum systems is not a unitary operation, since, based on Born's rule, measurements force a quantum system into one specific state, destroying the superposition of states. We therefore cannot use unitary transformations for measurements, but rather another class of operators that have the property of positive operator-valued measure. A positive operator-valued measure M is composed of a set of positive semi-definite operators $\{M_i\}_{i=1}^n$, where $\sum_i M_i = \mathbb{I}$. This ensures that indeed all measurement outcomes are accounted for, with which a valid probability distribution over the outcomes can be constructed.

With these ingredients, it is possible to develop complex protocols and algorithms that use the additional properties of quantum mechanics to enhance classical counterparts. Communicating with quantum systems follows much of the same principles as when using classical systems. To send messages using quantum systems, it is still required to encode information, transmit, and then decode. What changes between classical and quantum communication theory is the mathematical model, which needs to be

generalized to accommodate quantum systems.

So far, we saw how to manipulate pure state quantum systems for message encoding as well as the model for measuring them. What is missing from this is the model for transmission, which we will use quantum channels for. Quantum channels are a general concept, and essentially any manipulation of a quantum system that maintains a quantum system can be considered a quantum channel. Throughout this thesis, we will be concerned with a specific quantum channel, namely the (lossless) qubit channel. A qubit channel is a completely positive, trace-preserving map from the set of qubit states to the set of qubit states. Qubit channels that we consider face two types of errors, which are dephasing errors and depolarizing errors which affect the fidelity of the quantum state. These errors appear due to noise in the physical medium transmitting the quantum state and we investigate the effects on communication more deeply in this thesis.

With the ability to transmit qubits, it is possible to perform the superdense coding protocol—the foundation for entanglement-assisted communication. Superdense coding in the qubit scenario is an entanglement-assisted communication protocol that uses a qubit channel and one pre-established EPR pair to communicate two classical bits of information using the qubit channel only once. This protocol effectively doubles the classical communication capacity of the qubit channel, as otherwise only one bit of information can be encoded in a qubit per transmission and shown in the Holevo capacity theorem [11]. Superdense coding is the foundation for the first part of this thesis, and we will explore in depth how a network of channels implementing superdense coding for classical message transmission can be cast into a network setting.

1.2. Quantum Communication Networks

Communication networks—in the classical sense—are a major theme in today's society. Communication networks, such as the Internet, span the entire planet and even go into space via satellite communication. The success of communication networks is a major feat of humanity, allowing us to communicate across oceans in real-time, which would be an impossibility not long ago. Communication networks have a vast variety of use cases such as video streaming, video calling, and online banking, all requiring a unique transport protocol and a varying level of security. There are a vast number of topics surrounding communication networks, far too many to be covered in this thesis, but the topics now need to be well understood by quantum scientists with the coming of quantum networks. The state of quantum networks is much like the state of classical networks when the ideas were still in the infant stage. This time around, we now have the knowledge to learn best practices for the deployment of quantum networks. At the current development stage of quantum technology, we can only be mainly in the planning stage, and so concrete protocols for the varying applications of quantum networks are not yet standardized.

One of the most successful architectural ideas regarding communication networks is the abstraction of the varying required tasks when transmitting information from one point in the network to another into layers. Although quantum network protocols are

still far from being standardized, it is generally agreed upon that a layered architecture will be used in the future [12]. Isolating particular tasks into layers allows for easier development and standardization of network technologies, allowing more applications to run over less complex networks. It further allows changes at a layer level that do not require an entire network overhaul, since the changes would affect just a single layer. Moreover, a layered architecture provides a means of communicating particular networking tasks more abstractly. Quantum networks will no doubt benefit from a layered architecture, but the exact responsibilities of each layer are still open to standardization.

Another point of note is that when we refer to a quantum network, the meaning is a general-purpose quantum network and not a single application QKD network. QKD networks have the specific application of distributing keys in a network of nodes via quantum state transmission or entanglement, and the task of performing QKD can include additional layers, such as key management layers, error correction layers, and privacy amplification layers [13, 14]. In a general-purpose quantum network, these tasks can all be performed, as the goal of a general-purpose quantum network is to both distribute quantum states and entanglement. The specific implementation of the protocol, although, relies on further technological developments.

In this section, we describe the core layers of future quantum networks and the general concepts of what layers should do. Here we restrict our discussion to four network layers, which are clearer to explain, but defining a standardization is still a global work in progress. In classical network models, for example, the OSI model [15], there are usually more network layers but because it is generally accepted that quantum networks will not replace classical networks, it is unlikely that there will be a one-to-one correlation between all of the classical network layers and the coming quantum network layers. The layers we discuss are the Physical, Link, Network, and Application layers. These layers are all present in the OSI model and will also be important layers for quantum networks.

1.2.1. Physical Layer

The physical layer of a quantum network is the quantum channel that connects two devices in the network, be it via optical fiber or free space channels, and is responsible for forwarding arbitrary quantum states. The physical layer is not responsible for performing any logical tasks such as error correction or triggering retransmission, it simply attempts to transmit a quantum state over the channel.

As discussed in [16], the physical layer of a quantum network could involve various technologies that can store quantum information, namely, a quantum memory. Quantum memories can be implemented via solid-state technology such as ion traps or NV centers. Transporting quantum states though will be implemented via optical quantum states. Pirker and Dür, therefore, give the responsibility of converting a state stored in a quantum memory to a flying qubit and visa versa to the physical layer. Moreover, in a quantum inter-network, it could be that the optical frequency in which quantum states are transmitted is not universal, and so the task of frequency conversion is also given to the physical layer. Finally, a unique feature of a quantum network is the ability

to distribute entanglement. Entanglement generation can be probabilistic, and in some cases, probabilistic entanglement generation suffices for a particular application, for example, entanglement-based QKD protocols, whereas in others, for example, quantum teleportation, heralded entanglement is important. Distributing heralded entanglement can be done using a variety of protocols [17]. A task of the physical layer is then to either attempt entanglement distribution or perform a heralded entanglement generation protocol but perform no other action for error correction.

1.2.2. Link Layer

The link layer of a network is generally responsible for ensuring that the information transmitted between two points over the physical layer is transferred reliably, correcting any errors as well as performing retransmission requests. In a quantum network, there are many challenges to overcome concerning the reliable transmission of quantum states, since retransmission is generally not possible due to the unclonability of quantum states. To reliably transmit quantum states in a network, the first generation of quantum networks will use a combination of entanglement distribution and quantum teleportation [18] and therefore the link layer will be responsible for generating robust entanglement.

In [16], the link layer described is quite different from a classical link layer. The link layer proposed builds on a so-called connectivity layer, used for creating robust point-to-point and point-to-multi-point entanglement, generating Bell pairs and graph states. The entanglement generated via the connectivity layer forms a virtual "entanglement network", which is a network of nodes that are connected not via physical channels, but via entanglement. The link layer is then responsible for manipulating the entanglement network in such a way as to provide the service of generating arbitrary graph states between any collection of nodes in the network. The link layer can then use further protocols to distill entanglement, as in each local operation between nodes, the entanglement fidelity diminishes. An alternative proposal for a quantum link layer is described in [19]. This proposal is designed to generate robust entanglement, concerning the user's requirements, at a point-to-point level, relying on the upper layers of the stack to then create distant entanglement. This link layer proposal, while error correction can be integrated, initially uses error detection as a means of determining link quality to then prevent errors in transmission. The motivation for this is that in practice, performing error correction will initially be challenging experimentally.

Overall, any future quantum link layer will be responsible for reliably and robustly transmitting quantum states from one point in the network to another. As the technology develops, more complex methods of doing this will surely become enabled, but still, the underlying goals of the link layer will remain.

1.2.3. Network Layer

The network layer in classical networking is responsible for determining an optimal route in the network from the source to the destination, providing the routing information in an encapsulating network-layer packet to each node along the path to

transfer the link-layer data. Current proposals for network layer protocols for quantum networks, on the other hand, have no concept of packets or packet switching. The goal of a general quantum network is to move quantum states from one point to another in a network, and in the first generation of quantum networks, this is done using entanglement distribution and quantum teleportation. The goal of the network layer is therefore to create distant quantum entanglement using techniques involving quantum repeaters, entanglement swapping, and entanglement purification or distillation.

Quantum repeaters are quantum network devices that can be used to extend the range of quantum transmissions. Because transmitting optical signals is lossy and because signal amplification cannot be used on quantum signals due to the no-cloning theorem, it becomes challenging to directly transmit quantum states over a long range. The quantum repeater acts as a relaying node in the network whose job is to, in the first generation of quantum networks, extend the range of entanglement by using the entanglement swapping procedure [20]. Entanglement swapping works in a three-node setting, take for example a network with topology $A - B - C$, by first establishing two entangled pairs, one between A and B and another between B and C. The goal of entanglement swapping in this case is to use the entanglement between B and C to finally have entanglement between A and C. Essentially, the process consumes the entanglement between B and C using quantum teleportation, teleporting B's part of of the AB entanglement to C. The final state is then an entangled pair between A and C. This process is the basis for generating distant entanglement.

There are various approaches for generating distance entanglement in a quantum network. The most straightforward approach is to determine a path in the network, develop entanglement point-to-point in the path, and then perform an entanglement swapping procedure down the path, extending the method described in the three-node case. Another approach as described in [16] is to generate graph states, a particular quantum state that allows multi-party entanglement, which can be manipulated using local operations and classical communication to then establish a Bell pair between two of the parties in the overall multi-party state. After performing entanglement swapping, the fidelity of the entanglement diminishes and so to restore the fidelity to an acceptable level, the process of entanglement purification can be used [21, 22]. In [23], Kozlowski et al. incorporate all of these pieces, creating a network layer protocol for end-to-end entanglement generation.

These tasks—entanglement swapping and purifying—are all multi-party procedures, and therefore require coordination using network layers protocols. The network layer of a quantum network is thus responsible for orchestrating these routines.

1.2.4. Application Layer

The application layer of a communication network is responsible for setting the user-selected parameters of an application layer protocol and then moving the information to a lower network layer, abstracting away further processing. For example, in the (classical) Internet, requesting the HTML of a website is achieved by setting the parameters to the standardized HTTP protocol and pushing the request to the next layer to be processed through the other layers of the network. From the same point in the

network, one can instead connect remotely to another point in the network using the SSH protocol. There are various application layer protocols, and it is up to the end user to decide which task they would like to carry out.

In a quantum network, the application layer is very much analogous, with the difference that protocols for performing quantum-specific applications like QKD and distributed quantum computing will instead be of more importance. Because information encoded into quantum states is prone to loss and error during transmission, a unique feature of the application layer likely to appear in future quantum networks is a user-defined threshold for acceptable error. This is a key reason why a quantum network will likely come accompanied by a parallel classical network, as many of these parameters will need to be agreed upon between parties before the quantum part of the protocol begins. For some scenarios, like in QKD, errors in the sifted key—the remaining part of the key after the parties agree on the selected bases—can be corrected classically using methods of information reconciliation and so it could be acceptable to have a higher threshold to error. On the other hand, in distributed computing, the fidelity of the quantum state will be of much higher importance, as a low state fidelity could cause errors in the algorithm if too much noise is allowed, and so the threshold for error would be lower. We see therefore that the error tolerance is likely to be an application-specific parameter, at least until methods of quantum state transmission become robust.

1.2.5. Entanglement-Enhanced Classical Networks

Overall, we have described the high-level idea of a layered architecture of a network that is designed to transport quantum states from one point in a network to another. Such a network can be used for many quantum applications such as quantum key distribution and distributed quantum computing. The first generations of the network we described will use entanglement to teleport quantum states, but in the future, it could be that the direct transmission of quantum states is enabled using error correction techniques [18].

In this thesis, we therefore also consider a different type of quantum network which we call an entanglement-enhanced classical network that relies on direct transmission of quantum states—rather than teleportation. This network will have a network stack just like a classical network, except with additional features to enable entanglement generation in the network for purposes of enhancing classical message transmission. The task we focus on most in this thesis is the task of entanglement-assisted classical communication, which we saw in the previous section can have a strong rate advantage over unassisted rates. A general-purpose quantum network could indeed perform the task of entanglement-assisted communication, but a network designed specifically for the task could also have an advantage, as the design would be a bottom-up approach, integrating quantum features into already established networks from the lower layers, rather than a top-down approach which is to build the network with a particular application, in this case reliably transmitting quantum states.

2. Entanglement-Assisted Communication in Quantum-Enhanced Networks

The power of quantum communication over purely classical communication methods lies in the fact that quantum systems can be entangled—a special type of correlation found in quantum systems—adding a fundamental correlation between systems which is not possible to achieve using classical approaches. Entanglement can be used, in theory, to create advanced strategies and communication protocols between many parties such that no classical alternative can outperform them. In this section, we explore two such cases, where with pre-established entanglement, a communication advantage is observed. The first three subsections are related to entanglement-assisted communication.

Firstly, we investigate how the physical-layer properties of a communication channel affect the ability to perform entanglement-assisted communication when entanglement is generated during idle periods in communication. We use simulation to observe how changing the quantum memory storage properties affects the throughput of the system. We extend to a network setting and investigate a multi-party computation setting. In this section, we ignore for now the link-layer protocol required to perform entanglement-assisted communication.

In the next section, we consider a link-layer strategy for performing entanglement-assisted communication. We design a protocol as an initial step for noiseless quantum channels and provide the explicit protocols in detail. Later in this thesis, we will use this link-layer protocol to integrate quantum communication into a classical network scenario.

In the third section, now we review a network-layer perspective for entanglement distribution as well as entanglement re-balancing via entanglement swapping to use entanglement-assisted communication as frequently as possible. We use linear programs to optimize at the link level as well as the network level with a broader optimization scope. We simulate our network layer approach over three network topologies and see in some cases, a significant advantage is observed.

In the last section of this chapter, we see another use of entanglement other than entanglement-assisted communication. Here we explore a broadcast channel communication scenario where one user sends messages over a causal channel, where the receivers of the message must perform a cooperative decryption scheme for any of them to receive the broadcast. We see that with a shared Greenberger–Horne–Zeilinger (GHZ) state, the communication cost of the decoding is reduced by a polynomial factor.

Using entanglement for communication has a variety of applications and the extent of the applications is still a popular area of research. In this section, we aim to widen the scope of known applications and through analysis and simulation, determine the extent of the benefits that can be expected, as well as how feasible the deployment of such systems is.

2.1. A Physical Layer Analysis

Section based on the article: "A Communication Strategy for Entanglement-Assisted Communication"

The field of quantum networking is becoming an increasingly important topic as quantum hardware technologies begin to perform what has been predicted theoretically. Theoretically, networked quantum technologies offer a variety of use cases that promise to outperform current and potentially any future classical implementation. Such use-cases are in distributed quantum computing [24, 25], distributed quantum sensing [26], clock-synchronization [27], quantum key distribution [28], and especially related to this work, quantum communication [29, 30]. On the hardware side, experimentally we have seen non-local control gates between physically separated quantum processors [31], quantum clock-synchronization [32], deployed quantum key distributed networks [33], and recently an entanglement-assisted communication experiment [34]. Moreover, experiments conducted for quantum repeater technology with quantum memories are underway [35]. These results all give promise to the future goal of deploying networked quantum technologies for industrial use cases.

In this section, we focus on entanglement-assisted classical communication over quantum channels. At a high level, a quantum channel is a communication channel that can transport quantum systems from one physical location to another. Realizations of quantum channels are for example fiber optic cables. Quantum systems have a unique property that is not seen in purely classical states called entanglement. Two or more quantum systems being entangled means, irrespective of physical distance, that the systems share a special type of correlation. By measuring the quantum state of one system, one obtains information regarding the state of the other system, so long as one knows the originally prepared state. Entanglement-assisted communication is the communication scenario where, between communicating parties, entanglement resources are established before messages are transmitted, and at the time of message transmission, codewords are encoded into sending party's part of the entangled system to then be transmitted over a quantum channel. The receiving party decodes the incoming quantum system with the assistance of their part of the entanglement resource which they had ahead of time by measuring the transmitted system in a particular way. Entanglement-assisted communication has been proven in a variety of communication scenarios to increase the classical capacity of a quantum channel [5, 36].

Most important to any entanglement-assisted communication scenario is the ability to distribute entanglement so that when a message is transmitted, the required entanglement resources are available at both sides of the transmission. One can consider three possible ways to achieve this: 1) Generate entanglement at the time it is needed and then use it immediately to transmit a message, 2) Generate entanglement before it is needed and store it for later use, or 3) A hybrid of the two. On one hand, generating entanglement at the time it is needed adds a layer of communication complexity to the overall protocol, but reduces the need for highly robust quantum storage devices, and on the other, for storing entanglement resources, one needs access to a robust quantum memory such that the established entanglement resources are used before

they decohere or are lost, a highly-challenging technological problem.

Here we consider a communication protocol for the second scenario and integrate an entanglement distribution protocol which depends on the traffic of the network. In Fig. 2.1 is a depiction of the single link communication setting. What we consider is, a sender and receiver who share a synchronized quantum memory used to store entanglement resources, and the link connecting the two parties is a quantum channel. The sender additionally has a classical message buffer which receives bit-strings of some fixed length with some probability per time step. In the first part of this work, we define a stochastic process to model this scenario. At a high level, the process is that when the sender has no bits to transmit, they instead generate several entanglement resources and share half of each pair with the receiver to be stored, storing the other half locally. When entanglement resources are available in storage and there are bits to transmit, the sender uses those resources to transmit at the entanglement-assisted capacity of the channel. If no entanglement resources are available at the time of transmission, the sender simply uses an entanglement-free transmission, thereby using the classical capacity of the channel.

For a scheme to move from theory to practice, there are many hardware parameters to first consider determining, at least in theory, if there are advantages to gain worth the effort of developing the technology. In this work, we aim to add clarity to the questions regarding what quality of quantum memory is necessary to perform our protocol. After defining a general model for the communication scenario, we consider two cases for entanglement-assisted communication. For the first case, we consider just a single link between two parties. We analyze a communication scheme using a qubit model and superdense coding, transmitting two classical bits per channel use, and generating one entanglement unit per entanglement transmission when idle. We model this scenario in simulation using noise models for memory coherence. In our analysis, we ignore the loss effects caused by the channel and suboptimal writing or reading efficiency of the memory, as the effects trivially reduce the rates, that is, a rate reduction directly proportional to the loss probability. Secondly, we consider a four-node network under the same model, again analyzing the effects of memory noise. For the final sections of this work, we consider a model for a quantum-enhanced data center. The data center model is that each computing node in the network is connected via a network that is performing our communication protocol for entanglement-assisted communication. When the computing cluster needs to communicate with other nodes, they attempt to boost their transmission rate using entanglement-assisted communication, where the entanglement is generated during iterations of an algorithm computation.

2.1.1. Review of Related Work

Past work related to the topics covered here is those which are related to quantum channels that have a queuing aspect, co-dependent queues, entanglement distribution protocols, and trading off entanglement resources for achieving entanglement-assisted capacities. In this section, we review various results and discuss their relation to the present work. Overall, this article offers a unique perspective to analyzing how building-up entanglement resources, while the network traffic rates are low, provides a

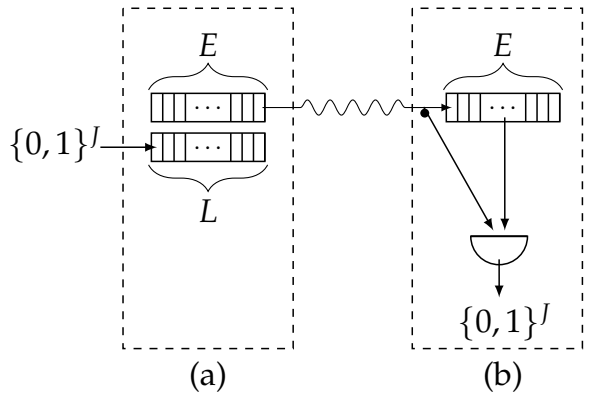


Figure 2.1.: A depiction of the buffered entanglement channel as a communication system. A communicating party in (a) has a classical message buffer of length L in which binary messages of length J arrive and are stored. Further, at (a) is a quantum memory with E slots used for entanglement buffering where entanglement is generated and stored. Connecting parties (a) and (b) is a quantum channel, where classically encoded quantum messages are sent, depending on the state of the entanglement buffer. At (b) is another entanglement buffer that is perfectly synchronized with the one at (a). Depending on the state of the entanglement buffer, messages are either decoded with a measuring device with entanglement assistance or not.

communication advantage in various network settings.

The first result we review is that of Mandayam et al. [37]. In the work, a communication setting where a sequentially processed stream of qubits is considered, where before the qubits are processed, they are stored in a quantum memory where processing is executing a quantum operation on the qubit or the transmission of the qubit over a quantum channel. During the time between when qubits enter the memory and are processed, the qubits decohere, losing information with time. The system is modeled after a single server queue and derived in the work as an expression for the classical capacity in terms of the waiting time of the qubit. Here, the analysis is performed on a queuing model that has unlimited storage space for arriving qubits being processed in a First-Come-First-Served order. Decoherence on the waiting qubits is modeled as a quantum channel which depends on the waiting time of the qubit. The key result of the work is a classical capacity formula for this so-called quantum-queue channel. Overall, the result differs from the present in a few ways. Firstly, the queued quantum systems that we consider are only used for storing entanglement. For data, there is a classical message buffer that stores classical bits of information to be encoded in an entanglement-assisted way. Once encoded in the stored entangled quantum system, the quantum system is then transmitted over a quantum channel to be decoded at the receiver. Here we use the entanglement assisted classical capacity of the channel to transmit at an accelerated communication rate, a different type of capacity than the stricter classical capacity.

A network of nodes that can build up and re-distribute entanglement among them-

selves is analyzed in [30]. In this work, various network topologies are considered, and a method of entanglement generation is proposed to maintain synchronized quantum buffers. This work deals with the specific case of noiseless qubit storage and superdense coding between the network nodes. Moreover, the work focuses mainly on the network layer of the stack. In this work, we take a similar approach, building on the results of [38] and focusing on the link layer. Here we are more concerned with noise models in quantum storage.

A method for using quantum channels to transmit a mixture of classical information and entanglement is analyzed in [39]. Here the method of "trade-off" coding is introduced which, for a given entanglement-consumption rate, some of the transmission uses entanglement-assisted coding and the remaining part of the transmission without entanglement. This method of transmitting with and without entanglement in one transmission outperforms a more common approach known as time-sharing, where some transmissions are only without entanglement assistance, and some are with it. [39] takes the perspective of finding the ultimate rates, but consideration for what hardware specifications are needed to execute such a protocol is not considered. In this work, our queuing model takes a time-sharing approach as a first step as our focus here is to benchmark quantum hardware for their ability to perform entanglement-assisted communication. We aim to clarify how good quantum hardware should be to perform entanglement-assisted communication at before moving to the more complex coding scenario of trade-off coding.

For a communication channel using an auxiliary channel for entanglement generation along with another quantum channel for data transmission, Djordjevic has considered the transmission rates with a bosonic model [40]. What is shown is that for such a channel, using the auxiliary channel, not for entanglement generation, but rather message transmission, the communication rates will always be higher compared to entanglement-assisted communication rates. Indeed, this work also does not consider that the entanglement can be built up and stored ahead of time. Storing quantum states for very long periods is beyond the ability of current technologies, and not making such considerations shows us that without such technology, entanglement-assisted communication may never provide an advantage over traditional methods. What we determine in this work is how much of a communication advantage is there when such memories exist, and if it is enough of a reason that additional efforts for building robust quantum memories for communication will lead to a large enough payoff.

The effects on entanglement concerning memory coherence times have been investigated in a simulation setting in [41]. In this work, the authors simulate entanglement sharing between two end nodes for entanglement distribution and consider an entanglement management scheme for optimizing performance. In the work, the application tested is the generation of entanglement over a linear chain of network nodes using an entanglement swapping procedure. The key metric considered is the entanglement generation rate over a system with length-dependent loss in the channel. The present work differs in that we consider a specific application requiring the consumption of entanglement for classical message transmission. In this initial work, we concern ourselves only with how the effects of memory influence the communication of classical messages.

2.1.2. A Queuing Process for Entanglement-Assisted Communication

We now discuss how one can use entanglement in a communication protocol to improve communication rates. The model we use is for a communication scenario where two communicating parties share a synchronized quantum memory which, depending on the network traffic level, will fill up and deplete to make use of entanglement-assisted communication rates.

In this section, we consider a communication single link, whereas later in Section 2.1.5, we will consider more complex network topologies. The single link model is composed of the following: two buffer types—one for classical information and one for entanglement storage; and a single quantum channel connecting the two parties. For this section, we first mathematically formalize an entanglement buffer and a buffered-entanglement quantum channel and then define the process that governs the communication scenario. From there, we can define a stochastic process for the model which governs the behaviors of the system.

Definition 1 (Entanglement Buffer)**.** *An Entanglement Buffer (EB)* $\mathfrak{EB}(M, \alpha)$ *for two communicating parties is defined by parameters* $M \in \mathbb{Z}$ *the size of the buffer and a rate* $\alpha \in [0, 1]$ *which represents the probability that after each time unit a stored entanglement unit becomes unavailable.*

For the communication channel which makes use of an EB, we define a Buffered-Entanglement Channel (BEC).

Definition 2 (Buffered-Entanglement Channel)**.** *A BEC* $\mathfrak{J} \coloneqq \mathfrak{J}(G, C, L, J, C_{EA}, C_C)$ *is a quantum channel* $\mathcal{S}(\mathcal{H}_A \otimes \mathcal{H}_S) \to \mathcal{S}(\mathcal{H}_B \otimes \mathcal{H}_S)$ *between parties A and B with shared access to* $\mathcal{H}_S$ *where* $G \in \mathbb{R}^+$ *is the amount of entanglement bits (ebits) that can be generated per transmission,* $C \in \mathbb{R}^+$ *is the number of ebits consumed to transmit* $C_{EA} \in \mathbb{R}^+$ *bits with one transmission, or without entanglement* $C_C \in \mathbb{R}^+$ *bits can be transmitted,* $L \in \mathbb{N}$ *is the maximum number of bits that can be queued at A, and J is the number of bits that an arriving job contains.*

With this definition, we can describe stochastic processes for a BEC. We define firstly a stochastic process for a system that can hold just one job in its processing queue, that is, $L = J$. We choose this restriction because, for our first analysis, we analyze the system analytically, arriving at closed forms for throughput and job drop rates. The restriction of $L = J$ simplifies this. In the next sections, we extend the process to handle multiple jobs at once, which becomes challenging to analyze analytically, especially with noise models, and therefore we will use simulation.

Definition 3 (One job BEC process)**.** *A stochastic process for a BEC* $\mathfrak{J}(G, C, L, J, C_{EA}, C_C)$ *is defined with a state space consisting of: 1)* $\mathbf{S}_B \coloneqq \{0, \ldots, L\}$*, a classical buffer at the sender; 2)* $\mathbf{S}_E \coloneqq \mathbb{N}$*, the state of the EB shared between both parties; 3)* $\mathbf{S}_T \coloneqq \mathbb{N}$ *the number of transmitted bits; 4)* $\mathbf{S}_J \coloneqq \{0, A, N\}$*, the active job processing type; and 5)* $\mathbf{S}_D \coloneqq \mathbb{N}$ *the number of dropped jobs. A state of the system is* $s = (b, e, t, j, d) \in \mathbf{S}_B \times \mathbf{S}_E \times \mathbf{S}_T \times \mathbf{S}_J \times \mathbf{S}_D$*. The variables for the "one job" stochastic process are* $r \in [0, 1], \lambda \in [0, 1]$*, and* $\tau \in \mathbb{N}$*. The variable r represents the probability that a message arrives after each time step,* λ *is the*

probability that after each time step, τ units of entanglement arrive from an outside source. The transition matrix for the system is

$$\begin{aligned} T(s'|s) :=& (1-\lambda)\left((1-r)\cdot T_0(s'|s) + r\cdot T_1(s'|s)\right) + \\ & \lambda\left((1-r)\cdot (E\circ T_0)(s'|s) + r\cdot (E\circ T_1)(s'|s)\right), \end{aligned} \tag{2.1.1}$$

with $E(s'|s) := \delta_{s'}((b, e+\tau, t, j, d))$,

$$T_0(s'|s) := \tag{2.1.2}$$
$$\begin{cases} \delta_{e'}(e+G)\cdot\delta_{b',d',j',t'}(b,d,j,t), & j=0, b=0 \\ \delta_{b'}(b-C_{EA})\cdot\delta_{d',e',j',t'}(d,e,j,t), & j=A, b>C_{EA} \\ \delta_0(b')\cdot\delta_0(j')\cdot\delta_{d',e',t'}(d,e,t+1), & j=A, b=C_{EA} \\ \delta_{b'}(b-C_C)\cdot\delta_{d',e',j',t'}(d,e,j,t), & j=N, b>C_C \\ \delta_{b'}(b-C_C)\cdot\delta_0(j')\cdot\delta_{d',e',t'}(d,e,t), & j=N, b=C_C \end{cases}$$

and,

$$T_1(s'|s) := \tag{2.1.3}$$
$$\begin{cases} \delta_A(j')\cdot\delta_{e-C}(e')\cdot\delta_{L-C_{EA}}(b')\cdot\delta_{d',t'}(d,t), & j=0, b=0, e\geq C \\ \delta_N(j')\cdot\delta_{L-C_C}(b')\cdot\delta_{d',e',t'}(d,e,t+1), & j=0, b=0, e<C \\ \delta_0(j')\cdot\delta_0(b')\cdot\delta_{d',e',t'}(d+1,e,t), & j=A, b=C_{EA} \\ \delta_{b'}(b-C_C)\cdot\delta_{d',e',j',t'}(d+1,e,j,t), & j=N, b>C_C \\ \delta_0(j')\cdot\delta_{b'}(b-C_C)\cdot\delta_{d',e',t'}(d+1,e,t), & j=N, b=C_C. \end{cases}$$

To provide a clearer picture of how the stochastic processes T_0 and T_1 in Definition 3 function, depicted in Fig. 2.2 is a high-level flow diagram of the process. We can analyze such a system analytically and determine a closed form for the average, throughout, job processing time, and drop rate. Below we give the formal definition of these values.

Definition 4 (Throughput and Rejected Job Rate)**.** *For $n\in\mathbb{N}$, let S_T^n, S_D^n be the respective total transmitted data and dropped jobs in the process defined in Definition 3 after n time steps. The throughput of $\mathfrak{J}$ is the supremum over all non-negative numbers T satisfying*

$$\liminf_{n\to\infty} \mathbb{P}\left(\tfrac{1}{n}S_T^n \geq T\right) = 1. \tag{2.1.4}$$

The job drop rate is the infimum over all non-negative numbers D satisfying

$$\liminf_{n\to\infty} \mathbb{P}\left(\tfrac{1}{n}S_D^n \leq D\right) = 1. \tag{2.1.5}$$

In this section, we analyze the BEC for an EB when $\alpha = 0$, that is the EB never loses any entanglement due to noise and only the communicating parties will reduce the number entangled pairs in the buffer via consumption.

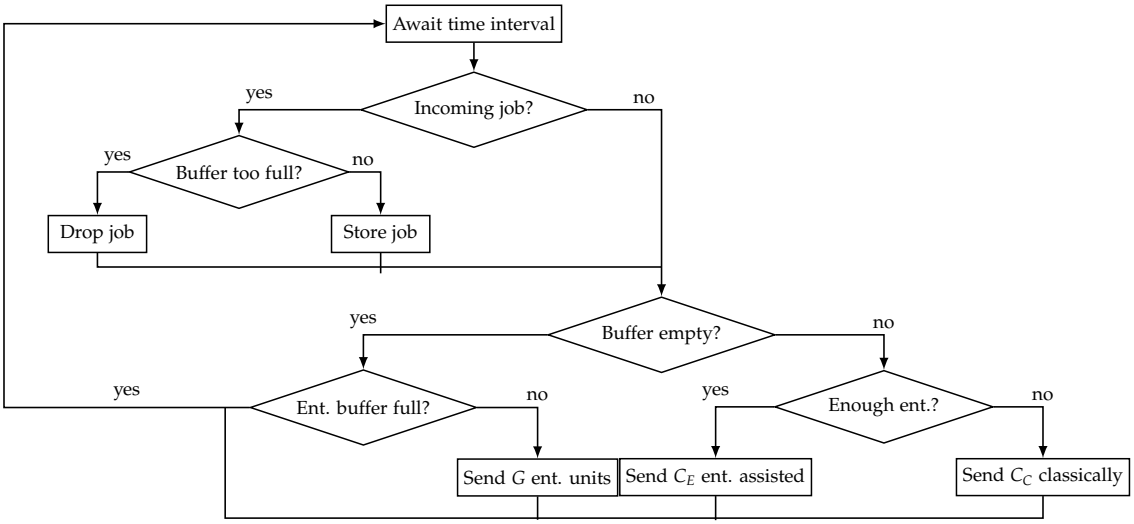


Figure 2.2.: A flow diagram for the queuing processes T_0 and T_1 which governs an buffered entanglement channel's transmission and internal entanglement generation.

Theorem 5. *[38, Theorem 3] When C_E divides L and $C_E/C_C = 2$, the throughput and job drop rates are described by the functions $T(\mathfrak{J})$ and $D(\mathfrak{J})$ of the system parameters $\mathfrak{J} = (G, C, L, J, C_{EA}, C_C)$ governed by the stochastic process in Definition 3 defined via*

$$T(\mathfrak{J}) \coloneqq \frac{r \cdot L}{1 + r \cdot (P(\mathfrak{J}) - 1)}, \tag{2.1.6}$$

$$D(\mathfrak{J}) \coloneqq \frac{r \cdot (P(\mathfrak{J}) - 1)}{1 + r \cdot (P(\mathfrak{J}) - 1)} \tag{2.1.7}$$

where

$$P(\mathfrak{J}) \coloneqq \theta \cdot t_A + (1 - \theta) \cdot t_N \tag{2.1.8}$$

is the average job duration and

$$\theta \coloneqq \begin{cases} \theta', & \theta' \leq 1, \\ 1, & \text{otherwise}, \end{cases} \tag{2.1.9}$$

with

$$\theta' \coloneqq \frac{\frac{1-r}{r}(G' + \lambda\tau') + t_N \cdot \lambda\tau'}{C + \lambda\tau'(t_N - t_A)}. \tag{2.1.10}$$

Further, $t_A = L/C_{EA}, t_N = L/C_N, G' = G/(t_N - t_A)$ and $\tau' = \tau/(t_N - t_A)$.

Corollary 6. *A classical system (one where $t_N = t_A$) is described by*

$$T_{cl}(\mathfrak{J}) = \frac{r \cdot L}{1 + r(t_N - 1)}, \tag{2.1.11}$$

$$D_{cl}(\mathfrak{J}) = \frac{r \cdot (t_N - 1)}{1 + r(t_N - 1)}. \tag{2.1.12}$$

Remark 7. *θ' for the system with parameters $\mathfrak{J} = (G, C, L, t_A, t_N, r, 0, 0)$, that is, with no external supply of entanglement, and where $C_E \ll L$, simplifies to*

$$\theta' := \frac{(r-1)(t_A - t_N)G}{r \cdot t_A \cdot C}. \tag{2.1.13}$$

The formulas for the various rates remain unchanged.

To get an idea of how the system behaves, in Fig. 2.3 is a plot for a system using superdense coding (i.e. $C = 1, C_{EA} = 2$, and $C_C = 1$). We vary the rate of external entanglement λ and plot the statistics against the rate of incoming messages for the average number of transmissions it takes to transmit a single job, the average throughput of the system, and the rate at which jobs are dropped at the sender. The plot of the analytical forms in Eqs. (2.1.6)–(2.1.8) against a simulation of the stochastic process in Def. 3. Overall, with varying parameters of the system, proving the correctness of closed-form formulas quickly becomes a challenging endeavor and likely a novel analysis technique should be used. On the other hand, a straightforward way to analyze the system is through simulations which are relatively efficient and can produce an accurate benchmarking of the system for an arbitrary choice of system parameters. In the next section, we continue with simulation and work towards integrating noisy memory models to benchmark how well superdense coding can perform.

2.1.3. Simulation Setup and Parameter Selection

The scenario we use for simulation, as was in the previous section, is a quantum channel with a classical buffer, quantum memory, and a parameter for the rate of incoming messages. For message transmission, classical information is encoded into qubits and when possible, using pre-established entanglement, transmitted using superdense coding. When there is no entanglement available, qubits are transmitted directly with no entanglement assistance. The simulations contain a quantum channel that is modeled after a lossless 1 *km* fiber with a fixed delay time of 5 μs, roughly the time light takes to travel through a 1 *km* fiber. This work aims to consider short-distance quantum networks as a starting point.

The flow of transmission and processing of the classical messages is as depicted in Fig. 2.2. When there are no messages to process, a single entangled pair is generated between the sender and receiver. The sender and receiver have a synchronized quantum memory which they use to encode and decode superdense-encoded messages. The qubits stored in the quantum memories experience time-dependent decoherence properties, and so with simulated time, the qubits decohere in the memories according to the memory T_1 and T_2 times. Here we are concerned with the quality of storage required to implement the protocol and do not apply noise effects to the writing and reading steps.

The simulations we perform are concerned with how the physical layer—namely the quantum memory—properties affect the throughput and error rate during communication. To simplify this as much as possible, we program the simulation to be independent of any link-layer protocols, but in a real communication setting it would be required for the receiver to distinguish qubits to be stored and qubits to measure.

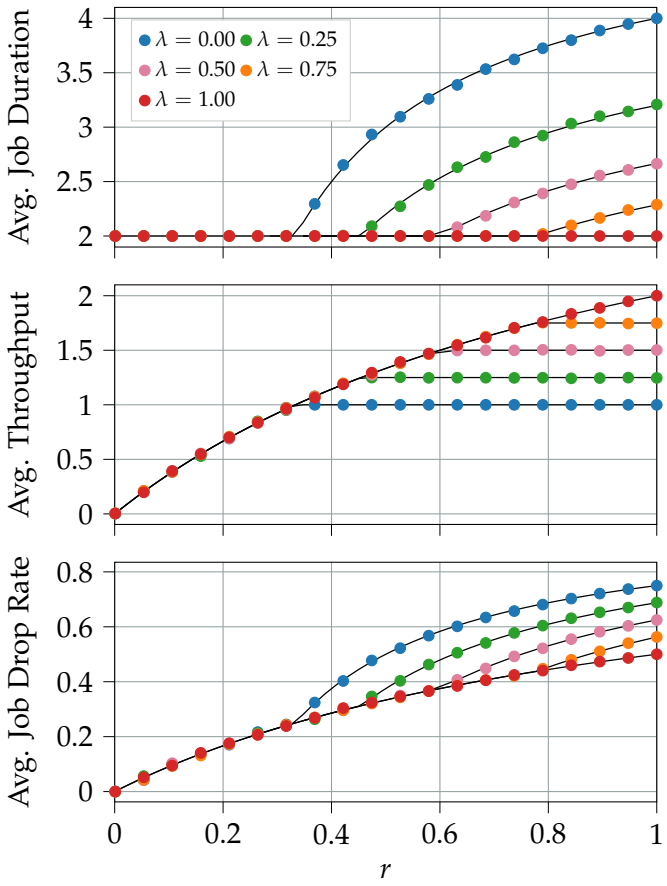


Figure 2.3.: A data stream using superdense coding as the encoding and decoding scheme, i.e. $C_{EA} = 2$ and $C_C = 1$. The associated parameters to the model are $G = 1$, $C = 1$, and we have chosen $L = J = 4$. Varied are the incoming message rate r on the horizontal axis and the external rate of entanglement generation λ and fix $\tau = 1$ in the plots. In all plots, the black line is the analytical formulae of Theorem 5 and the dots are the results of numerical simulation.

To accomplish this in simulation, we use global parameters in software so that the receiver can make the distinction. An initial link-layer approach for this particular communication setting has been considered in [42], where a data frame of qubits is transmitted behind a frame header, indicating how the following qubits should be treated at the receiver. Here, we ignore the link-layer protocol to focus on the best-case effects for the physical layer.

To develop the simulations, we use the quantum network simulator NetSquid [43]. NetSquid is a Python-based software library used for simulating—for one—various physical layer effects in quantum networks. Incorporated into the software are the physical models of various quantum technologies such as quantum memories, fiber optical channels, and photon sources and detectors. NetSquid is developed in a modular way, so that network nodes can contain any variation of the built-in hardware models. We construct a sender and receiver using NetSquid's models for the scenario depicted in Fig. 2.1. The simulation configuration using NetSquid's models is depicted in Fig. 2.4, where the sender's classical message buffer feeds data to the quantum processor.

The exact models we use in NetSquid are as follows. For the sender and receiver memories, we use the `QuantumMemory` object. The quantum memory has parameters for the number of memory positions and the decoherence model, where we select the `T1T2NoiseModel` error model. Each party is also equipped with a `QuantumProcessor`, which can encode and decode a superdense encoded message, whose memory slots are also governed by the `T1T2NoiseModel` error model. The quantum processor can perform a set of physical instructions which perform single and two-qubit operations with 1-time unit and 10-time units respectively. To generate the qubits for transmission, the sender has an EPR generating source as well as a qubit source both implemented using the `QSource` object. With the `QSource`, one can select which type of quantum systems are emitted, in this case, either Bell states or ground state qubits, and at which frequency. For our experiments, we only concerned ourselves with the properties of the memory, and so we allow these systems to emit the expected qubits with certainty, that is, we assume a perfect quantum source. Connecting the two parties is a `QuantumChannel` modeled with the `FixedDelay` delay model.

2.1.4. Point to Point Channels

In this section, we develop and present simulations of the BEC model using varying quantum hardware parameters and analyze the achievable throughput and decoding error rates for various scenarios. The key aspects we are concerned about here are the properties of quantum memories. We investigate how the T_1 and T_2 times of the stored qubits and the number of memory positions available affect the throughput achieved when using the communication setting previously defined for generating or consuming entanglement. These simulations provide an initial estimate for the required quality and quantity of memory positions the quantum memory should have for an advantage to be seen over purely classical communication methods using the principle of generating entanglement while idle.

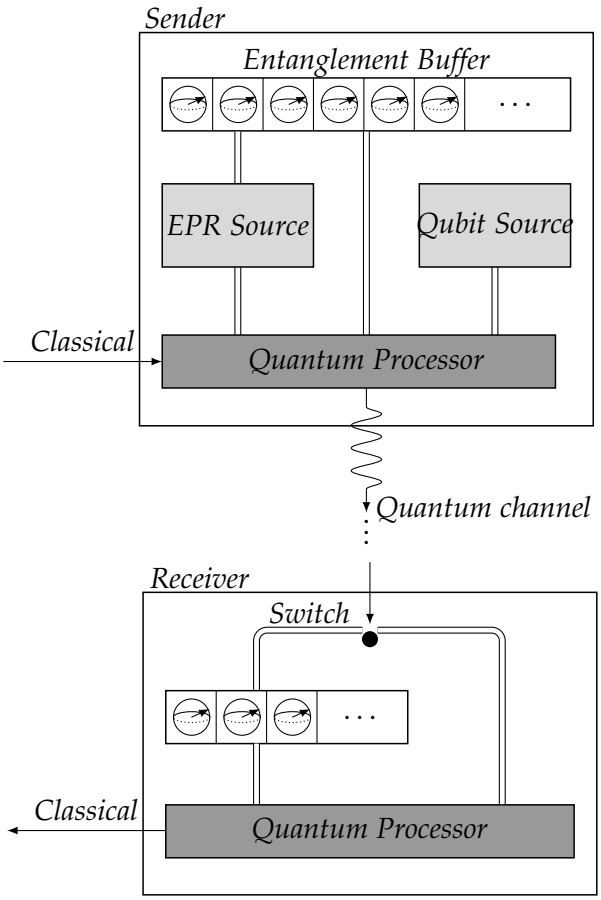


Figure 2.4.: The sender and receiver node configuration using the NetSquid simulations. The sender and receiver each have a quantum memory, which is synchronized. The sender has two quantum sources, one for generating qubits and one for generating EPR pairs. The sender has a classical message buffer that feeds data to the quantum processor which prepares fresh qubits from the source or qubits removed from the memory for transmission. Once transmitted, the receiver can process the qubits or store them, and outputs a classical message accordingly.

Simulation Configuration and Execution

We use the simulation configuration as explained in the previous section. Here we describe the precise program logic of the simulation. To begin, each party starts with empty entanglement buffers and the sender's message buffer is empty. With some probability r, a bit-string of length J fills a classical buffer of capacity L if possible, otherwise, the message is dropped. If no message arrives, the sender uses its qubit source to generate an EPR pair, storing half of the pair in its memory and transmitting the other half over the channel. We test two approaches here, namely, if the quantum memory is full to replace the oldest qubit or to simply discard the newly generated qubit.

To transmit messages, when there is no entanglement stored in the memory, 1 bit of the message is encoded into a single qubit generated by the secondary qubit source which produces ground-state qubits and is sent over the fiber after encoding in the quantum processor. When entanglement is instead present in the memory, 1 qubit is removed from the memory using the selected queuing priority. Here we test two approaches which are using a First-In-Last-out (FILO) priority and a First-In-First-Out (FIFO) priority. The qubit removed from memory is then put into the quantum processor to be encoded and transmitted.

We run each simulation with different parameters for 2,500,000 time iterations, which at this the simulation time scale represents 2.5 milliseconds. We compare the transmitted messages to those received to determine the percentage of the messages that were transmitted with and without error. To compute the average throughput, we determine the fraction of the number of bits transmitted in the total iterations multiplied by the decoding error rate. We simulate 15 different settings grouped into three different simulations. Firstly, we replicate the simulation from the previous section, using $L = J = 4$ varying the T_1, T_2 times of the model and fixing $E = 200$ memory slots. Next, we repeat the process using $J = 4$ and $L = 5J$. Finally, we fix the $T_1, T_2 = 1100, 1000$ ns and vary the number of memory slots in the quantum memory. The sender attempts to send a message every 10 *ns*—simulating a maximum of 400 mb/s link at most—or else generates entanglement using an entanglement source.

Analysis and Discussion

We review the results of the simulations, the first of which are plotted in Fig. 2.5. In the upper plot, we see the average throughput results for varying T_1 and T_2 times of the buffer against a varying message arrival rate. As expected, the longer the T_1 and T_2 times, the smaller the error rate and thus a better throughput. We also observe that under a certain threshold for the T_1 and T_2 times, superdense coding performs far worse than in a purely classical way, where the dashed line represents the entanglement-free case. We can see when the T_1, T_2 times of the memory are $(11, 10)$ respectively, since the messages are 4 bits long, until $r \approx 0.38$, the transmission produces completely random outputs with a decoding error rate of $1 - 1/2^4 = 0.9375$. If we compare to the plots in Fig. 2.3, we see the point $r \approx 0.38$ plays a special role as well, where having no externally generated entanglement sent into the system, that is, when $\lambda = 0$, begins to affect the trend. The point is exactly that at which the majority of the messages are sent

using classical means over entanglement-assisted.

In the plot of the error rates, we note that the red curve representing the T_1, T_2 times $1100, 1000$ ns is not non-increasing, and at around the point $r \approx 0.38$ raises slightly to a local maximum before beginning to steadily fall. This is because the system uses FILO ordering for entanglement consumption, leaving some entangled pairs to decohere for long periods. With the red dashed line in the same plot, we show the error rates for a First-In-First-Out (FIFO) ordering which does not increase in error rate but performs much worse overall as FIFO will choose the entangled pairs that have been stored the longest when performing entanglement-assisted communication. Another point of note is when we apply the approach of discarding old entangled pairs, replacing them with fresh pairs when the memory is full. The throughput and error rates in this case are shown with dotted lines. Here we see significant improvement at low traffic rates, eventually converging to the same trend at higher rates.

In Fig. 2.6, we repeat the simulation, but increase the message queue size so that at most 5 jobs are in the message queue at once and observe similar trends but with the plots squeezing more tightly to the smaller values of r. Because fewer jobs are dropped with a larger message queue size, the system is idle less often and therefore has fewer opportunities to generate entanglement. Indeed in this scenario, the average throughput in the classical case (the dashed black line) outperforms the three cases of noisy memory that we analyze, which is in contrast to the single job buffer. Moreover, comparing the classical trend to the trend for the perfect quantum memory, there is only a small portion of the incoming job probability domain in which superdense coding with GEWI outperforms the classical case. For the error rates, we again compare the FILO and FIFO consumption of entanglement. We see again that FIFO ordering performs poorly, no better than random decoding. Again, the dotted trend represents the case where entanglement bits are refreshed, and significant improvement is observed.

The final set of simulations we performed in this section is comparing the performance when varying the size of the quantum memory. In Fig. 2.7 are the trends for four memory sizes which all share the properties for $T_1, T_2 = 1100, 1000$ ns. What we observe is that with this choice of T_1, T_2 times, of all the entanglement buffer sizes that we tested, when the rate of incoming messages is low, a smaller quantum memory size performs better than a larger one, and indeed when $E = 10$, until $r \approx 0.35$, outperforms the cases when $E = 200$. This is due to the aging of the entangled pairs and using a FILO consumption without replacing the older stored pairs. Since the rate of incoming messages is too low, the stored EPR pairs decohere before they can be used, whereas in the $E = 10$ case, the buffer is emptied more often and so the EPR pairs are younger on average. When $r \gtrsim 0.38$, the rate at which entanglement is consumed increases enough until both $E = 10$ and $E = 200$ perform equivalently. With the entanglement replacing strategy, we see that for low traffic rates, replacing the entanglement with fresh units improves performance significantly, reducing error rates drastically and thus improving the overall throughput.

Overall, we can conclude the following: Firstly, not only do the T_1, T_2 times greatly affect throughput, but also the order in which the entanglement is consumed plays a large role. By testing three orders of magnitude in this case, we can already observe T_1, T_2 values that perform sub-classically, and those which perform super-classically. More-

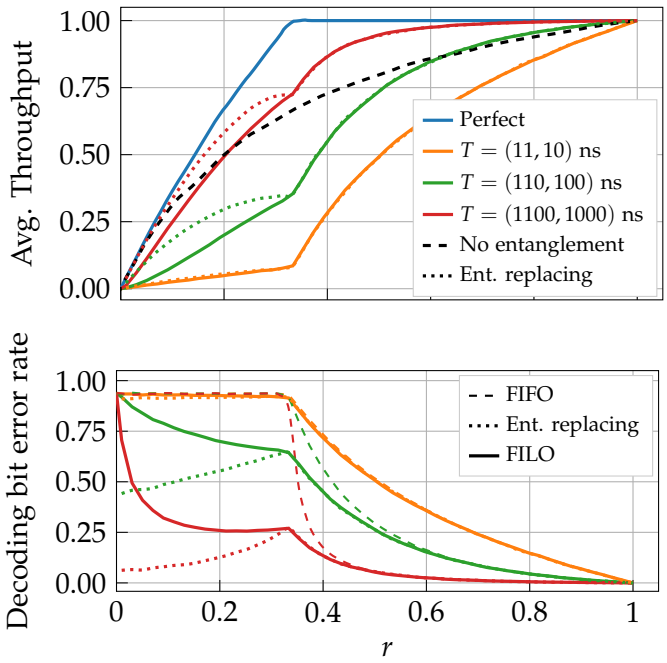


Figure 2.5.: A simulation of a BEC using the NetSquid simulation platform. In the simulation, two nodes are separated by a fibre 1 *km* long with a fixed delay of 5000 *ns*. In the first simulation, Each node has an entanglement buffer of 200 memory slots which have a specific T_1 and T_2 times associated with them.

over, common orderings like FIFO and FILO are sub-optimal and so future work will therefore include a deeper analysis into how the ordering of entanglement consumption optimizes the throughput of the system; Second, with sub-optimal entanglement consumption orderings, we see that the size of the entanglement memory plays an important role as well. With shorter T_1, T_2 times, using a smaller entanglement buffer will ensure the entanglement buffer is emptied more often, thereby maintaining fresher entanglement units on average and hence better decoding error rates. Lastly, in all cases, replacing stored EPR pairs with fresh pairs will improve the decoding error rates, thereby improving the total throughput.

2.1.5. Networks of Buffered-Entanglement Channels

In the previous section, we analyzed the properties of a single link under the queuing process of GEWI, and so the next naturally arising question is: What are the properties of a network of such links? These types of questions have arisen in other queuing theory contexts, specifically, Jackson or Kelly networks for various queuing models [44, 45]. In this section, we take a similar approach and analyze a network of such queues. We begin by defining the network of BECs. For our analysis, we model the network of queues after an open queuing network—a network with data sources and sinks where data enters from outside the network at the source, routes itself in the network, and eventually exits at the sinks. For more details regarding networks of queues and the

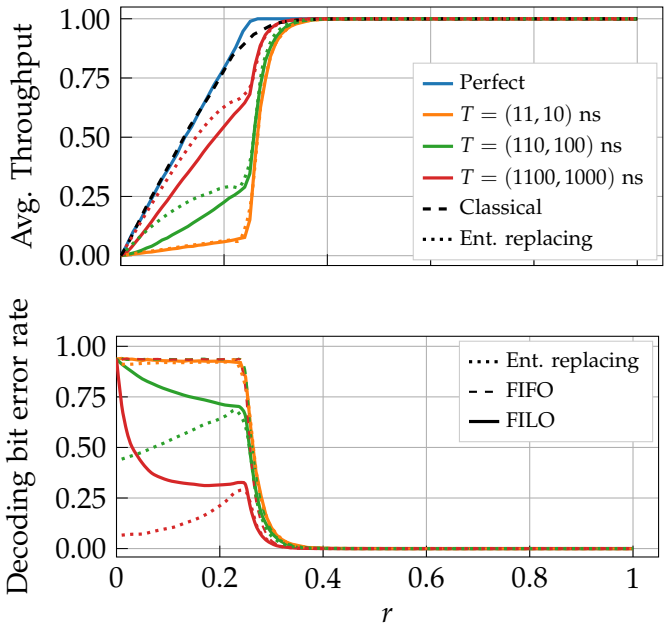


Figure 2.6.: The same configuration as Fig. 2.5 but the size of the message buffer is increased to $L = 5J$.

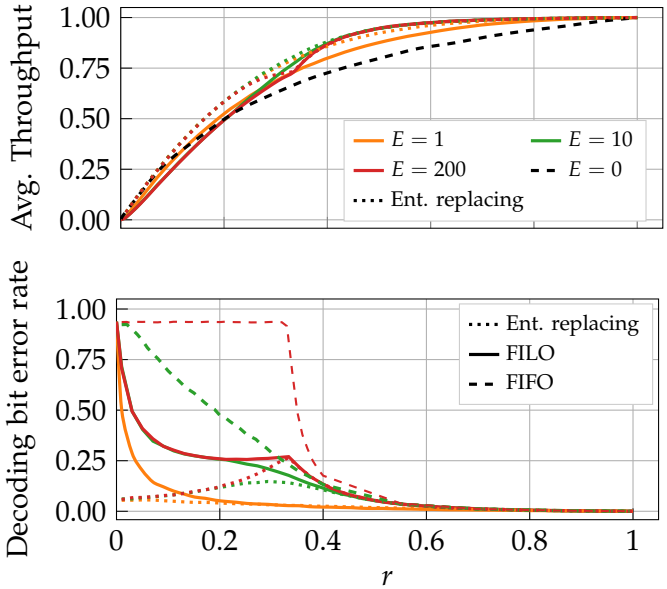


Figure 2.7.: T_1 and T_2 times fixed at 1100 ns and 1000 ns respectively. The size of the entanglement buffer is varied.

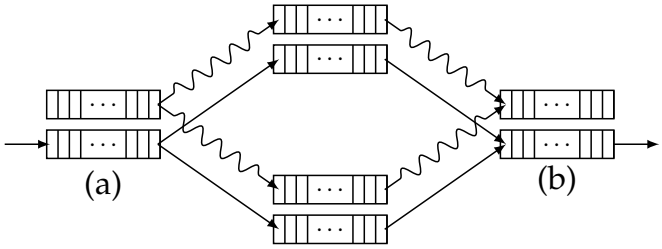


Figure 2.8.: A depiction of the network of entanglement-buffered channels. Each connected pair of nodes have a shared buffer and each process its classical message queues according to the transition matrix of the BEC. Here (a) is the source node, where classical data arrives to be sent into the network and (b) is the sink, where classical data will be processed and removed from the network.

different types of queuing network models, we refer the reader to [46]. To start, we define a buffered entanglement channel network.

Definition 8 (Buffered Entanglement Channel Network)**.** *A Buffered Entanglement Channel Network (BECN) $N = (G, R)$ is made up of a directed graph $G = (V, E)$ where each $v \in V$ represent the network nodes and the edges $(u, v) \in E$, represent that node u shares a BEC with node v. G contains a single source node s and a sink node d. The source node receives classical messages from outside of the network at a rate of r messages arriving per unit time and stores the messages in its buffer if possible. The sink node processes messages and takes them out of the network. A BECN also contains a routing algorithm $R : V \times V \to E$, which provides the next edge in a path within G, where R need not necessarily be deterministic. R can moreover use the current state of the networks to determine the next step in a route.*

For the remainder of this section, we present simulation results using the four-node network topology depicted in Fig. 2.8. As in Section 2.1.4, we again concern ourselves with how the T_1, T_2 times of the quantum memory affect the transmission throughput and decoding error rates.

Simulation Configuration and Execution

The configuration in this simulation is the following: For the nodes with outgoing connections, using the stochastic process defined in Fig. 2.2 for each connection, the node transmits data or entanglement depending on the state of its message buffer and the choice of routing. When qubits are sent to another node, they are firstly measured, thereby extracting the classical information, and reprepared to be transmitted again depending on the state of the entanglement buffer. This implies that each connection builds up and consumes its own entanglement, with one pair of synchronized entanglement buffers per connection. In Fig. 2.2, the nodes other than the source (a) have a larger message buffer so that messages that are put into the network do not drop en route. Decoding at re-encoding at the mid-way nodes eliminates the need for relaying the quantum traffic over multiple network hops as well as the need for entanglement

swapping, simplifying the technological requirement for a physical realization in this first stage.

The three nodes with outgoing channels all have the components as seen in the upper portion of Fig. 2.4. The three that have incoming channels, components for the lower portion of Fig. 2.4 are also added so that the middle nodes can both receive and transmit qubits. The simulation again uses a fixed delay model based on a 1 *km* lossless fiber model for each connection in the network. For the message sizes, we use a similar setting as in the single link case, which is that the source node can first store just one message at a time with $J = L = 4$ bits. We repeat the simulation except with set $L = 5J$, allowing 5 messages in a buffer at once. For each time step, a message randomly arrives at the source and if there is data to transmit, an outgoing edge is selected and data is transmitted. For the edges where no data is being transmitted, instead, entanglement is generated to be stored. Each node can store $E = 200$ entanglement units with varying T_1, T_2 times for the simulations. In this case, since we have timing, the time that the polls for new messages every 10 *ns*, thereby simulating a maximum of 400 mb/s network throughput.

In each of the simulations, we use a simple routing approach. We set R in Definition 8 to be such that it chooses the outgoing link that has the most stored entanglement at the time of transmission. The result of this is that entanglement-assisted transmission is used as often as possible. Alternative approaches can be used, for example in [30], the stored entanglement over entire paths is considered for routing. Given the single source of network traffic being analyzed in this case, the alternative approach is equivalent. We again bypass any link-layer protocols of the network by using a global variable in software that allows the receiver to distinguish when data arrives versus when entanglement arrives.

Analysis and Discussion

For the setting when $J = L = 4$, the average throughput, and decoding error rate are displayed in Fig. 2.9. Compared to the single-link case, the average throughput and the decoding bit error rates differ quite dramatically compared to those in Fig. 2.5. The main difference is, with two outgoing links from the sender, there is always one link idle during any transmission—one link is transmitting data, and the other is idle. This implies that for each transmission of data, one EPR pair is generated across the idle link and therefore entanglement is always available across the two links. Using superdense coding is therefore always possible for the first hop and an average throughput from the sender is maximized at two bits per unit time under perfect memory settings. What becomes clear is that a quantum storage of size $E = 200$ is excessive and $E = 1$ for these simulations produces the same average throughput and error plots for the different storage parameters. We, therefore, do not show plots for the throughput and decoding error rates for varying E in this case. In the latter half of the network, the middle nodes will have the same property where one will always be idle, therefore having the ability to generate entanglement with the receiver. In this case, since $E = 1$ will achieve the same throughput and average error rate, we further do not need to consider entanglement queuing practices, since for a queue with one position, all practices are

equivalent. With dotted lines, we show the trend for replacing old entanglement units, which demonstrates, as with the point-to-point case in the previous section, a significant improvement in error rates and thus throughput.

What differs in the network setting as compared to the single-link setting is that, because entanglement is always available hop-by-hop, entanglement-assisted communication is always used, thereby leading to larger error rates with respect to the memory noise. Here we see that for $T_1, T_2 = 11, 10$ ns, the error rate never falls from totally random decoding of 4 bits. On the other hand, with $T_1, T_2 = 1100, 1000$ ns, a significant throughput advantage is observed, outperforming the classical case with a relatively low incoming message rate of $r \approx 0.2$. For the $T_1, T_2 = 110, 100$ ns case, the classical rate is only surpassed at a high incoming message rate of $r \approx 0.9$, where entanglement will be stored for only short periods. The case when $L = 5J$, where $J = 4$, has the same properties of the previous case, except here the case when $T_1, T_2 = 110, 100$, the advantage over the purely classical approach is seen at a much lower incoming message rate of $r \approx 0.5$.

In conclusion, what we can say in this case is that already with two connections, the use of the GEWI communication strategy can already supersede a purely classical transmission approach even with low storage times of $T_1, T_2 = 110, 100$ ns for high rates of traffic, and with an order of magnitude larger, $T_1, T_2 = 1100, 1000$ ns, can observe super-classical rates even with low incoming message rates. Here the size of the buffer is less important as with two links and one source and the method of routing selected, there is more idle time on average between the two links in comparison to the single link case.

2.1.6. Two-Way Communication for Distributed Computing

For the single-link setting, an important and natural extension to consider is a two-way communication scenario, where there are idle periods between responses. Such scenarios in which two parties communicate between themselves with periods of idle between communication is a common setting in, for example, distributed computing [47]. In this setting, two parties send messages back and forth between themselves and once messages are received, there is a period of processing or idle until a response message is sent back. In this case, the processing or idle time is an opportunity that can be used to build up entanglement on the respective channels. When data is ready for transmission, the two parties can then make use of the established entanglement to accelerate their transmission.

In this section, we focus on a (classical) distributed computation setting and simulate a setting where two parties are performing the unsupervised learning algorithm k-means clustering [48] in a distributed fashion [49–51]. The initial condition of the distributed computation is that each party has access to the complete data set as well as knowledge of the initial centroid points. For the first iteration of the algorithm, each party computes the distance to the centroids for a fair share of the data, thereby labeling just their part of the data. The computed labels are then sent to the other party, each party receiving the labels from the other, to update the centroid locations. An alternative approach can also be taken, where with the partial label data, partial

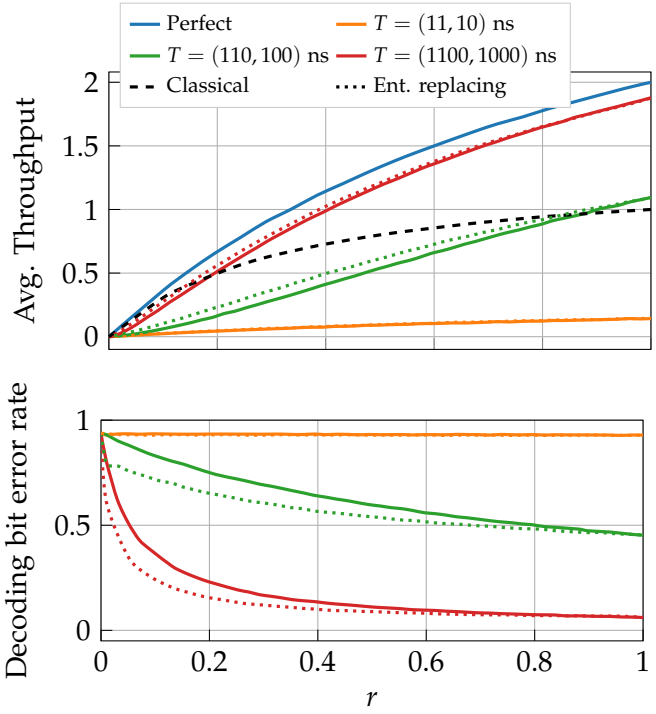


Figure 2.9.: A simulation of a BECN as in Fig. 2.8. In the simulation, the nodes are separated by a fiber 1 *km* long with a fixed delay of 5000 *ns*. Each node has an entanglement buffer of 200 memory slots which have specific T_1 and T_2 times associated with them. Here the source node has a buffer capacity for one message.

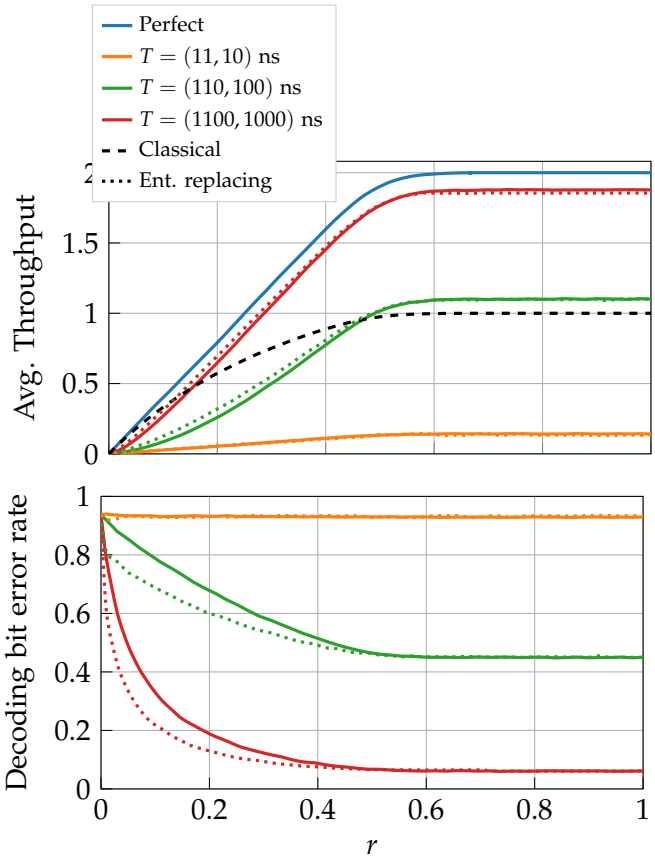


Figure 2.10.: A simulation of a BECN as in Fig. 2.8. In the simulation, the nodes are separated by a fibre 1 *km* long with a fixed delay of 5000 *ns*. Each node has an entanglement buffer of 200 memory slots which have specific T_1 and T_2 times associated with them. Here the source node has a buffer capacity for five messages.

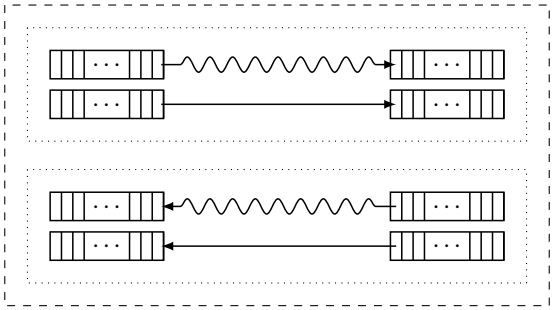


Figure 2.11.: A depiction of the communication setting we consider for a distributed computing experiment with entanglement generation. The two parties each have a one-way channel towards the other party such that two-way communication is achieved while both parties can generate entanglement during the computation phase of an algorithm.

average positions for the new centroid positions can be computed and these partial averages can be shared with the network to then update the centroid locations. This is possible since all parties know the number of data points each other party is averaging over ahead of time. Once the centroids are updated, the remaining unlabeled points can then be labeled. The difference here is that the precision of a real number comes into play and a reasonable choice for encoding such values becomes a trade-off to make. When noise is considered, given that, depending on the data, small changes to centroid locations can still produce accurate labeling, it might be the better option over label data transmission, where a mistake in the labeling would likely be more detrimental to the result. Indeed, when the error is in the more significant digits, the result could make the results worse, so ensuring the most significant digits are sent noise-free would be critical.

Here we only consider the first case as a proof-of-concept with binary labeling, thereby using superdense coding to transmit the label information at an accelerated rate. The idea to enhance the distributed computation with stored entanglement is that while the parties are computing their part of the distances between centroids and data points, entanglement is generated as a side process so that when the individual parties complete their processing, the respective data can be transmitted with fewer transmissions using an entanglement-assisted capacity. The simulation setting we use in this case is much like the single link simulations in Section 2.1.4, except we extend the channel slightly to accommodate two-way communication. We depict the scenario in Fig. 2.11.

Simulation Configuration and Execution

The simulation setting, in this case, is the following: The topology of the network is the one depicted in Fig. 2.11. The nodes can communicate back and forth over a quantum channel, where all messages are transmitted via qubit mediums. Each node can store 500 EPR pairs in a quantum memory that has decoherence properties based on a T_1, T_2 time model, as described in earlier sections. The memory we select here is one with

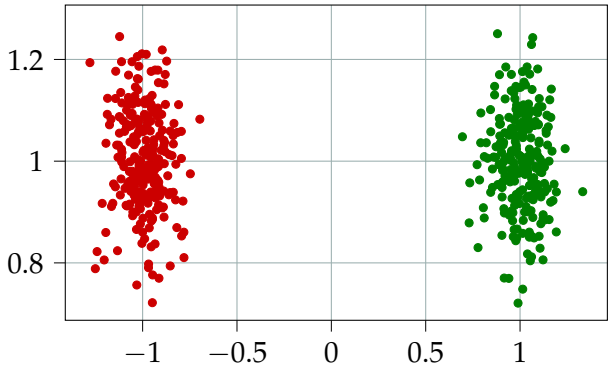


Figure 2.12.: The synthetic data set used for the distributed clustering simulation.

$T_1 = 1100$ ns and $T_2 = 1000$ ns. We repeat the process for more robust memories with $T_1 = T_2 = 1$ ms and $T_1 = T_2 = 2$ ms. The channel connecting the nodes in this case is reduced to 20 meters to simulate a data center distance, reducing the transmission latency in comparison to the previous section.

The data set we consider here is depicted in Fig. 2.12 which is simply a synthetic data set consisting of 500 data points surrounding 2 clusters clustered around $(-1, 0)$ and $(1, 0)$. This simple configuration is chosen so that here we can focus on the communication aspect of the distributed computation rather than an overly complex computation scenario. The data sets are generated with a standard deviation of 0.1 around the center.

The computation and communication scenario is the following: For a maximum of 10 iterations, the two nodes compute the distances for their fair share of the 500 data points (i.e. 250 points each). During this time, they transmit a varying number of EPR pairs between themselves. We assume that the time between the last EPR transmitted to the end of data processing is 1 ms, a duration selected based on the performance of a modern laptop performing one iteration of clustering on 250 data points. During this 1 ms, the qubits decohere in the memory. After the 1 ms, the two parties share their respective 250 labels with the other party so that new centroid locations can be computed. In this case, we are considering binary classification and so this is a binary string of 250 characters. Using the pre-shared entanglement, they transmit several messages to the other party using superdense coding as much as possible and then reverting to classical message transmission when not. To mitigate errors, after each iteration of clustering, the indexes of the data points labeled by each party in the previous iteration swap, and so each party labels the data points of the other in an alternating fashion.

The metrics for performance that we consider are the total number of message transmissions used during the execution of the protocol, and the F_1 score [52] of the classification comparing the final labels of Alice and Bob, which is defined as, for a collection of binary labels A and B,

$$F_1(A, B) := 2 \cdot \frac{precision_{AB} \cdot recall_{AB}}{precision_{AB} + recall_{AB}}. \tag{2.1.14}$$

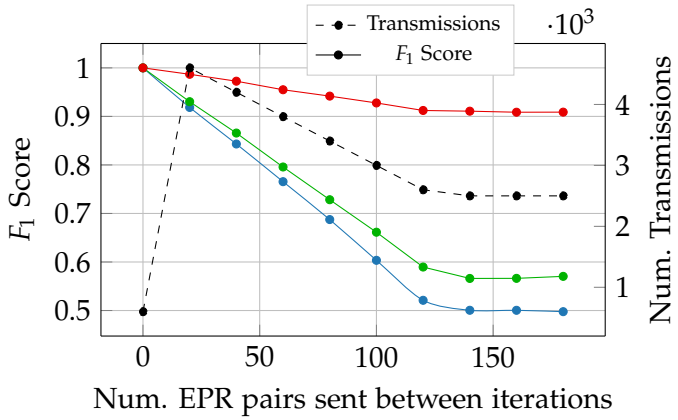


Figure 2.13.: The trends for a total number of transmissions made between two computers over a quantum channel versus the accuracy they can achieve for the binary classification. Note that the number of transmissions is scaled down by 3 orders of magnitude on the right Y-axis. The red (up most) plot represents $T_1 = T_2 = 10$ ms, the green (middle), $T_1 = T_2 = 1$ ms and the blue (lowermost), $T_1, T_2 = 1100, 1000$ ns.

We run each simulation 200 times over, plotting the average. The standard deviation in the F_1 score is $< 2\%$ in each case, and for clarity do not plot error bars in the figures.

Analysis and Discussion

In Fig. 2.13, we see the performance of this mixed computation and communication setting. What we observe is that with a $T_1, T_2 = 1100, 1000$ ns, the fidelity of the entanglement is not high enough. As more entanglement units are used, more noise is introduced to the messages, making the decoding process nearly random, and eventually so much so, that the labels of Alice and the labels of Bob become completely random at the end of the 10 iterations. As one could expect, as the memory fidelity increases, the F_1 score improves as well, implying better matching between sender and receiver. Because superdense coding is being used to communicate, fewer and fewer transmissions are made, but at the cost of a reduced F_1 score. When the T_1, T_2 time approaches 1 ms, the best trade-off is seen, where there is a high F_1, ≥ 0.9, score and fewer total transmissions used.

2.1.7. Summary

In conclusion, we investigated a communication scenario where, to boost communication rates, entanglement is generated during times when communication is idle, to be used when communication is required. We proposed a communication scheme that implements this, and modeled it after a stochastic process, reviewing the theory surrounding the model. Next, to investigate the physical hardware properties required to implement the communication strategy, we considered three communication settings: a point-to-point channel, a network of four nodes with a single source and a single

receiver, and a two-party computation. To analyze the three settings, we implemented simulations that simulate physical models of the quantum memory to determine various performance metrics for imperfect memory coherence times. What our results show is that even with near-term quantum memory coherence properties, using our communication model, it is possible to outperform purely classical settings, where no entanglement is used. Moreover, by varying the entanglement consumption queuing practice and replacing entanglement resources as often as possible with fresh pairs, even more, efficiency is obtainable.

2.2. A Link-Layer Protocol for Quantum-Enhanced Classical Networks

Section based on the article: "Integrating Quantum Simulation for Quantum-Enhanced Classical Network Emulation"

Future communication networks will encounter performance boundaries that will be difficult to overcome using traditional approaches. Significant trade-offs will make requirements for very-high data rates (at very-low latency with extreme reliability and resilience) very difficult. At the same time, networks need to continue implementing complete network softwarization and massive data mining and processing because of in-network Artificial Intelligence (AI) [53], for example. To go beyond these technological limitations, new communication paradigms are needed. Recent advances in entanglement-assisted data transmission [5] highlight the potential of quantum communication methods for hybrid networks, utilizing these novel methods at lower network layers: this advantage being simplified by the integration with the existing infrastructure [38]. The question arises whether this concept of hybrid classical-quantum networks carries further than described previously [38, 54, 55]. As the answer to this question is of a highly interdisciplinary nature, we highlight a full integration of the quantum network simulator QuNetSim [56] into the link layer of the classical network emulator ComNetsEmu [53]. With this tool, a variety of networking questions can be answered using the large set of possible quantum communication techniques [57] as part of hybrid classical-quantum communication networks.

We display the potential use of the software integration with a clear focus on the case of a hybrid entanglement-assisted quantum-classical communication network. We assume that the protocol stack above the link layer remains unchanged. The link and the physical layer, however, are modified to enable quantum information processing and communication. In addition to the integration, we propose a novel link-layer protocol, which implements the queuing model in [38], where entanglement-assisted data transmission is a stochastic process generating, sharing, and storing entanglement between network nodes during stagnant communication periods, enabling higher data rates. If the stored entanglement is depleted during data transmissions, any further message in the buffer of the sending node is transmitted using classical communications. This protocol is labeled “Generate Entanglement While Idle” (GEWI, see [38]), and we use it as a proof-of-concept for our emulation tool. Finally, the protocol is evaluated

over a single network link using the integrated software platform. In this way, we close an existing gap between quantum network simulation and classical network emulation.

2.2.1. Link Layer Protocols for Quantum Networks

The following briefly reviews existing link-layer protocols for quantum networks and explains some important concepts for understanding the contribution of this article. Here the link layer is considered a mechanism providing error-free qubit transmission between the interfaces connecting the network nodes. A network not making use of quantum effects will encode one bit per qubit at those interfaces. A network utilizing quantum effects can, in addition to this, distribute, store, and measure Einstein-Podolsky-Rosen (EPR) pairs between any two connected interfaces and execute arbitrary gates on each interface. A complete overview of this process is found in [57, Chapter 6.4]. This encapsulation of quantum functionalities into the lower layers simplifies the integration in terms of software and hardware, and is also in line with recent research, which we review in the following.

In [54], a classical-quantum network layer is considered, which can increase network throughput using entanglement-assisted message transmission. Features for a link-layer protocol are also proposed, which aim to generate entanglement for subsequent use in entanglement-assisted message transmission. Missing from [54] is the explicit protocol used on the link layer to transmit classical messages using the quantum channel. It only considers the statistical properties of the link layer to measure the performance of the network.

Pirker and Dür propose a protocol stack for a quantum network [16]. They introduce a so-called "connectivity layer" between the quantum physical and link layer. Such a layer does not have an analog in the Open Systems Interconnection (OSI) model [15]. The connectivity layer handles the requests from the upper layers by converting them into instructions for the physical layer. Such instructions can, for example, be a request for qubit transmission or EPR pair generation, without assumptions of the underlying physical models. Moreover, the connectivity layer provides means of handling quantum errors arising at the physical layer, detaching the link-layer logic from the physical layer. The proposed link layer further keeps track of the current quantum entanglement status in the network to later generate long-distance quantum entanglement. The connectivity layer hides the particular implementation of the physical layer from the higher layers. Our exemplary implementation of a network link, utilizing superdense coding, rests on the assumption of error-free transmission of (entangled) qubits, a functionality that can be achieved using functionalities of a connectivity layer.

In [19], a quantum network link-layer protocol is proposed for the generation and distribution of entanglement between network nodes. The main focus of the work is establishing multi-party entanglement in quantum networks rather than a mixture of entanglement generation and classical communication. The duty of the proposed link layer is to schedule which requests for entanglement will be served first. In the work, this is left open to the specific application and so one can consider the GEWI protocol as an explicit schedule to the proposed protocol. Moreover, since here we use the quantum channel to transmit classical information as well as entanglement, one

could make use of the "Create and Keep" protocol to store entanglement to perform entanglement-assisted communication.

Among these works, only [54] explicitly considers enhancing communication rates in classical networks with entanglement, but an explicit protocol is not considered. The use of entanglement in the other works is primarily for entanglement-swapping routines or teleportation of quantum states. Using entanglement primarily for enhancing classical communication rates, and generating entanglement only when the network is idle, allows the network to see performance improvements when entanglement is established, and still function normally otherwise. We consider both the explicit link-layer protocol as well as a simulation tool for determining these performance improvements.

2.2.2. Proof-of-Concept: A Link-Layer Protocol for Classical-Quantum Communication Networks

In this section, we describe a link-layer protocol to perform entanglement-assisted transmissions which we will then use with the classical-quantum simulation platform described in the previous section for proof-of-concept. The protocols referred to are defined below. Protocol 1 contains the heart of the logic. In each iteration of Protocol 1, the sender checks if there is classical data to send at the upper layers. If there is data, then a quantum data frame is generated and transmitted using Algorithm 1, which uses the state of the entanglement buffer to determine if the classical data frame should be encoded into qubits (thus encoded into an entanglement-assisted data frame) or not, using the stored EPR pairs in a First-In-First-Out (FIFO) ordering. On the other hand, if there is no data to transmit, a frame of EPR qubits is sent if the quantum storage can accommodate it.

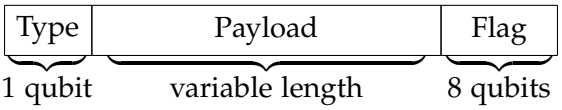


Figure 2.14.: The link-layer data frame used for the entanglement-assisted quantum channel.

The data frame structure we use is depicted in Fig. 2.14. One leading qubit is used to indicate the *Type* of payload—entanglement or data—followed by the payload itself, which is of variable length. The benefit of this is the avoidance of padding, which could result in a waste of quantum resources. Next, a reserved byte sequence *Flag*, representing the "end of frame", signals the end of the frame. Alternatively, we could have included the number of bits transmitted in the frame, but when this binary number is greater than 255 bits, it becomes less efficient than using a termination flag. When an EPR frame is sent, the protocol uses a maximum size of the EPR payload L. In this noiseless and ideal setting, we exclude a leading flag for the data frame, since qubits here are a discrete resource, for example, a single polarized photon. After the end of a sequence, any qubit detection would indicate the beginning of a new incoming frame. In more realistic scenarios, the data frame would need to be modified according to the hardware parameters.

Once a data frame is received, the receiver performs Algorithm 3. The receiver simply decodes the first incoming qubit to determine the payload type. If the header indicates EPR pairs, they store the payload for later use. If the header indicates data, then the information is decoded either with entanglement assistance or without. The sender and receiver agree with this before Protocol 1 starts.

Protocol 1 Entanglement-Assisted Message Transmission

Sender

1: **while** *active* **do**
2: **if** *there is data to transmit* **then** *send data frame*
3: **else if** *entanglement buffer is not full* **then** *send EPR frame*

Receiver

1: **while** *active* **do** *receive frames*

Algorithm 1 Send Data Frame

1: $payloadQubits \leftarrow$ *initialize empty list*
2: $q_h \leftarrow$ *initialize qubit in the* $|1\rangle$ *state*
3: **for** byte **in** data stream **do**
4: **for** (b_1, b_2) **in** byte **do** ▷ iterate 2 bits at a time
5: **if** *ent. buffer non-empty* **then**
6: $q_1 \leftarrow$ *pop qubit from ent. buffer*
7: *superdense encode* q_1 with (b_1, b_2)
8: *add* q_1 *to* $payloadQubits$
9: **else**
10: $q_1, q_2 \leftarrow$ *initialize two qubits in* $|0\rangle$ *state*
11: **if** $b_1 = 1$ **then** $X(q_1)$ ▷ excite q_1 to $|1\rangle$
12: **if** $b_2 = 1$ **then** $X(q_2)$ ▷ excite q_2 to $|1\rangle$
13: *add* q_1, q_2 *to* $payloadQubits$
14: $flagQubits \leftarrow$ *generate 8 qubit flag state*
15: $frame \leftarrow (q_h, payloadQubits, flagQubits)$
16: *send frame*

In Chapter 4 of this thesis, we demonstrate an integration of this protocol into a classical-quantum network simulation merger. We combine the network simulators Mininet and QuNetSim and analyze the performance of this protocol in depth. The results show that this link layer can offer a performance improvement for classical communication over qubit channels.

2.3. Network-Layer Protocols for Entanglement Redistribution

Section based on the article: "Entanglement-Enhanced Communication Networks"

An entanglement-enhanced communication network is a network where, depending on its phase of development, the nodes in the network can generate, store, and also swap entanglement. Thereby network nodes can share growing amounts of entangled

Algorithm 2 Send EPR Frame

payloadQubits ← *initialize empty list*
q_h ← *initialize qubit in the* $|0\rangle$ *state*
while ent. buffer not full **and** $|payloadQubits| < L$ **do**
  q_1, q_2 ← *initialize two qubits in* $|0\rangle$ *state*
  entangle q_1, q_2
  store q_1 in ent. buffer
  add q_2 to *payloadQubits*
$flagQubits$ ← *generate 8 qubit flag state*
$frame \leftarrow (q_h, payloadQubits, flagQubits)$
send frame

Algorithm 3 Receive Frame

m_h ← *receive header qubit and measure*
bytes ← *initialize empty list*
bits ← *initialize empty list*
if $m_h = 1$ **then** ▷ Data frame
  while $bits \neq FLAG$ **do**
    reset bits
    while $|bits| \neq 8$ **do**
      q ← *receive qubit*
      if *ent. buffer non-empty* **then**
        q_e ← *pop qubit from ent. buffer*
        b_0, b_1 ← *superdense decode* (q, q_e)
        add (b_0, b_1) to *bits*
      else ▷ classical decode
        $b_0 \leftarrow measure(q)$
        $bits.add(b_0)$
    add *bits* to *bytes*
else ▷ EPR frame
  qubits ← *receive L qubits*
  add qubits to ent. buff

states over distances that increase as the technology matures. This enables the execution of novel network protocols that initially accelerate only the already known tasks of data transmission, while at the later stages of the network, development allowing for execution of novel tasks and ultimately being able to network quantum computers. With this, we explore a phased approach to how such a network can be developed.

Phase One: In phase one, the aim is to develop the physical layer and the link layer so that shared entanglement allows to temporarily increase the throughput of a point-to-point link. With this, only the capacity of each link can benefit from newly developed quantum technology for entanglement generation and transmission, allowing the use of the entanglement-assisted capacity of the link [36]. Recent results show that potential gains for entanglement as a plug-in resource to classical communication can by far exceed the factor 2 demonstrated in superdense coding [58]. Preliminary analysis of such an advantage is analytically demonstrated in [5, 59–61]. Experiments

of superdense coding demonstrating an entanglement-assisted classical capacity of an optical fiber have been achieved in [62]. Entanglement across each link is created by the link itself during periods when no or small amounts of data need to be transmitted, and therefore no information to higher layers is required in this phase. We detail the effects on the affected layers.

Physical Layer: To generate entanglement, the physical layer must produce entangled photons to distribute from one node to another. Hardware additions that can generate heralded entanglement between two parties will be needed. We enforce that entanglement pairs are generated on one side of a link and then half of the entanglement is transmitted to a second party, where a protocol (see section 2.3.1) would be put in place such that using classical communication, the arriving qubits would be known to be an entanglement unit. Further, quantum memories used for storing quantum entanglement between connections will also be needed. The memory hardware can vary node to node, but the main principle is that each memory can store entanglement in some way.

Link Layer: The link layer determines how information is encoded to transmit it over the link. If there are entanglement resources available, the link layer can choose to encode information such that the entanglement-assisted capacity of the channel [36] is used. Alternatively, if there are no entanglement resources, links can transmit purely classically. Optical fibers connecting routers will enable the transmission of information either based on logical qubits or based on logical bits, depending on the state of the entanglement resource buffer. To accomplish such a task, for example, one can use a software-based switching mechanism as in [63]. We place the entanglement generation logic on this layer since no entanglement routing is needed and the network in the phase is oblivious as to how the information is transmitted. Reliable entanglement generation is thus also left to the link layer.

Network layer: In this phase, we can begin to consider routing algorithms that take into consideration the potential entanglement resources along a path. Using link-state routing methods, using routes that have more entanglement can be considered for single- or multi-path routing.

Phase Two: In phase two, we focus on the network layer. Here the key is to develop network protocols so nodes can swap entanglement amongst themselves such that links under heavy load can benefit from entanglement that was generated on other links of the network, also increasing entanglement utilization. Link cost calculation methods can also reflect the potential to swap entanglement if swapping can be done before transmission. Routes can then be calculated based on various link cost metrics and using for example a max-flow-min-cost approach. Phase two can also be considered as a series of sub-phases. We use the term "physical link" to mean a direct connection between two nodes (e.g. a cable or air interface), and "virtual link" for a connection that is established via intermediate nodes. Then, the "swap distance" of a network can be defined as the maximum number of intermediate nodes in a virtual link of the network that has entanglement shared between its endpoints. For each proceeding phase, the swap distance is increased by one. We provide simulation results in this thesis for the initial phase, where the swap distance is one, and for a phase where the swap distance is unbounded. The sub-phases are highly correlated with the advancement of quantum

repeater chains and the generations of quantum repeaters such as defined in [18].

Network Layer: The network layer will need to coordinate entanglement swapping amongst the nodes to make use of new routing approaches and to add entanglement resources to routes that are under heavy load. New network protocols to reliably accomplish such tasks will be needed. Depending on the sub-phase of phase two, the level of coordination of these protocols will increase. While the swap distance increases, we can envision novel research topics regarding routing and applications for entanglement-enhanced networks.

Phase Three: Up to now, we have considered methods for transmitting only classical information through the network, which is encoded classically or quantumly depending on the state of the entanglement in the network. In this final phase, we integrate the transmission of quantum information, that is, the possibility to transmit arbitrary quantum states. Developing an internet that transmits quantum information is currently receiving much attention (see for example [64]). This phase will require joining the two paths to form a unified classical-quantum Internet.

2.3.1. Entanglement Generation and Balancing in the Network

Through all phases, our assumption in this thesis is that entanglement is generated independently between direct endpoints of physical links during idle times, as described in [59]. That means we explicitly exclude the possibility of forwarding or relaying qubits in the entanglement generation process. Thus, we avoid contradiction with the end-to-end principle, stay in tune with the current network architecture, and open a path of subsequent evolutionary optimization towards a fully quantum internet. In this section, we propose a “sliding-window"-like method for distributing entanglement across a physical link. We emphasize that, from a network perspective, the entanglement is considered generated only after it is distributed across a link. Further, we describe an entanglement balancing protocol as an optimization step.

Entanglement Generation and Distribution

In phase one, we consider a system that is continuously attempting to generate entanglement between nodes when they are idle. This generation mechanism is autonomous and unknown from the perspective of the network layer which is in alignment with the end-to-end principle [65]. The generation of entanglement is therefore a link-layer protocol.

We rely on the ability to store quantum entanglement in quantum memories for these protocols. Using a heralded entanglement generation technique, for example using the technique described in [66], individual entanglement units are generated locally to a minimum entanglement fidelity threshold F_{min}. With these individual units, a frame of entanglement is generated where a “frame" in this setting is a collection of entanglement units. We propose a protocol for efficiently distributing entanglement frames between two hosts connected by a quantum link. We base the protocol on the TCP sliding window protocol.

The protocol runs as follows. A frame of EPR pair halves is generated with a minimum fidelity F_{min} and is transmitted. The qubits are sent one after another to the

receiver and at the last qubit in the frame, a timer starts at the sender. As with a sliding window algorithm, each frame is given a sequence number. After the frame is sent, the sender awaits an acknowledgment from the receiver for a fixed amount of time. If the acknowledgment is received within the timeframe, then the EPR pair halves are stored at the sender. If no acknowledgment is received, the sender discards their halves. At the receiver, if the frame meets the acceptance criteria, the pair halves are stored, and an acknowledgment is sent. The procedure of preparing and accepting frames is given in Protocol 2 and a depiction of the procedure is in Fig. 2.15.

In Protocol 2, we refer to an **await** task that is waiting for N qubits to arrive. This is dependent on the method of transmitting physical qubits. Presently, there are methods for non-destructive detection of photons, even demonstrated experimentally in [67], but the detection method does not in general maintain quantum correlations. Future generations of quantum optics technology promise improvements to non-destructive detection methods of photons [68, 69].

The link layer is further responsible for maintaining entanglement units with high enough fidelity. Based on the properties of the quantum memory device, entanglement units in storage will be automatically eliminated from the system based on the expected coherence time of the memory devices, or they can be consumed by performing a round of entanglement distillation to generate higher quality entanglement [70].

Protocol 2 Sliding window entanglement generation protocol

Input: A previously agreed upon entanglement fidelity $F_{\min}$ and values N, total number of entangled pairs and T the number of qubits for foreign fidelity testing. We assume that a handshake protocol to initiate entanglement frames transfer has been established when the data transmission buffer at the sender's side is empty. Moreover, when a frame is resent, the quantum memory holding the qubits in that frame at the sender is first cleared.

Sender

1: Prepare N entangled pairs using (for example) heralded entanglement generation with fidelity $\geq F_{\min}$
2: *S_Measurements* $\leftarrow$ Measure T halves of the N entangled pairs
3: Send the remaining N halves to Bob in a frame with *S_Measurements*
4: Store remaining $N - T$ halves and await

Receiver

1: **await** Receive N entanglement halves from sender
2: *Timer* $\leftarrow$ On the arrival of the first qubit, start a timer to track frame transmission time
3: **if** number of received $< N$ after a waiting period **then**
4: **reject** frame
5: *R_Measurements* $\leftarrow$ Measure the agreed subset of T qubits received
6: **if** ratio of matches in *R_Measurements* and *S_Measurements* $< F_{\min}$ **then**
7: **reject** frame
8: Store remaining $N - T$ halves and **accept** frame

Entanglement Buffer Balancing

As an optimization step, a protocol for balancing the entanglement buffers can be used. We assume each transmission link is directed and has one entanglement buffer

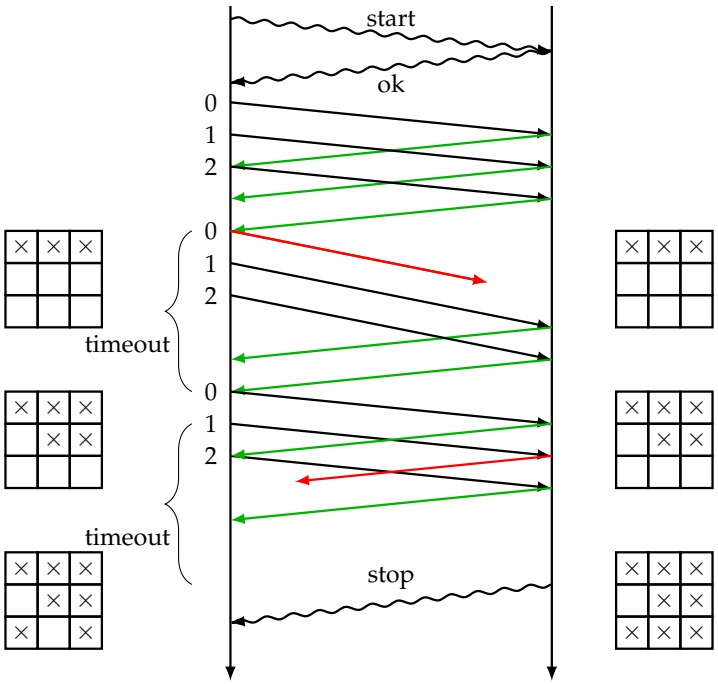


Figure 2.15.: In the first three frame transmissions, all three frames are acknowledged, and they are stored successfully at the sender and receiver. In the second window, the first frame transmission does not arrive at the receiver. The receiver does not add the frame to memory and as a consequence neither does the sender. In the next window, the receiver receives all three frames, but an acknowledgment for frame 1 to the sender is lost. The sender discards their half of the EPR pair after a timeout period and continues the process, but the receiver holds their half of the EPR pair. The qubit in the receiver memory will eventually expire and the sender will not use that frame for transmission. Finally, the receiver sends a signal to the sender to stop sending entanglement frames.

consisting of two quantum memories at its endpoints (see Fig. 2.16). Let a link $A \to B$ be accompanied by its reverse link $B \to A$. If A transmits data to B, but B does not transmit data to A at the same time, entanglement units are created on the $B \to A$ link. While A and B consume the shared entanglement stored in the $A \to B$ entanglement buffer, a messaging protocol between A and B can "relabel" entanglement units from the entanglement buffer of the $B \to A$ link to that of the $A \to B$ link. The protocol is given in full in Protocol 3.

Protocol 3 Entanglement Balancing

Input: E_{thresh}, E_{min}

Sender

1: *EntBufferLow* ← Number of available entanglement units on $A \to B$ link is less than E_{min}
2: *EntBufferHigh* ← Number of available entanglement units on $B \to A$ link is more than E_{thresh}
3: **if** not (*EntBufferLow* & *EntBufferHigh*) **then**
4: **continue**
5: *EntUnits* ← select the N newest entanglement frames in the $B \to A$ entanglement buffer
6: **send** classical request to B to make the *EntUnits* entanglement frames available for $A \to B$ transmission
7: **await** *Avail* ← List of available frames from B
8: **if** *Avail* is non-empty **then**
9: Use new *Avail* entanglement frames for transmission
10: **else**
11: **continue**

Receiver

1: **receive** request from A to re-address the *EntUnits* entanglement frames from $B \to A$ memory to $A \to B$ memory
2: **for each** e in *EntUnits* **do**
3: *success* ← mark e as an $A \to B$ entanglement frame, return success status
4: **if** *success* **then**
5: add address e to list *Avail*
6: **send** *Avail*

Supporting Routes via Nearby Entanglement Swapping

Entanglement in the network can be a valuable resource, and in some cases, it can be beneficial to move entanglement around in the network to high-demand routes rather than waiting to generate more entanglement. We motivate with an example.

If large amounts of data need to be transmitted in a short time, entanglement swapping can be beneficial. Consider the exemplary situation in Fig. 2.17 with $t = 5$, $x = 3$, and $y = 4$. Assume the number of EPR pairs stored per link is 3. The act of sending 12 bits from s to d using swapping can take 6 time units: In the first 3

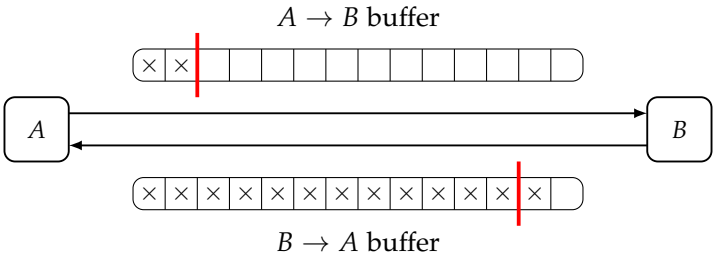


Figure 2.16.: The upper entanglement buffer stores entanglement created on the $A \to B$ link and the lower that created on the $B \to A$ link. Each buffer is physically shared between A and B. Red lines represent example thresholds where balancing can be triggered.

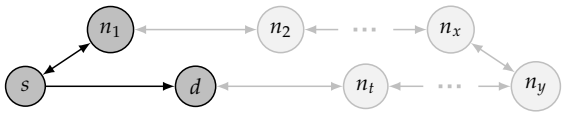


Figure 2.17.: Example of a growing swap chain.

time units all entanglement on (s, d) is consumed while transmitting the first 4 bits. Meanwhile, the entanglement on $(s, n_1) \dots (n_6, d)$ is swapped to (s, d) via a chain of measurements and classical messaging. The first swapped EPR pair is available in the 3rd communication round when the entanglement originally present on (s, d) has just expired. Thus in each round 2 bits are sent. Sharing the load fairly over (s, d) and $(s, n_1), \dots, (n_t, d)$ will take 8 time units, using only (s, d) without swapping will take 9 time units, using no entanglement but all available paths will take 9 time units as well.

Once a route or a collection of routes is established between source and destination, we can consider supporting the route with additional entanglement resources. We consider an approach to orchestrate the nodes which are not part of the route, that share entanglement with those which are, to supply it with entanglement resources. We provide a sample algorithm to do this in Algorithm 4.

Algorithm 4 Entanglement Supplementation

Input:
R: Route steps from s to d (i.e.$[(s, l_1), (l_1, l_2), \dots, (l_n, d)]$)
K: List of candidate supplier nodes

```
for each (l_1, l_2) in R do
  for each k in K do
    if not(connected(l_1, k) and connected(k, l_2)) then
      continue
    if entUnits(l_1, k) = 0 or entUnits(k, l_2) = 0 then
      continue
    swap(k, l_1, l_2)
    updateLinkInfo(k, l_1, l_2)
```

We can instead use a linear program to find the optimal swaps to make to provide a given route with sufficient amounts of external entanglement. As we have Algorithm 4,

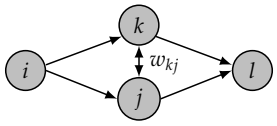


Figure 2.18.: Node k has the ability to perform an entanglement swap to the edge (i, j) and (j, l) if the necessary entanglement has been pre-established. In this case, the amount swapped to (i, j) or (j, k) must be less than the weight w_{kj}.

there is a list of candidate supplier nodes K where each $k \in K$ has edges to nodes i and j where (i, j) is an edge in the route. In this direction, let $x_{kij} \in \mathbb{N}$ represent the amount of entanglement swapped from node k to the link (i, j), where the edge (i, j) is an edge in the route. $\mathcal{S}$ is the set of all possible swaps in this form. $\mathcal{E}(R)$ is the amount of entanglement that exists on the route R and $\mathcal{E}$ the total entanglement in the network. $\mathcal{C}$ is the set of swap triples $((k, i, j), (k, j, l))$ where the node k can swap to the edges (i, j) or (j, l) where j is a common node in the swap (see Fig. 2.18 for a depiction). Finally, M is the number of bits to transmit. The linear program is given in Linear Program 1.

Linear Program 1 Local Entanglement Redistribution

Maximize: $\sum_{(k,i,j)\in\mathcal{S}} x_{kij}$.

Subject to conditions:

1. $\sum_{(k,i,j)\in\mathcal{S}} x_{kij} \leq \mathcal{E} - \mathcal{E}(R)$,
2. $0 \leq x_{kij} \leq \min\{w_{ki}, w_{kj}, \lceil M/2 \rceil\}, \quad \forall (k, i, j) \in \mathcal{S}$,
3. $x_{kij} + x_{kjl} \leq w_{kj}, \quad \forall ((k, i, j), (k, j, l)) \in \mathcal{C}$.

Supporting Routes via Global Entanglement Swapping

We define a linear program allocating entanglement to a given route from the entire unused part of the network. For a collection of edges $L \subset E$, we write $G \setminus L$ to denote $(V, E \setminus L)$. A path p_{uv} is an ordered collection of edges from vertex u to v. Let p_{sd} be a path connecting the source s and destination d in the graph G. Define $G' := G \setminus p_{sd}$. Then G' is the part of G that can contribute entanglement to support message transmission via p_{sd}. Let for each edge $(u, v) \in G'$, $P(u, v)$ be the set of all paths in G' that connect u to v. For each step $(u, v) \in p_{sd}$, $P(u, v)$ is the set of paths that can swap entanglement towards (uv). The length of $q \in P(u, v)$ is denoted $|q|$.
Optimization variables: $x^{uv}_{ijq} \in \mathbb{N}$ where $(i, j) \in G'$, $(u, v) \in p_{sd}$ and $q \in P(u, v)$. Each x^{uv}_{ijq} is the amount of entanglement taken from link (i, j) via path q to edge (u, v). $(w_{ij})_{(i,j)\in G'}$ is the entanglement in G'.

Linear Program 2 Global Entanglement Redistribution

Maximize: $x \in \mathbb{N}$

Subject to conditions:

1. Cannot swap more than minimum amount per path.

$$0 \leq x_{ijq}^{uv} \leq \min_{(k,l)\in q} w_{kl}, \quad \forall (i,j) \in G', \forall (u,v) \in p_{sd}, \forall q \in P(u,v). \tag{2.3.1}$$

2. Cannot overuse entanglement on the link,

$$\sum_{(u,v)\in p_{sd}} \sum_{q\in P(u,v)} x_{ijq}^{uv} \leq w_{ij}, \quad \forall (i,j) \in G'. \tag{2.3.2}$$

3. All edges use the same amount of entanglement,

$$x_{ijq}^{uv} = x_{klq}^{uv}, \quad \forall (u,v) \in p_{sd}, \forall q \in P(u,v), \forall (i,j),(k,l) \in q. \tag{2.3.3}$$

4. To max-minimize the bottle neck on each path,

$$w_{uv} + \sum_{q\in P(u,v)} \sum_{(i,j)\in G'} x_{ijq}^{uv} \geq x, \quad \forall (u,v) \in p_{sd}. \tag{2.3.4}$$

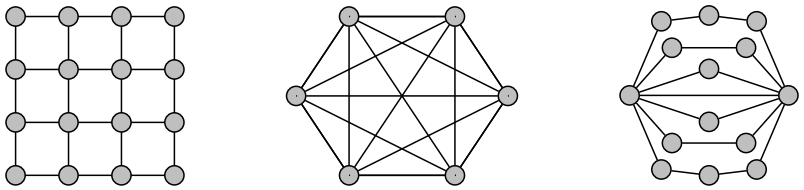
5. Should not waste entanglement, $\forall (u,v) \in p_{sd}$,

$$\sum_{q\in P(u,v)} \sum_{(i,j)\in G'} \frac{x_{ijq}^{uv}}{|q|} \leq \max\left(0, \lceil M/2 \rceil - w_{uv}\right). \tag{2.3.5}$$

The linear program is written in Linear Program 2. With the linear program solved, for each path q, the value x_{ijq}^{uv} is the amount of entanglement that can be moved to the link (u,v) and it is the amount of entanglement removed from reach edge (i,j) in the path q. In this initial work, we leave open the question of how to optimally execute the messaging that is needed to perform the optimal allocation as calculated in Linear Program 2. In simulations, we work with the simplified assumption that the delay until the entanglement is present at (u,v) is $\min\{\lceil \frac{|q|}{2} \rceil : x_{ijq}^{uv} > 0\}$ and links contributing an amount E of EPR pairs cannot build entanglement for E time units.

Simulation Results

To compare the approaches to routing and path augmentation discussed in the previous sections, we simulate two entanglement-enhanced network configurations depicted in Fig. 2.19. We compare the results to the classical setting using single- and multi-routing. For the analysis, we use graph $G = (V, E, W)$ defined in the previous section. We assume that there are several messages N that are being transmitted from a source to a



(a) Grid network (b) Complete network (c) Wide network

Figure 2.19.: Networks used for simulation analysis. In the grid network, transmissions are alternated from bottom-left to top-right corner and bottom-right to top-left. In the complete network, the source and destination are alternated clockwise along the outer edges. Source and destination are fixed for the wide network, which is the left and right nodes in the center row respectively.

destination. Each message has 6 bits to transmit. The simulation runs by selecting a source and destination and transmits a message completely. Except in the wide network case, once the message is transmitted, the source and destination change, and another message is transmitted until all N messages are transmitted. For the wide network, to incorporate noise, we assume that with a 50% chance an entanglement unit is generated on an unused link, and it costs 2 e-bits to send a superdense message. For the other network types, at each time step, an e-bit is generated with certainty on each unused link and the cost to send a superdense message is one e-bit. In the case of superdense coding, when entanglement exists over a link, the classical capacity is doubled. We use this knowledge to determine the capacity of the links before routing. An entanglement swapping procedure is assumed not to increase routing time. We show the results in Fig. 2.20.

2.4. Entanglement-Assisted Cooperation in MIMO Channel Settings

Section based on the article: "Undoing Causal Effects of a Causal Broadcast Channel with Cooperating Receivers using Entanglement Resource"

The use of entanglement resources in classical communication scenarios has been shown to reduce communication resources [71], increase the capacity of channels [72], and enable communication [38, 73]. Entanglement can moreover be used to improve upon protocols for secure multi-party computation and communication [74, 75], allowing parties to compute a common function securely. In this thesis, we introduce a communication scenario with a single sender broadcasting classically encoded messages to N receivers. To decode the messages, the receivers must cooperate, or conference [76], with one another or otherwise, because of the channel's causal properties, the capacity of the channel vanishes. This scenario mimics one where a sender broadcasts a message to a collection of receivers such that either all receivers receive the message or none of them receive it, based on their cooperation.

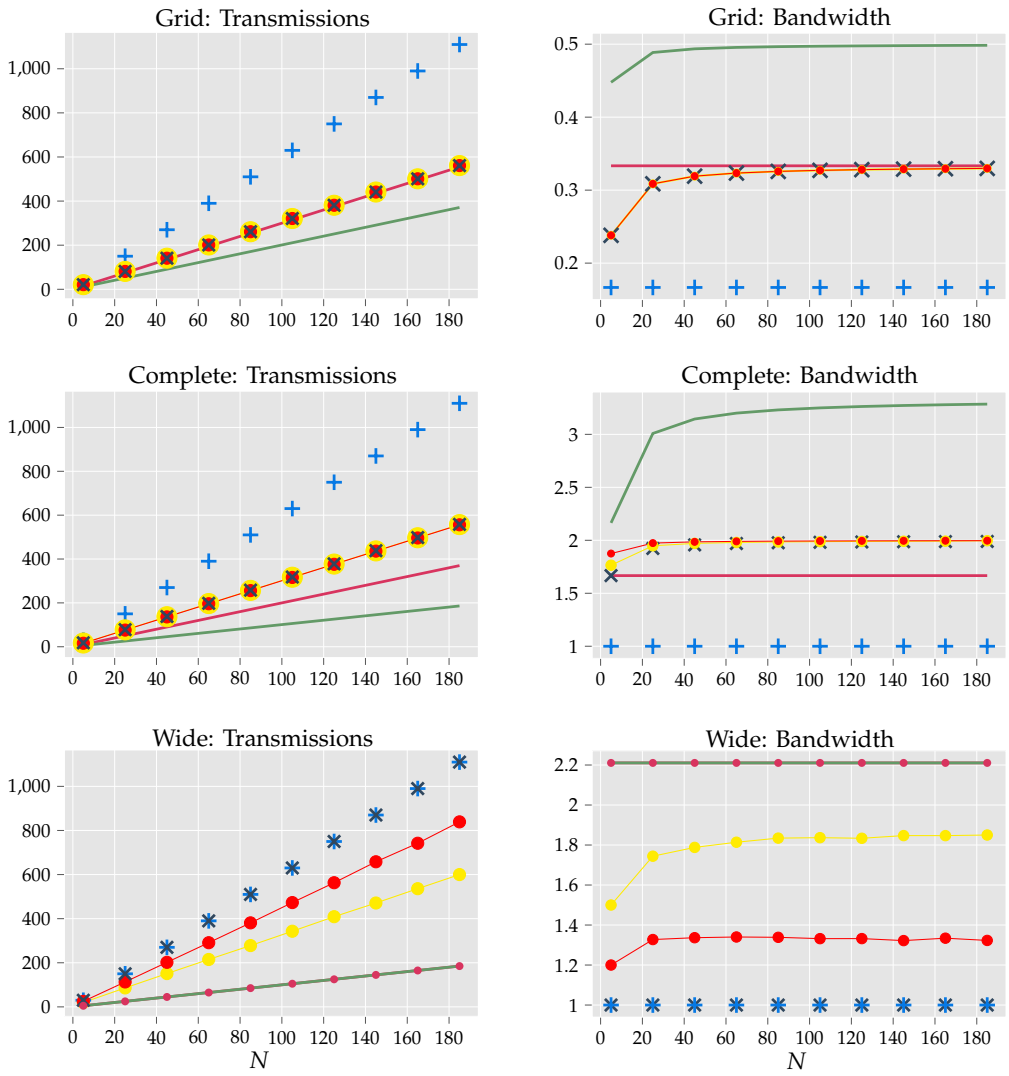


Figure 2.20.: The green lines represent max-flow routing with quantum resources and the red lines represent max-flow routing without quantum resources. For the shortest-path, the blue crosses are no quantum resources, the black crosses are with quantum resources without entanglement swapping, the red dots are with quantum resources and depth one entanglement swapping, and the yellow dots are with quantum resources and global swapping. Sub-figures in the left column represent the number of transmissions from the senders. Sub-figures in the right column represent the average network bandwidth, where each link in the network has one unit time for transmission.

The channel model in this thesis has a causal state variable that determines how the channel transmits the sender's messages. Specifically, "causal" here means that with each transmission made over the channel, the transmission behavior of the channel changes depending on a random state variable. When a message arrives at the receiver, it is accompanied by a piece of state information. The state information is split between the receivers such that alone, the best the receiver can do is randomly guess the channel state. The receivers' task is to cooperate to fairly determine the channel state – essential for decoding the sender's message. In the general case, the receivers can collude with each other in groups to attempt to determine the channel state unfairly, that is, where the receivers in the colluding group determine the channel state and the others do not. To overcome this, we apply secure multi-party protocols that work under a maximum colluding number of receivers. We investigate protocols under three conferencing resource scenarios.

The scenarios we consider are: when the receivers are only able to share entanglement, but are unable to classically communicate; second, they can classically communicate but not share entanglement; and lastly when they can both share entanglement and can classically communicate. In these scenarios, we determine a lower bound for the communication cost for performing protocols that allow the receivers to fairly determine the channel state. We find that with only classical resources, the receivers can perform a fair and conditionally secure protocol to determine the channel state but at the expense of additional classical resources compared to the case where they can in addition share entanglement. We see an overall quadratic reduction of the communication resources needed when entanglement resources are available with an addition of provable security.

2.4.1. Related Work

In this thesis, we consider a causal broadcast channel communication scenario with cooperating receivers. In [73], a dual scenario is considered where instead of a cause broadcast channel, the channel is a causal multiple access channel with cooperating senders. In that case, the receivers need not communicate, as shared entanglement is enough to achieve a positive channel capacity. In [77], a causal broadcast channel with channel state information at the sender is analyzed. In this thesis, we make use of the multi-party modulo summation protocols developed in [78] and [79], extending and optimizing them for this particular communication scenario when n modulo sums are needed.

We propose a communication model with a sender and many receivers that can be used to ensure that either all receivers receive their message with certainty if they cooperate or otherwise none of them receive their message. We use protocols developed in [78, 79] to develop a scheme that is fair and secure such that all receivers can decode messages from the sender fairly when the protocols run honestly, even when some receivers collude. We consider the trade-offs for allowing the receivers to share classical and quantum resources and find when the receivers can share entanglement and can broadcast messages, they can verifiably securely perform the multi-party computation needed to determine the channel state which encrypts each transmission

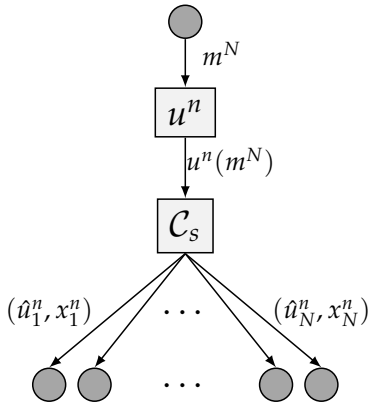


Figure 2.21.: Depiction of the causal, state-dependent, broadcast channel. The sender at the top sends m^N to the encoder u^n which encodes the message as u^{Nn}. After encoding, the message is put through the state-dependent channel $\mathcal{C}_s$, where for each bit of u^{Nn} sent through the channel, a state is selected uniformly at the random game using the modulo sum of x^N. One piece of x^N is sent to the respective receiver until all x_i^n are received. The receivers can then use their part of x^{Nn} as input to their coordinator q to receive γ_i^n which can be used to aid in decoding $\hat{u}_i^n$.

from the sender. We determine the communication complexity for each case where communication is possible.

2.4.2. Notation

Given a finite alphabet $\mathbf{X}$, the set of probability distributions on it is $\mathcal{P}(\mathbf{X})$. The corresponding state space for quantum systems on a finite-dimensional Hilbert space $\mathcal{H}$ is denoted $\mathcal{S}(\mathcal{H})$. The n-fold composition $\mathbf{X} \times \ldots \times \mathbf{X}$ is written $\mathbf{X}^n$. A classical channel W with input alphabet $\mathbf{X}$ and output alphabet $\mathbf{Y}$ is defined by a matrix $(w(y|x))_{x\in\mathbf{X},y\in\mathbf{Y}}$ where $w(\cdot|x) \in \mathcal{P}(\mathbf{Y})$ for all $x \in \mathbf{X}$. The set of all such channels is denoted $C(\mathbf{X}, \mathbf{Y})$. Specific channels used in this thesis are: The identity map on $x \in \{0, 1\}$, defined by $\mathbb{1}(x) = x$, the bit-flip on $\mathbb{F}$ as $\mathbb{F}(x) = x \oplus 1$, and the binary symmetric channel (BSC) with parameter $\nu \in [0, 1]$ defined as $BSC(\nu) \coloneqq \nu\mathbb{1} + (1 - \nu)\mathbb{F}$. The entropy of $p \in \mathcal{P}(\mathbf{X})$ is $H(p) \coloneqq -\sum_{x\in\mathbf{X}} p(x)\log(x)$, (log being calculated with base 2 and using the convention $0\log(0) = 0$). The mutual information of $p \in \mathcal{P}(\mathbf{X})$ and $W \in C(\mathbf{X}, \mathbf{Y})$ is $I(p; W) \coloneqq H(p) + H(Wp) - H((W, p))$. Distribution $\pi \in \mathcal{P}(X)$ represents the uniform distribution on X.

2.4.3. Channel Model

The broadcast channel consists of one sender and N receivers, of which up to $N - 2$ can be colluding, as to satisfy the maximum collusion condition. A single sender intends to transmit a collection of private messages $m^N \coloneqq (m_1, ..., m_N)$, one to each of the respective receivers. For message transmission, the sender uses encoder $u^n : (m_1, ..., m_N) \mapsto (u_1^n, ..., u_N^n)$ which maps the messages $(m_1, ..., m_N)$ to block-length n

messages $u_i^n \in \{0,1\}^n$ for n channel uses. For convenience, we define the map $u^n(m_i) \mapsto u^n(m^N)_i$, a mapping to the i-th index of the encoded message string. Between the sender and each receiver i is a state dependent binary inference channel $\mathcal{C}_s \in C(\{0,1\},\{0,1\})$ with an environmentally controlled state $s \in \{0,1\}$. The channel behaves as follows: When $s = 0$, $\mathcal{C}_0 \coloneqq \mathbb{1}$ is the identity channel, and when $s = 1$, $\mathcal{C}_1 \coloneqq \mathbb{F}$ is the bit flip channel. The total channel $\mathcal{C}_s^N$ is a product of the channels, that is,

$$\mathcal{C}_s^N \coloneqq \prod_{i=1}^{N} \mathcal{C}_s, \tag{2.4.1}$$

where each channel in the product shares the same state s.

The main requirement for channel state selection is that the state of the channel is uniformly random between two outcomes such that for three or more players, no receiver can infer the channel state based on their individual state information. We use one such example as follows. Prior to transmission of $u_t = \{0,1\}^N$ at time t, an environmental state $x_t^N \in \{0,1\}^N$ is realized according to the uniform distribution on $\{0,1\}^N$. For each i, one part $x_{i,t}$ of x_t^N is sent along with other information to receiver i with no encoding or effects from the channel when a transmission is made. The state s at transmission t is selected using $s = \sum_i x_{i,t} \mod 2$. For convenience we define $\Phi(x_t^N) = \sum_i x_{i,t} \mod 2$. We use this example modulo sum throughout this work. $\Phi(x^{Nn})$ denotes $(\Phi(x_1^N), ..., \Phi(x_n^N))$.

When the sender sends a message to the receivers, the message is encoded into n bits and transmitted over n uses of $\mathcal{C}_s^N$. At each transmission t the receivers receive their part of x^N and use it to assist in decoding their private message from the sender. Each receiver $i \in [N]$ can use $\{x_{i,t}\}_{t=1}^n$ to aid in selecting a decoder $d_i^n \in C(\{0,1\}^n \times \Gamma^n, \mathbf{M}_i)$, $\mathbf{M}_i$ the codebook for receiver i, which takes as input n outputs of the channel $\mathcal{C}_s$ and a coordination parameter $\gamma_i^n \in \Gamma^n$ which serves to coordinate the N receivers while decoding. The parameter $\gamma_i^n \in \Gamma^n$ is distributed according to the channel $q \in C(\{0,1\}^{Nn}, \Gamma^{Nn})$ which we call the coordinator. Each receiver inputs their part x_i^n of x^{Nn} to q and receives a response γ_i^n which is used for decoding.

Definition 9 (Non-signalling channel)**.** *A channel $q \in C(\mathbf{X}, \Gamma)$ is called non-signalling if, for all γ_1, x_1, x_2, x_2',*

$$\sum_{\gamma_2} q(\gamma_1, \gamma_2 | x_1, x_2) = \sum_{\gamma_2} q(\gamma_1, \gamma_2 | x_1, x_2') \tag{2.4.2}$$

and for all γ_2, x_1, x_1', x_2,

$$\sum_{\gamma_1} q(\gamma_1, \gamma_2 | x_1, x_2) = \sum_{\gamma_1} q(\gamma_1, \gamma_2 | x_1', x_2). \tag{2.4.3}$$

In this thesis, we use various approaches at the receivers for determining the decoding parameter γ_i^n. Here we consider the following three scenarios:

1. Classical communication is not available between receivers, but shared entanglement is.

2. Classical communication is available between receivers but not entanglement.

3. Classical communication and shared entanglement are both available.

In the first case, correlations are non-signaling, whereas in the other two, they are signaling. We define the channel formally as follows:

Definition 10 (Broadcast Code)**.** *A broadcast code C for block-length n and N receivers consists of message sets $\mathbf{M}_1, ..., \mathbf{M}_N$. Moreover C contains an encoder $u^n(m_1, ..., m_N)$ where $u^n : (m_1, ..., m_N) \mapsto (u_1^n, ..., u_N^n) \subset \{0,1\}^{Nn}$. Next, for each i, C contains a causal decoder $d_i^n(\hat{m}_i | u_i^n, \gamma_i^n)$ which decodes the channel output when n bits arrive. d_i^n assigns an estimate $\hat{m}_i \in \mathbf{M}_i$ using decoding sets $D_i^{\gamma_i} := \{D_{i,j}^{\gamma_i}\}_{j=1}^{|\mathbf{M}_i|} \subset \{0,1\}^n$ where for a fixed i and $j \neq k$ $D_{i,j}^{\gamma_i} \cap D_{i,k}^{\gamma_i} = \emptyset$.*

Definition 11 (Transmission Error)**.** *Let C_n be a broadcast code with block-length n and N receivers transmitting over $\mathcal{C}_s^N$ as defined in* (2.4.1)*. Let $M := \prod_{i=1}^N |\mathbf{M}_i|$. The average probability for transmission success is,*

$$P_{succ}(C_n) = \frac{1}{M 2^{Nn}} \sum_{m^N} \sum_{\substack{x^{Nn}, \\ \gamma^{Nn}}} q(\gamma^{Nn} | x^{Nn}) \cdot \prod_{i=1}^{N} d_i^n(D_{i,m_i}^{\gamma_i^n} | \mathcal{C}^n_{\Phi(x^{Nn})}(u^n(m_i)), \gamma_i^n). \tag{2.4.4}$$

where $\mathcal{C}^n_{\Phi(x^{Nn})}(u^n(m))$ represents the channel output after n uses of $\mathcal{C}_{\Phi(x_t^N)}$ using $\{x_t^N\}_{t=1}^n$ for transmission t to send the t-th bit of encoding $u^n(m)$. The probability of error is defined as $P_{err}(C_n) = 1 - P_{succ}(C_n)$.

Definition 12 (Achievable Rate Tuple)**.** *A rate tuple $(R_1, ..., R_N)$ is called achievable if there exists a sequence of causal codes $(C_n)_{n \in \mathbb{N}}$ such that for any $i \in \{1, ..., N\}$,*

$$\liminf_{n \to \infty} \frac{1}{n} \log |\mathbf{M}_{i,n}| \geq R_i \tag{2.4.5}$$

while at the same time

$$\lim_{n \to \infty} P_{err}(C_n) = 0. \tag{2.4.6}$$

The capacity region of a channel is the closure of all achievable rate tuples.

In order to determine the channel state for each transmission in this communication scenario, the receivers will need a scheme such that they can determine the modulo sum of the combined state information of each other receiver. Since some of the receivers are possibly colluding, each receiver should not simply give away their state information and thus use a random variable R_i to encode X_i with. Formally, we enforce that for any proposed protocol for multi-party modulo sum, it must be that for colluding receivers $Z \subset [N], |Z| \leq N - 2$ and each for all non-colluding receivers $i \in [N] \setminus Z$,

$$I(R_i; Y_Z) = 0, \tag{2.4.7}$$

where Y_Z is all of the information obtainable by the coalition excluding information derivative to the multi-party calculation. One might consider enforcing $I(X_i; Y_z) = 0$

for all i as well, but for any multi-party summation protocol, there is always a way to determine the sum of the inputs of the non-colluding parties amongst the coalition. We also enforce a reliability condition which is that when all N receivers perform the protocol honestly, then each of the receivers determines the channel state. Once the channel state is determined for each transmission, they generate a variable γ^{Nn} which contains the state information for each transmission $1 \leq t \leq n$. With this, each receiver i can choose the respective decoding set $D_i^{\gamma_i^n}$.

2.4.4. Entanglement Without Classical Communication

Theorem 13. *When classical communication between the receivers is not available but entanglement is, the rate region equals 0^N. Moreover, without classical communication, no protocol exists using only entanglement to compute the modulo sum of the channel state information.*

In this scenario, the receivers cannot classically communicate but can share entanglement resources in any form distributed prior to transmission. For each transmission t, each receiver i receives their bit $x_{i,t}$ where based on the channel state selection mechanism, each $x_{j,t}$ is independent of $x_{i,t}$ for $i \neq j$. The task for the receivers is to correlate themselves such that they have a better than $p = 1/2$ chance of guessing the channel state. To do this, the parties need to devise a joint measurement of their entangled states that can signal, contradicting the no-signaling theorem. Because the channel state is selected using an unbiased modulo sum the channel state equals the identity channel as it is the bit flip channel, the overall channel is a binary symmetric channel with $p = 1/2$, well known to have 0 capacity. That the parties are malevolent makes no difference here as there is no way for the receivers to gain any information from the other receivers.

Proof. For the channel C_s^N, assume there is an achievable rate tuple $(R_1, ..., R_N)$ where for at least one i, $i = 1$ without loss of generality, $R_i > 0$. Then, for any $\epsilon > 0$ there is an n large enough such that,

$$1 - \epsilon \leq \frac{1}{M2^{Nn}} \sum_{m^N} \sum_{\substack{x^{Nn} \\ \gamma^{Nn}}} q(\gamma^{Nn}|x^{Nn}) \cdot \prod_{i=1}^{N} d_i^n(D_{i,m_i}^{\gamma_i^n}|\mathcal{C}_{\Phi(x^{Nn})}^n(u^n(m_i)), \gamma_i^n) \tag{2.4.8}$$

$$\leq \frac{1}{\mathbf{M}_1 2^{Nn}} \sum_{m_1} \sum_{\substack{x^{Nn} \\ \gamma^{Nn}}} q(\gamma^{Nn}|x^{Nn}) \cdot d_1^n(D_{1,m_1}^{\gamma_1^n}|\mathcal{C}_{\Phi(x^{Nn})}^n(u^n(m_1)), \gamma_1^n). \tag{2.4.9}$$

In order to better predict $\Phi(x^{Nn})$, the receivers coordinate with $q(\gamma^{Nn}|x^{Nn})$. Shared entanglement alone is non-signalling [80] and so

$$\sum_{\gamma^{(N-1)n}} q(\gamma_1^n, \gamma^{(N-1)n}|x_1^n, x^{(N-1)n}) = q_1(\gamma_1^n|x_1^n). \tag{2.4.10}$$

The channel state $\Phi(x^{Nn})$ for channel $\mathcal{C}_{\Phi(x^{Nn})}^n(\hat{u}^n|u^n(m_1))$ and decoder d_1^n with n transmissions leads to an effective channel from sender 1 to receiver 1. Let g denote

the channel from sender 1 to receiver 1, including the reception of a copy x of the state variable x_1.

$$g(x,y|u) = 2^{-N}\sum_{x^N}\delta(x,x_1)\delta(y,\Phi(x^N)\oplus u) \tag{2.4.11}$$

$$= \tfrac{1}{2}\sum_{x_1,\hat{x}}\delta(x,x_1)\tfrac{1}{2}\delta(y,\hat{x}\oplus x_1\oplus u) \tag{2.4.12}$$

$$= \pi(x)\pi(y). \tag{2.4.13}$$

Obviously, for the Shannon capacity C of g it holds $C(g) = 0$. Therefore, no coding scheme, and in particular none that uses the particular random decoder

$$\hat{d}(m|x_1^n,y^n) := \sum_{\gamma_1^n} q(\gamma_1^n|x_1^n)\mathbb{1}_{D_m^{\gamma_1^n}}(y^n), \tag{2.4.14}$$

can transmit at positive rates over g. This contradicts (2.4.8) and thus only 0^N is achievable. □

2.4.5. Classical Communication Without Entanglement

In the next two cases, we develop a signaling coordinator q that can be used to achieve positive capacity.

Theorem 14. *When classical communication is allowed between the receivers, using Protocol 5 the rate tuple 1^N is achievable and the communication complexity is $\Omega(N^2)$ using a conditionally secure protocol.*

Remark 15. *For unconditional security, the secrecy of each communication channel between each pair of connected receivers must be verified, which would add significant communication resource overhead to the protocol. With only classical resources, one could consider private key distribution, but with colluding receivers, this would not work since private keys could be shared in the collusion.*

When classical communication is available, the receivers can collaborate to each send the x_i amongst each other receiver to perfectly determine the channel state s for each transmission. When all receivers are honest, nothing more needs to be done, however, when some receivers are dishonest, more complexity is needed to preserve secrecy. We can turn to theory for secure multi-party computation (MPC), specifically, secure multi-party modulo summation to compute the common function $\Phi(x^N)$. Since in this case the parties are not allowed to share entanglement and are allowed only classical communication, we only investigate purely classical strategies. In all strategies, the assumption of minimum non-colluding parties (i.e., at most $N-2$ parties can collude) is made. In the following, we sketch the strategy and the complexity analysis.

Strategy I: Secure multi-party modulo summation

For the first strategy, we consider a protocol that requires secure channels between each of the receivers. We enforce this in this case in a purely classical way (e.g., using

quantum key distribution schemes is prohibited). If we assume a standard public key cryptography system then there is an additional communication overhead of $\Omega(N^2)$. This is because with just forward communication as the protocol needs,

$$(N-1)(N-2)/2 = (N^2 - 3N + 2)/2, \tag{2.4.15}$$

secure channels are needed and therefore as many public keys need to be distributed. Commonly used public key cryptography schemes like RSA are not unconditionally secure as they can be broken with enough computing power or a quantum computer [2]. In contrast, as we will see in Section 2.4.6, when entanglement is available, secure communication is not necessary.

In [78], Chor and Kushilevtiz give a multi-party protocol to compute a modulo sum with a security trade-off parameter $t \in [N]$. Here t represents that no coalition of size at most t can infer any additional information from the other receivers' x_i from the other $N - t$ parties other than the modulo sum value, with $t \leq N - 2$. This protocol uses $N\lceil (t+1)/2 \rceil$ conferencing messages assuming preestablished secure communication channels. For completeness, we state the protocol in Protocol 4, where t is assumed to be $N - 2$.

Protocol 4 Classical State Decoding

1: **for** $i \in \{1, ..., N-2\}$ **do**
2: Receiver i awaits $z_{j,i}$ from receivers $j < i$ and calculates $w_i = \sum_{j=1}^{i-1} z_{j,i} \mod 2$ with $w_1 = 0$
3: Receiver i selects uniformly at random $z_{i,i+1}, z_{i,i+2}, ..., z_{i,N-1}$ each from $\{0, 1\}$
4: Receiver i determines $z_{i,N}$ such that $x_i + w_i = \sum_{j=i+1}^{N} z_{i,j}$ and sends $z_{i,j}$ to the respective receiver $j > i$
5: Receiver $N-1$ computes $z_{N-1,N} = x_{N-1} + \sum_{j=1}^{N-2} z_{j,N-1} \mod 2$ and sends it to receiver N
6: Receiver N computes $s = x_N + \sum_{j=1}^{N-1} z_{j,n} \mod 2$ and broadcasts s to all other receivers

The proof of correctness for this protocol is given explicitly in [78]. The downside of using this protocol alone is that for decoding a message of length n, the protocol needs to execute n times. In Strategy II, we use this protocol to generate a zero-sum random variable in the first step, and then using this zero-sum random variable, one can save resources by reusing it for each transmission. We explore the pros and cons in depth.

Strategy II: Modulo-zero summation from modulo summation

We consider another strategy where the receivers use Protocol 4 to produce random variables $R = (r_1, ..., r_N)$ such that $\sum_{i=1}^{N} r_i = 0$ and then perform steps 4 and 5 from Protocol 7 using this zero-sum randomness. To generate such a zero-sum random variable, one can use the protocol defined in [79, Protocol 2] which we include explicitly as a subprotocol in Protocol 5. Once a zero-sum realization is generated, it can be used repeatedly to decode a message of length n. The full protocol is given explicitly in Protocol 5.

Protocol 5 Classical Secure Multi-Party Summation

1: Each receiver i generates a uniform random number Y_i independent of the other receivers
2: The receivers run Protocol 4 with $\{Y_i\}_{i=1}^N$ and each receiver will have the result $\sum_{i=1}^N Y_i$
3: Receiver 1 uses random variable $r_1 = Y_1 - \sum_{i=1}^N Y_i$, receivers $i \in \{2, ..., N\}$ use $r_i = Y_i$
4: To determine channel state s_t for transmission $1 \leq t \leq n$, each receiver broadcasts $s_{i,t} := x_{i,t} + r_i$. The channel state is found by $s_t = \sum_{i=1}^N s_{i,t} \mod 2 = \sum_{i=1}^N x_{i,t} + r_i \mod 2 = \sum_{i=1}^N x_{i,t} \mod 2$

We summarize the communication complexity analysis. For establishing (conditionally) secure connections using public key distribution techniques requires each receiver (in the limit of increasing receivers) to send a public key to each other receiver consuming $\Omega(N^2)$ messages. Next, generating a zero-sum random variable using Protocol 4 requires $\Omega(Nt)$ for t colluding parties. Once a zero-sum random variable realization is successfully generated, the receivers can use step 4 from Protocol 7 which requires each sender to use a broadcast channel to transmit their encrypted state information. In total, the communication is, therefore, $\Omega(N^2 + Nt)$.

That the zero-sum random variable can be recycled because of its zero-sum condition. For example, assume $N-2$ parties collude and share their r_i in an attempt to determine, without loss of generality, r_1 and r_2. Then at the end of the protocol, the $N-2$ colluding parties have $(x_1 + r_1) + (x_2 + r_2) + \sum_{i=3}^N r_i = x_1 + x_2$, and thus no reminisce of r_1 or r_2 remains.

2.4.6. Classical Communication with Entanglement

Theorem 16. *When classical communication and entanglement are allowed between the receivers, there exists a multi-party summation protocol such the rate tuple 1^N is achievable with classical communication complexity $\Omega(N)$ and entanglement resource complexity $\Omega(N^2 m)$ with unconditional security.*

We sketch the proof. To determine the channel state s, the receivers perform Protocol 7 for multi-party modulo summation extended from [79, Protocol 4], which offers verifiable randomness for zero-sum random variables and unconditional security. The protocol uses phase-GHZ states defined as follows. Let $R^N := \{r^N = (r_1, ..., r_N) : \forall i, r_i \in \{0, 1\}, \sum_{i=1}^N r_i = 0\}$. The phase-GHZ state is defined as:

$$|GHZ\rangle_p := \frac{1}{\sqrt{2^{N-1}}} \sum_{r^N \in R^N} \left| r^N \right\rangle . \tag{2.4.16}$$

This ensures every joint measurement result of $|GHZ\rangle_p$ sums to 0. With security parameter f, we write the necessary protocols that the receivers use to securely communicate their piece of the channel state such that the security is verifiable. Using these protocols, the receivers can with certainty and security (with respect to parameter m) determine the channel state s for each transmission, thereby allowing for perfect decoding of the message.

The communication complexity of this approach is as follows. As far as we know, there are no protocols currently used for distributing GHZ states, and so to estimate

Protocol 6 Phase-GHZ State Validation

1: With $4Nf$ copies of the $|GHZ\rangle_p$ state, randomly divide the states into $4f$ groups with n states each.
2: Apply measurements to the respective groups using the measurement structure in [79, Table 1].
3: Await a broadcast from all other receivers with their measurement results from the previous step.
4: Check that inequalities (9) and (10) in [79, Protocol 3] hold.
5: If the inequalities hold report success, otherwise report failure.

Protocol 7 Entanglement Assisted State Decoding

1: The N receivers generate $4N^2m + 1$ copies of the $|GHZ\rangle_p$ state.
2: From the $4N^2m + 1$ $|GHZ\rangle_p$ states, each receiver allocates $4N^2m$ of the copies, randomly selecting $4Nm$ copies from their respective part, and applies as a sub-protocol Protocol 6. If the sub-protocol runs successfully, they continue to the next step.
3: Each receiver i measures the last remaining copy in the computational basis and stores the output as r_i.
4: For each $x_{i,t}$, each receiver i calculates $s_{i,t} := x_{i,t} + r_i$ and broadcasts all $\{s_{i,t}\}_{t=1}^{n}$ to all other receivers.
5: Each receiver calculates $s_t = \sum_{i=1}^{N} s_{i,t} \mod 2 = \sum_{i=1}^{N} x_{i,t} + r_i \mod 2 = \sum_{i=1}^{N} x_i \mod 2$.

the communication complexity of the task we assume there is a communal source generating $|GHZ\rangle_p$ states and transmitting them to the receivers in a "frame" of qubits using one transmission to transmit the $4N^2m + 1$ $|GHZ\rangle_p$ states, namely $\Omega(1)$ classical resources to generate $\Omega(N^2m)$ entanglement resources. Next, the receivers use Protocol 6 to validate their GHZ states using $\Omega(N)$ messages over a broadcast channel. Finally, the receivers again use $\Omega(N)$ messages to broadcast the state information encoded with the zero-sum random variable. In summary, $\Omega(N^2m)$ entanglement resources are used with $\Omega(N)$ classical resources for verifiable unconditionally secure decoding of the state information satisfying all of the protocol requirements.

In each case, the converse arguments are the same. The broadcast channel transmits the messages in such a way that without signaling communication amongst the receivers, the channel acts as a $BSC(0.5)$ channel, well known to have a capacity of 0. Because the channel state information vector x^N is generated i.i.d. randomly, there is no correlation between any two receivers' individual state information. Because of this, there is no way for the receivers to achieve any positive rate without the cooperation of all receivers.

In summary, we have constructed a communication scenario over a causal broadcast channel such that for the channel to have a positive capacity, the receivers must cooperate to compute a multi-party modulo sum. We considered three scenarios where the receivers have access to various resources and showed that when the receivers can share entanglement and use a separate classical broadcast channel to cooperate, they can most efficiently determine the channel state to achieve the full capacity of the channel. We can further investigate different methods of selecting the channel state based on results where there is a known quantum advantage.

3. Networked Quantum Computing

Another major use case of future quantum networks will be to connect physically separated quantum computers or to access remote quantum computers. Distributed quantum computing requires a paradigm shift concerning classical distributed quantum computing. In this chapter, we will explore two types of future distributed quantum computing approaches. The first method of distributed quantum computing is to connect, in a classical sense, to a quantum computer operating in a cloud environment to run quantum algorithms. Here we explore integration of a practical application of quantum computing, running in the IBM Quantum Experience environment. We conduct several experiments, benchmarking how well a commonly used algorithm can be performed on real hardware, using a cloud-based integration. We further explore a real-use, unsupervised learning experiment, using quantum computers to cluster a dataset of the energy grid network in Germany. Next, in a more traditional sense, we explore how multiple quantum computers can interconnect to perform parallel and distributed quantum algorithms. We formalize the meaning of a parallel and distributed quantum algorithm and investigate some well-suited candidate applications for distributed quantum computing.

3.1. Quantum Computing in the Cloud

Section based on the article: "Practical Quantum K-Means Clustering: Performance Analysis and Applications in Energy Grid Classification"

Given the challenging engineering requirements for building and maintaining quantum computers, it is likely that quantum computers will only be accessible through cloud services for the majority of users. Quantum computers, depending on the qubit technology, can require a complex construction and maintenance schedule that makes it impractical for the average user to own [81]. Rather than building quantum computers as a hardware product to sell to consumers, companies like Amazon, Microsoft, and IBM are rather developing cloud-based platforms for online access to their quantum devices. Although these quantum cloud services are currently accessible, the question of how useable they are for practical, industrial use cases arises. For our interest, we ask: Can these quantum computers produce accurate enough distance estimates for clustering? Can we achieve any speedup with them currently? The focus of this work is therefore to benchmark this simple, common use-case using the IBM Quantum cloud services [82].

Clustering algorithms can be used on unlabeled data to find relationships between the data's various features. To perform clustering, an algorithm introduced by Lloyd in 1982 called k-means clustering [48] can be used. The k-means algorithm takes as input a collection of unlabeled data points, or feature vectors, and outputs a list of labels, one for each data point. The data points are labeled based on the minimal distance to a particular centroid. During execution, the algorithm improves the centroid locations by running iteratively, updating the location to be the mean of the data points that are

determined nearest to them based on a distance metric.

Classically, the usual method for measuring the distance between centroids and data points is to simply compute the Euclidean distance. For feature vectors of N features, computing the Euclidean distance requires $O(N)$ computational steps. With a quantum approach, using quantum amplitude embedding (see [83] for details), one can encode length-N vectors into $O(\log_2 N)$ qubits, an exponential decrease in resources for embedding, assuming one can load quantum states into a quantum random access memory [84]. With this embedding, one can perform what is known as a swap test using a quantum computer, as described in [84, 85], to compute an estimate for the Euclidean distance between two vectors. Because the swap test requires several operations proportional to the number of qubits used for embedding—needed for swapping two multi-qubit states—in theory, this would result in an exponential speedup in runtime complexity. Moreover, the minimum distance to a centroid for each point can be found using Grover's search [86] for an additional quadratic speedup. Using a simpler data embedding approach like angle embedding, the theoretical advantage provided by amplitude embedding is lost, as angle embedding requires several qubits directly proportional to the data dimension. However, the benefit of using this alternative embedding is that the depth of the state preparation circuit is constant, whereas, with amplitude embedding, the state preparation circuit uses exponentially more non-local gates as the data dimension grows [87], leading to vastly deeper circuits using current approaches.

This result makes quantum clustering and nearest-neighbor classification appear as very attractive use cases for quantum computing since they both use distance estimation and therefore can benefit from a theoretical speedup. When put into practice though, there are various challenges to overcome before one can effectively perform distance estimation on a quantum computer. Moreover, a strong assumption of efficient state preparation needs to be made to have an exponential speedup using the quantum approach over the classical approach [88]. Nonetheless, with this article, we aim to expand the results related to clustering on real, gate-based, quantum hardware, and in particular, to explore how this type of algorithm can be executed on IBM's quantum cloud computing service. In this work, we use only the software libraries and services available to us with no direct hardware access, aiming to demonstrate how well one can expect quantum clustering to perform using generally available resources.

We begin by reviewing how we perform data encoding and how we calculate the distance estimation in our quantum algorithm using the cloud service. We then benchmark how well distance estimation can be performed, testing for various Euclidean distances and dimensions in simulation and on hardware. Further, we analyze various clustering experiments using synthetic data with two and four dimensions with several datasets. Given the results of the performance analysis, we then apply the findings to a non-trivial clustering problem that is relevant in the energy sector. In particular, we show that by decomposing high-dimensional vectors into 2-dimensional subspace projections, we can compute the overall distance in a parallel fashion which significantly reduces the error induced by existing quantum hardware, while simultaneously reducing the total number of circuits.

3.1.1. Related Work

Performing clustering and nearest-neighbor-type algorithms using quantum computers has been studied in various contexts. Improving the encoding strategy to work better with IBM's quantum computers was studied by Khan et al. in [89]. In the article, they describe an encoding mechanism for feature vectors and benchmark the approach on IBM's quantum computer. Feature vectors, after PCA is performed, of dimension two are considered and benchmarked with the MNIST dataset using quantum hardware. Using the IonQ quantum hardware, Johri et al. perform data classification using clustering on their trapped ion quantum computer [90]. In their work, they define an optimized method for encoding 8-dimensional classical data into the quantum computer and use PCA to benchmark against MNIST data for ten different labels, and perform their quantum algorithm for nearest-neighbor classification. In these works, an explicit benchmarking of distance estimation accuracy is not demonstrated. Moreover, the only experiments tested on the IBM quantum system were of two dimensions. In a quantum annealing setting, clustering has also been considered. The authors of [91] and [92] map a clustering problem to a quadratic unconstrained binary optimization (QUBO) problem for an adiabatic quantum computer and use hardware to test their approach. Quantum annealing uses a different approach to quantum computing versus the gate-based model, indeed no swap test is involved, and the results from annealing experiments do not paint a clear picture of performance using a universal quantum computing approach.

In [93], Benlamine, et al. review three methods for distance estimation using a quantum approach and benchmark the approaches using nearest-neighbor classification in simulation only, not testing their approaches on real hardware. Further modified quantum clustering approaches were proposed in [94, 95]. These works did not perform tests on real physical hardware, and therefore did not benchmark their performance, as we do in this work. In [96], Nguyen et al. run experiments to test the accuracy of the swap test using their trapped ion system in a continuous variable setting, but do not perform any experiments of clustering or distance estimation for classical input vectors. Overall, our work is the first to benchmark these popular swap-test-based distance estimation techniques in a general way and for classification and clustering.

3.1.2. Summary of Contributions

In this work, we analyze how well current quantum computers can perform unsupervised clustering. To do this, we use two different distance metrics that are based on the swap test and thoroughly benchmark their accuracy. The swap test is a general method for computing distance between vectors on a quantum computer and is therefore important to thoroughly understand. The two metrics we use are based on popular, well-known, methods of data embedding, and therefore of general interest to explore, something this work does for the first time on real quantum hardware. To compare the accuracy of the real quantum device to the true result, we use noiseless simulation as the ground truth. To perform many distance measurements at once for k-means clustering, we define two approaches to parallelize the calculations. In the first case, we offer a parallelization approach when the quantum computing platform does not

allow for sending circuits to execute in bulk. We tested these approaches to perform benchmarking, but because of the large performance improvement when using bulk circuit execution of the second approach, we use only the second approach for our analysis in this work. We benchmark distance estimation via the swap-test, varying the scale of the distance, the number of shots used, and the dimension of the data. To the best of the author's knowledge, no such benchmarking has been previously demonstrated.

Next, we analyze unsupervised clustering on real quantum devices. Using the results of the distance metric analysis, we benchmark the best cases in terms of dimension—2D and 4D—using synthetic data. We found that the accuracy of the real devices using standard approaches and synthetic data proved relatively low. We next tested the ability to perform a true unsupervised clustering problem using real data from the German energy grid. We found that using a standard approach for quantum k-means clustering, the accuracy result output from real quantum hardware is low. We then tested our novel approach to performing the distance metrics, which is to decompose the distance calculation into smaller-dimensional projections, and then use quantum circuits that compute multiple swap tests at once, thereby accommodating the additional swap-tests required after projecting to smaller dimensions. With this approach, we were able to achieve high accuracy concerning the classical results, thus demonstrating a scalable approach to distance estimation with high accuracy on real quantum devices.

3.1.3. Program Setup and Configuration

In this section, we review the setup and configuration used to perform clustering algorithms. In the first subsection, we review how the synthetic data is generated. Next, we review how the quantum algorithm works as well as the two methods used for classical data embedding in quantum states. Finally, we review the software approach used to execute the algorithms on the quantum cloud hardware.

Generating Synthetic Data and Quantum Data Loading

In the experiments conducted, we used synthetic data generated with varying dimensions, number of clusters, cluster variance, and minimal distance between the centers. At a high level, the cluster generation algorithm used works by first selecting k center points to then generate cluster data around them. Using the center points as the multi-dimensional mean of a multivariate normal distribution, a set number of cluster points are generated surrounding the center. Input to this algorithm is a cluster-variance parameter which we use to set the variance level of the respective dimension to control the "tightness"—how we measure the difficultly of clustering—of the cluster. To avoid the randomly initialized center points being too close to each other, an additional step that resets a center point if it is within some ϵ distance from the already initialized center points is added. The synthetic data generated can be seen in Figure 3.8.

To perform the quantum distance estimation algorithm, the generated data points first need to be embedded into a multi-qubit quantum state. For this, we use two types of embeddings, namely amplitude embedding and angle embedding, and test them

Algorithm 5 Synthetic Cluster Data Generation

Input:

- k: The number of clusters
- dim: The dimension of the data
- var: The variance of the normal distribution
- seed: The random seen
- ϵ: The minimum distance between two cluster centers
- rng: The range of the data in the form of a tuple
- num: The number of points surrounding a centroid

Output: A set of cluster data

set the random seed
$centers \leftarrow$ empty list
$data \leftarrow$ empty list
$i \leftarrow 0$
while $i < k$ **do**
  $point \leftarrow random(low{=}rng[0], high{=}rng[1], size{=}dim)$
  re-initialize $point$ while $dist < \epsilon$ to other points in $centers$
  $centers.add(point)$
  $i \leftarrow i + 1$
$cov \leftarrow \mathbb{I}_{dim}/10^{var}$ ▷ Setting the co-variance matrix
for $center \in centers$ **do**
  $cluster \leftarrow normal(mean{=}center, cov{=}cov, points{=}num)$
  $data.add(cluster)$
return data

independently. To perform the embeddings, we use the circuit structure shown in Fig. 3.1 and Fig. 3.2. To implement amplitude embedding in code, we simply use the built-in `initialize` function offered by Qiskit, which is a function that takes as input a real vector and returns the necessary gate set for complex amplitude embedding [97]. For angle embedding, we use two-dimensional rotations to embed two dimensions of a data vector per one qubit. This type of embedding is also referred to as dense angle embedding [98], but in this thesis, we will only use the name “angle embedding" to refer to it. The scaling in terms of circuit depth and the number of two-qubit gates varies significantly between the two embeddings. When embedding data using angle embedding, gate depth does not increase with dimensional, but circuit width grows linearly. On the other hand, embedding classical data using amplitude encoding can scale quite poorly in depth concerning the data dimension [87]. We plot the scaling concerning the data dimension in Fig. 3.3 where we plot both the depth and the number of non-local gates required for initializing data for a completely connected quantum computer (solid line) and the 65-qubit IBM Brooklyn topology (dashed line), followed by a swap-test.

Quantum Circuits for Distance Estimation

To perform clustering, we replace the distance calculation from the classical algorithm with a quantum algorithm for distance estimation. Using amplitude embedding, we use a direct Euclidean distance approximation, based on the one developed in [84],

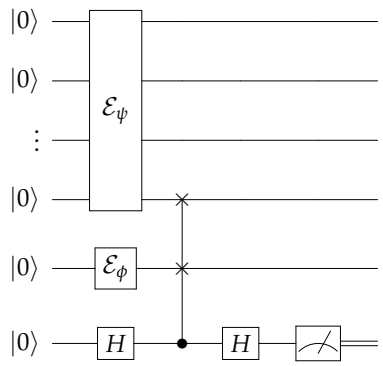


Figure 3.1.: Circuit for embedding classical data using amplitude embedding. We make use of the built-in features of Qiskit to initialize the amplitude-encoded data.

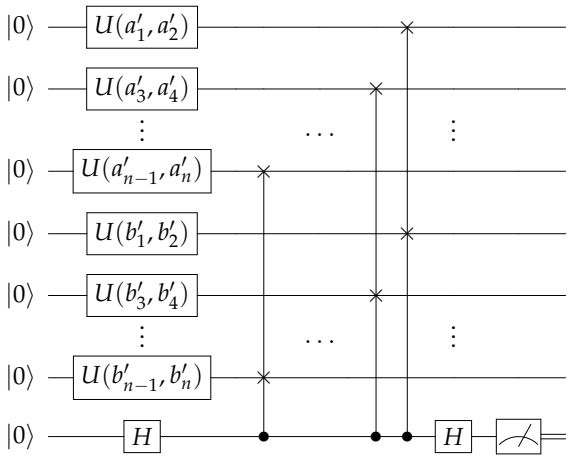


Figure 3.2.: Circuit for embedding classical data using angle embedding with swap test.

reiterated in [99], and for angle embedding, we define a simple encoding that properly scales the data for embedding. The algorithm used for distance estimation first embeds the data for two vectors and then performs a swap test. To then approximate the distance, several repetitions of the circuit are used to buildup measurement statistics for a single ancilla qubit. Once the repetitions are complete, by using the measurement statistics, the distance estimation can be reconstructed using the following formulas:

$$\Pr(0) = \frac{1}{2} + \frac{1}{2}\left|\langle\psi|\phi\rangle\right|^2, \tag{3.1.1}$$

$$\Pr(1) = 1 - \Pr(0) = \frac{1}{2} - \frac{1}{2}\left|\langle\psi|\phi\rangle\right|^2. \tag{3.1.2}$$

By determining the probability for a 0 or 1 measurement outcome of the ancilla qubit in the swap-test using a computational basis measurement, we can estimate $|\langle\psi|\phi\rangle|^2$, thereby providing an approximation to the inner product for our choice of embedding or at least a measure that scales with the inner-product in the case of angle embedding.

In the two cases of amplitude embedding and angle embedding, the swap-test procedure differs in terms of the number of controlled swaps— also known as Fredkin gates—used due to the difference in qubit resources required to perform the data embedding as well as the data representation strategy. For two (not necessarily normalized) data vectors $a = (a_1, a_2, ..., a_n)$ and $b = (b_1, b_2, ..., b_n)$—which in the case of clustering or classification would be one centroid and one data point—the two quantum states that are compared when performing the swap-test when using amplitude embedding represent, as defined in [84]:

$$|\psi\rangle := \frac{1}{\sqrt{2}}\left(|0\rangle|a\rangle + |1\rangle|b\rangle\right), \tag{3.1.3}$$

$$|\phi\rangle := \frac{1}{\sqrt{Z}}\left(|a||0\rangle - |b||1\rangle\right), \tag{3.1.4}$$

where $Z := |a|^2 + |b|^2$. To recover the distance estimation in this case, an ancilla qubit initialized in the $|0\rangle$ state is added to the system, and after applying a Hadamard gate to it, a Fredkin (or controlled-swap) gate is applied with the ancilla as the control and $|\psi\rangle$ and $|\phi\rangle$ as the target systems. In this case, since $|\phi\rangle$ is just one qubit, one Fredkin gate is needed, greatly reducing the number of controlled swaps needed. Another Hadamard gate is applied to the ancilla qubit and then it is measured. By repeating the process a number of times, $\Pr(0)$ can be estimated to then recover the distance:

$$|a - b|^2 = 4Z(\Pr(0) - 0.5). \tag{3.1.5}$$

For angle-embedding, we prepare the two vectors $a' = (a'_1, a'_2, ..., a'_n)$ and $b' = (b'_1, b'_2, ..., b'_n)$ which are angle-encodings for a and b respectively and defined as,

$$a'_i := \tfrac{\pi}{2}(a_i + 1) \tag{3.1.6}$$

$$b'_i := \tfrac{\pi}{2}(b_i + 1). \tag{3.1.7}$$

used to ensure the values are between 0 and π. With this embedding, we can encode data using a two-dimensional rotation operation defined by U:

$$U(\theta, \gamma) := \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ e^{i\gamma}\sin\frac{\theta}{2} & e^{i\gamma}\cos\frac{\theta}{2} \end{pmatrix}. \tag{3.1.8}$$

We encode a' and b' respectively in $\lceil n/2 \rceil$ qubits, using 1 qubit per two dimensions of each vector resulting in the overall states,

$$|\psi\rangle := \bigotimes_{i \in odd(n)} U(a'_i, a'_{i+1}) |0\rangle \tag{3.1.9}$$

$$|\phi\rangle := \bigotimes_{i \in odd(n)} U(b'_i, b'_{i+1}) |0\rangle , \tag{3.1.10}$$

where $odd(n)$ is the set of odd numbers from 1 to n, where in our case we use only powers of 2 for the data dimension, so n is always even. To recover the distance estimation in this case, again an ancilla qubit is introduced to the system. A Hadamard gate is applied to it followed by a series of $n/2$ controlled-swap gates using the ancilla as the control and one qubit from $|\psi\rangle$ and one from $|\phi\rangle$. A final Hadamard gate is applied to the ancilla qubit and then it is measured. The overall circuit is depicted in Fig. 3.2. The goal, in this case, is to produce an estimate for $\Pr(1)$, which is a valid distance metric (i.e. it satisfies $d(a, a) = 0$). Because we are using arbitrary data not normalized in advance, we accommodate for this in the distance metric as is also done in amplitude embedding. For this, we set $Z := |a|^2 + |b|^2$ and a and b are reassigned to $a = a/\sqrt{Z}$ and $b = b/\sqrt{Z}$. We define the metric as

$$d(a, b) = \sqrt{Z \cdot P(1)}, \tag{3.1.11}$$

where $P(1)$ is the same as in (3.1.2). We note that although this will normalize any two points, creating a global normalization factor based on the dataset to subdivide the degrees of freedom on the Bloch sphere more evenly would likely lead to more accurate results. For this thesis, we test a distance estimation function that takes arbitrary data as input with no additional pre-processing for scaling that can be applied to any dataset in the same way.

3.1.4. Running Clustering on the Quantum Cloud

To perform and benchmark the clustering experiments, we implemented k-means clustering algorithm as in [48] with a modularized distance estimation step so that distance calculation can be replaced for the variety of distance estimation methods used in this thesis which are the purely classical approach, a noiseless quantum simulation to verify the approach, and the same circuits as in the noiseless case but using cloud quantum computers.

To implement the circuit preparation and execution of the quantum circuits in software, we use IBM's Qiskit Python framework [100]. With Qiskit, one can construct circuits using a gate-based model. Parts of the Qiskit ecosystem are simulators and various cloud services for executing circuits online. IBM's Quantum Experience uses Qiskit natively, so circuits constructed in the Qiskit language can be sent to the IBM cloud service for execution directly. There are two methods of sending circuit jobs to the IBM cloud. The first is to send a single quantum circuit to be executed. When the circuit is sent to the cloud with the execution parameters, it is first queued until the computing resource becomes available, then the circuit is run until completion, or an

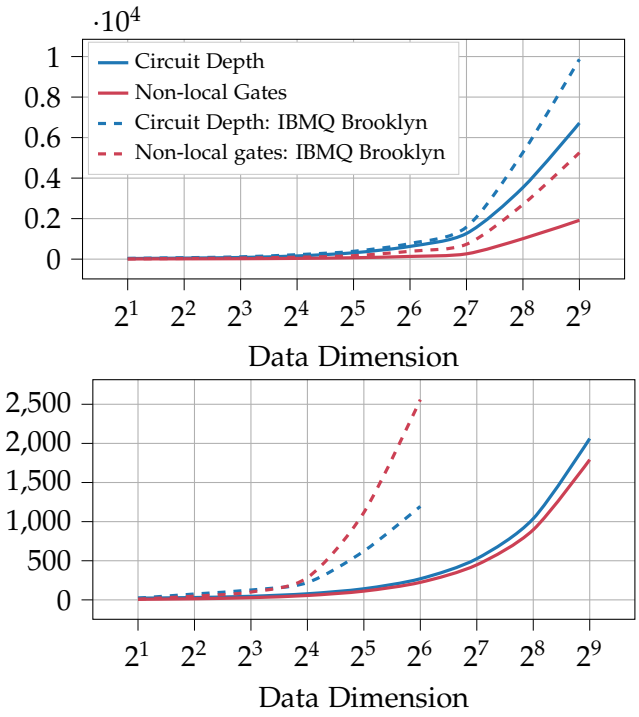


Figure 3.3.: The circuit-depth and number of non-local gates in a circuit that embeds classical data using amplitude encoding and performs the swap test of varying dimension transpiled with the basis gates for the IBM cloud computers under full connectivity and for the IBMQ Brooklyn device.

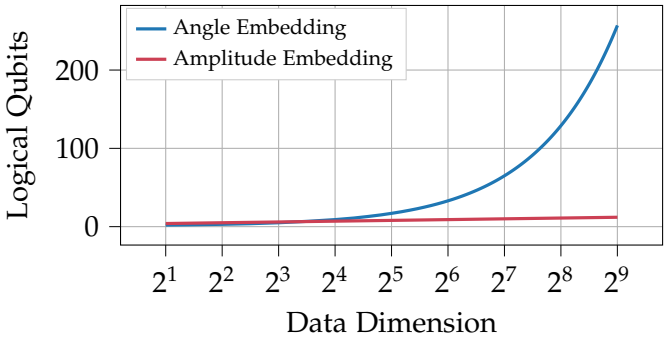


Figure 3.4.: The number of qubits required to embed classical data of varying dimension.

error occurs. For quantum clustering, many circuits need to be executed to compute a distance estimation between the k centroids and all of the data points. Sending one circuit at a time very quickly becomes a bottleneck, especially with queuing.

A property of the k-means clustering algorithm is that it is highly parallelizable, and we use this property to execute the clustering algorithm more time-efficiently. In each iteration of k-means clustering, a distance is calculated between the current set of centroids and the data points. These distance calculations are independent of each other and can be computed in parallel using multi-processing or in a batch job. In the quantum case, to parallelize, we take two approaches. Firstly, we prepare one circuit per distance estimation and then send the collection of circuits as a batch job to the cloud service. The response is the measurement results of the circuits which can then be post-processed according to the embedding. The second way we parallelize the algorithm is to embed multiple swap tests into one circuit and execute them on the same quantum computer at once. This allows for multiple distance estimations to be done on a single quantum computer.

We developed the first parallel approach in two ways, one using our approach and later, when it became available, using a more Qiskit native approach. For the first approach, we use many processes locally to send many requests simultaneously to the server to reduce queuing time. In simulation, the approach improved performance, but when there is only one quantum computer, this makes no difference. The advantage of parallel execution is achieved with multiple quantum computers working together to compute distance estimations as described in [101]. The better approach was to send the circuits in a batched job, which is a recently added feature called "circuit-runner". This feature allows the user to send up many circuits at once to be executed, where job queuing is done only once, saving a significant amount of time. The performance improvement between parallel execution and batch-circuits was significant. Indeed, some quantum cloud platforms do not yet implement batched jobs in the same way, but due to the significant performance enhancement, we foresee this to be a universal feature for all cloud platforms. To clarify the difference between the two approaches, we describe the approaches at a high-level in Algorithms 6 and 7.

3.1.5. Benchmarking Distance Estimation

To develop a clear understanding of how well clustering and nearest-neighbor classification can be performed using current quantum, it is helpful to test how accurate distance estimation can be performed on a quantum computer. In this section, we benchmark a variety of cases for performing distance estimation and compare the estimations using simulation and IBM's quantum hardware. To perform the analysis, we test three different cases. We first test how varying the number of circuit shots affects the estimation in both two and four dimensions. Next, we vary the distance of the data points also in two and four dimensions. Lastly, we test how varying the dimensions affects the estimation up to 32 dimensions using amplitude embedding and 26 using angle embedding.

In all tests, for each data point in these experiments, we use 100 repetitions, plotting the average output and the standard deviation. In each case, we use the measurement

Algorithm 6 Clustering on the Quantum Cloud with Multi-processing

Input:

- k: The number of clusters
- data: The data to cluster
- embedding: The choice of data embedding
- ϵ: The minimum distance between two cluster centers
- *maxIterations*: The maximum number of iterations to make
- *processes*: A list of running processes

Output: An ordered list of labels for the data points

centroids ← initialize the centroids using ϵ min distance
for $i <$ *maxIterations* **do**
  circuits ←generate all circuits with the data, centroids, and embedding choice
  dists ← initialize empty shared storage
  while not all circuits have been processed **do**
    if a process is idle **then**
      job ← send single circuit to server and await response
      dist ← process *job* results according to the embedding
      add *dist* with circuit number to shared storage
    else
      wait
  Sort *dists* according to circuit number
  dists ← using the returned, ordered measurement results from the server, complete the distance estimation procedure
  labels ← using the distances to the centroids, label the data points
  centroids ← with the updated labels, recompute the centroids as an average position of the labeled data. Delete centroids for empty clusters.
  check if centroid position converged and break accordingly
  $i \leftarrow i + 1$
return *labels*

error mitigation feature when running the experiments on the physical hardware. To execute the 100 instances, we make 100 copies of the circuits and use the circuit-runner service to execute the circuits in a batched job. We then process the results which significantly sped up the processing time in comparison to sending one circuit at a time.

The results of varying the number of shots are plotted in Fig. 3.5. In this experiment, we created a circuit for the vectors $(1, 0)$ and $(1, 1)$. Comparing simulation to the physical device results, in the simulation we see a convergence in the number of shots to the true answer, and moreover, the average is close to the true distance as desired. With the real hardware, we see no convergence trends behind the dotted line, and using more shots than around 2000 does not generally perform better than using the maximum number of shots 8192. In some cases, a lower number of shots performed better, having a lower variance in the standard deviation than with more shots. On the other hand, when we switched to a quantum computer that supported a higher number of shots, much better and more consistent results are seen. For 4D, the points

Algorithm 7 Clustering on the Quantum Cloud with Batched Circuits

Input:

- k: The number of clusters
- data: The data to cluster
- embedding: The choice of data embedding
- ϵ: The minimum distance between two cluster centers
- $maxIterations$: The maximum number of iterations to make

Output: An ordered list of labels for the data points

```
centroids ← initialize the centroids using ϵ min distance
for i < maxIterations do
    circuits ←generate all circuits with the data, centroids,
                and embedding choice
    job ← send the collection of circuits to the cloud server
          for processing and await response
    dists ← using the returned, ordered measurement results from
            the server, complete the distance estimation procedure
    labels ← using the distances to the centroids, label the
             data points
    centroids ← with the updated labels, recompute the centroids
                as an average position of the labeled data.
                Delete centroids for empty clusters.
    check if centroid position converged and break accordingly
    i ← i + 1
return labels
```

$(1, 0, 0, 0)$ and $(1, 1, 1, 1)$ are used, and the same effects are more or less seen. Converge is not reached using real hardware, but a relatively rapid converge is seen in simulation, where with a high number of shots, the results improve significantly.

An important point of note is that for angle embedding, the difference in output between the simulation and the hardware is much starker than with amplitude embedding. The reason behind this is since on the IBM quantum devices, the accuracy of qubit rotations is much less precise with the available gate basis of $[CX, ID, RZ, SX, X]$ and so significant differences between the simulation, where such rotations are precise, are observed. Another noteworthy aspect is that in some instances, the variance in the standard deviation becomes very small for the 100 samples, with no simply recognizable trend, for example, the 7000 shots point in the upper-left plot and the 6000-shot point in the lower-left plot. We suspect this may come from a periodic hardware calibration that is performed by IBM that was executed by the quantum devices between experiments.

For the next set of experiments we vary the distance between two points in 2D and 4D and plot the outputs in Fig. 3.6. The points we choose have the shape $(1, ..., 1)$ and $(x, ..., x)$ where we vary x to vary the distance between the points. The results of the 2D experiments show that the simulation and the physical device have similar outputs for $x \leq 5$, but for $x > 5$, the outputs from the simulation and real device start to diverge. In 4D, the effects of encoding show a stark difference between amplitude and angle embeddings. Interestingly, the simulation results for 4D amplitude embedding match very closely to the simulation for all tested values of x, more so even than in 2D. On the other hand, angle embedding performs worse in 4D than in 4D, where already for

$x = 5$ the difference between the simulation and the real device differs significantly. This extra noise is again likely because for more dimensions, the limited precision of the angle embedding is now applied for another two dimensions.

In the last set of experiments, we vary the dimension of the data to observe the limits to the number of features we can use and with what level of accuracy. We test up to 32 dimensions for amplitude embedding and up to 26 dimensions for angle embedding. We test vectors with the shape $(1, ..., 1)$ and second coordinates with the shape $(2, ..., 2), (3, ..., 3)$, and $(4, ..., 4)$. What we observe is indeed with 4 dimensions, the accuracy is at a reasonable level, but with 8 dimensions or more, the difference between simulation and real hardware loses too much accuracy to be reliable. In Tables 3.1 and 3.2, we show the percent difference between the simulation outputs and the results from the quantum computer for amplitude and angle embedding respectively.

Dim.	$x = 2$	$x = 3$	$x = 4$
2	66.44%	9.56%	13.69%
4	97.95%	30.29%	9.21%
8	128.25%	61.44%	8.42%
16	133.22%	54.98%	34.08%
32	110.86%	47.40%	24.72%

Table 3.1.: Amplitude Embedding: Percentage difference comparison between simulation and real data for the distance between data points of varying dimension and distance on IBMQ Sydney.

Dim.	$x = 2$	$x = 3$	$x = 4$
2	85.87%	61.87%	61.26%
4	144.24%	117.87%	98.77%
8	161.70%	131.13%	115.94%
16	156.84%	128.40%	113.8%
26	158.58%	129.59%	111.42%

Table 3.2.: Angle Embedding: Percentage difference comparison between simulation and real data for the distance between data points of varying dimension and distance on IBMQ Sydney.

3.1.6. Benchmarking Quantum Clustering

With distance estimation benchmarking results, we move on to benchmarking how well, even with noisy quantum devices, clustering can be performed. In this section, we test the ability to cluster in various dimensions and the number of clusters using synthetic data. To determine the accuracy of clustering using a quantum approach, we generated three types of synthetic data using 2 and 4 dimensions. The first two types of data that we generate are an *easy* data set, and a *hard* data set both with 4 clusters and 15 data points in each cluster. This totals 60 data points. The third data set we

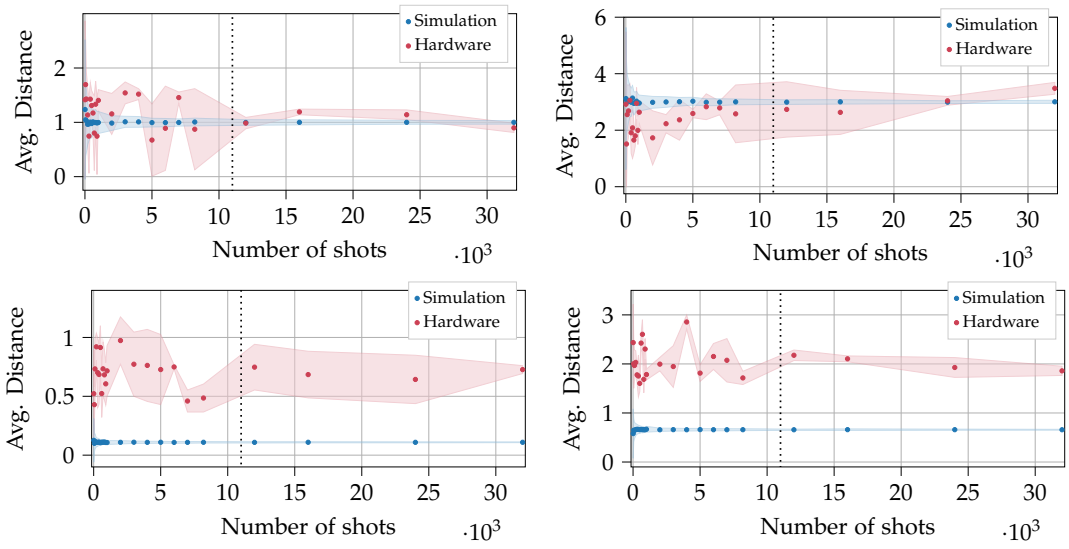


Figure 3.5.: The distance estimation for amplitude encoding (upper) and angle embedding (lower) of $(1,0)$ and $(1,1)$ in 2D (left) and $(1,0,0,0)$ $(1,1,1,1)$ in 4D (right). The circuit for measuring the distance between these two points is generated and ran a varying number of shots. Displayed is the average output of 100 trials with the standard deviation shaded around the average. The black dotted line indicates where the quantum hardware used switches from IBMQ Bogata on the left of the line to IBMQ Casablanca on the right.

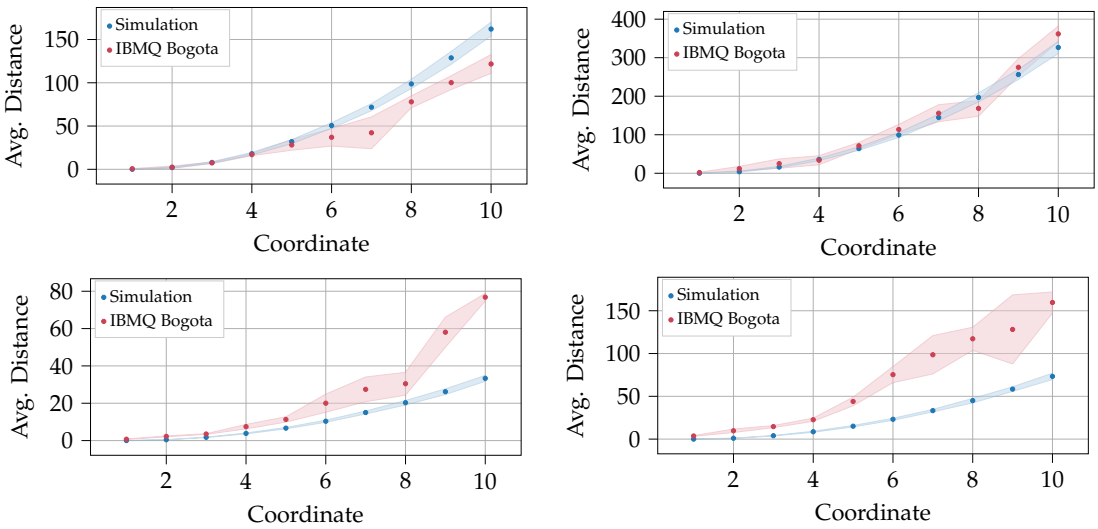


Figure 3.6.: Plots for varying the distance between points. The plots on the left are amplitude and angle embedding in two dimensions. The right plots are for amplitude and angle embedding in four dimensions. We run the experiments for a vector of shape $(1,...,1)$ for the base points and, $(x,...,x)$ for the varying point. We repeat the experiments 100 times with 2048 shots plotting the average with the standard deviation.

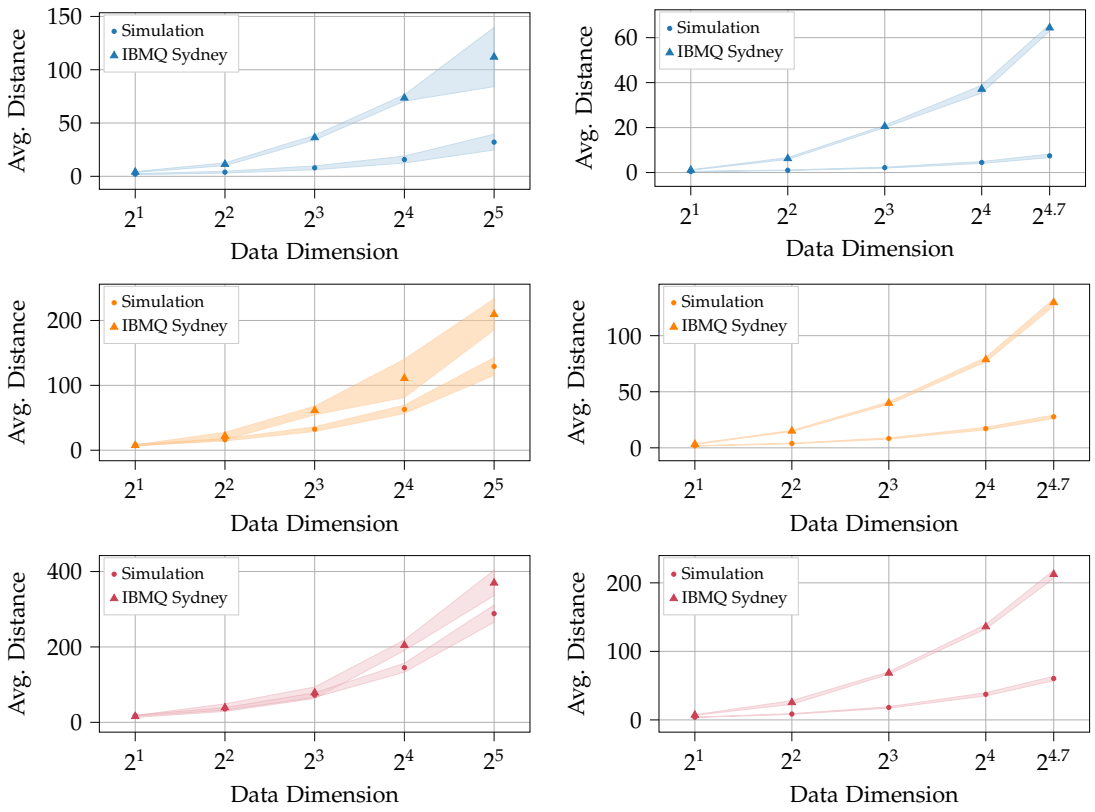


Figure 3.7.: Plots for varying the dimension of the data, using the vectors $(1, ..., 1)$, comparing $(2, ..., 2), (3, ..., 3), (4, ..., 4)$ respectively from top to bottom to benchmark distance estimation accuracy. We run the experiment 100 times with 2048 circuit shots and plot the average output and standard deviation of simulation after running on the IBMQ Sydney device for up to 32 dimensions with amplitude embedding (left) and 26 dimensions with angle embedding (right).

use has a variance between the *easy* and *hard* sets, but with 8 clusters and 14 points per cluster. The number of data points and clusters was selected to most easily work with the circuit-runner service, reaching the limits to how many circuits can be sent at once. The data is depicted in Fig. 3.8. In the case of the four clusters *easy* and *hard*, we analyze under the same conditions. We use both amplitude and angle embedding with a maximum of 5 iterations or until convergence of the centroid locations and run the three different approaches taken for distance estimation, using the outputs of the classical algorithm as the base truth. We use 8192 shots, for each experiment, using the option for measurement error mitigation in all cases. For these experiments, we use quantum devices with quantum volume 32. The results of the experiments for 2D are seen in Figs. 3.9. The results displayed are, on the left side, left column, the confusion matrices comparing the classical baseline to the quantum outputs using amplitude embedding, and the analogous for angle embedding on the right column. The results show that, in the simulation setting, the classical outputs are matched perfectly. For real hardware, the results do not match perfectly and indeed the results show a relatively low accuracy in the labeling for both embeddings using the real hardware. The results are similar for the hard data set as seen on the right side of Figs. 3.9.

For the 8-cluster data, we perform the same simulation steps, but for a maximum of 3 iterations with 8192 shots. In simulation, convergence is reached with 2 iterations with perfect labeling results. We test both 2D and 4D data sets. The 2D results are shown in Fig. 3.10. For 4D data, the results had very low accuracy, with essentially a random labeling and we neglect the results here for concern of text length.

These experiments motivate that a single quantum circuit with such low shots to generate a distance estimation is not enough, to improve the results of this experiment, likely we would need to average multiple distance estimations times to have consistency enough to provide accurate distance estimations. We do exactly this in the next section with real-world data to validate this hypothesis.

3.1.7. Applications for Energy Subgrid Clustering

The motivation for performing a detailed analysis on how well real quantum hardware performs on various distance-related metrics for high dimensional data is because, typically, many real-world datasets are non-trivial, high dimensional, and not well clusterable. In classical computing solutions involving unsupervised clustering, the problem of aggregating and grouping sets of objects usually involves some dimension reduction techniques and some type of domain knowledge to tune any machine learning algorithm used. To the best of the authors' knowledge, there is no existing work on a large dataset, non-trivial (e.g. input vector dimensional greater than 2), unsupervised clustering use cases using real quantum hardware. Since quantum machine learning and variations of clustering are touted as being a possible avenue for quantum advantage, we aim to employ the findings in the previous section in a real use-case that provides business value. In doing so, we propose an alternative k-means clustering algorithm to overcome some of the deficiencies revealed in the previous section, namely, the inability of the vanilla k-means algorithm to be able to handle classical input vector dimensions of more than 4.

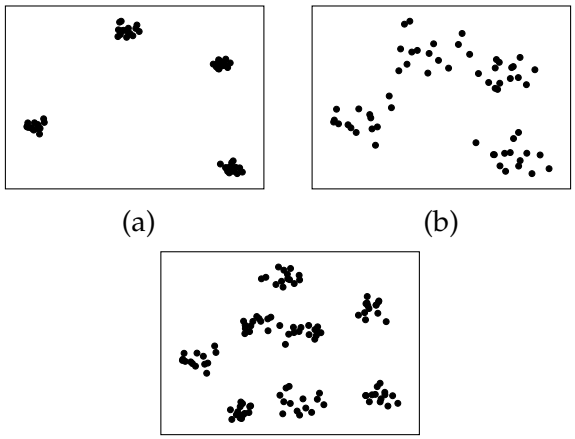


Figure 3.8.: Synthetic 2-dimensional data used for clustering. In (a), the data is tightly clustered with 4 clusters with 60 points in total which we consider an *easy* clustering. In (b) the data is more scattered with 4 clusters and 60 points total which we consider a *hard* clustering. In (c) are 8 clusters with 14 points per cluster for a total of 112 points.

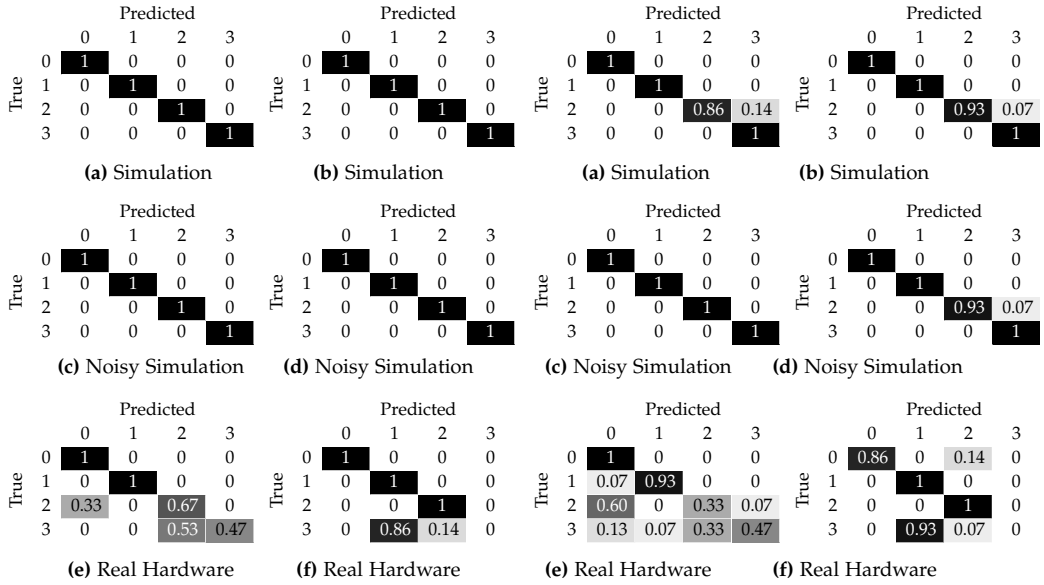


Figure 3.9.: The confusion matrices for the execution for clustering the *easy* (left) and *hard* data set (right) with 2 features using amplitude (left column) and angle (right column) embeddings. In (a) and (b) are the clustering outcome of the noiseless quantum simulation. In (c) and (d) are the outcomes when a noise model taken from parameters of the IBMQ Sydney device are used. In (e) and (f) are the results of running on the real IBMQ Sydney device. Each instance ran for a maximum of 5 iterations using the maximum number of shots 8192.

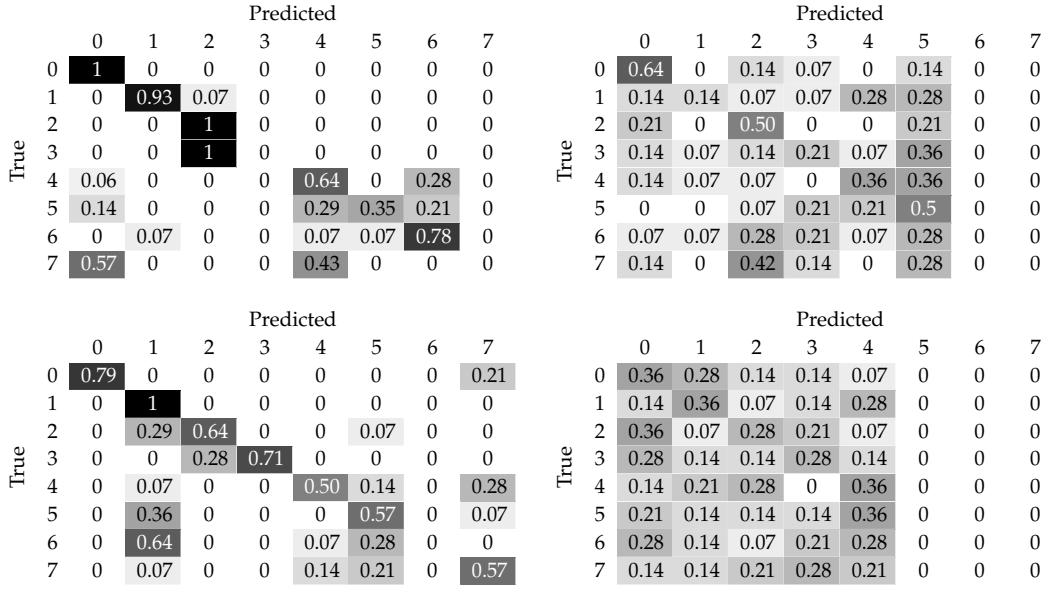


Figure 3.10.: Results from clustering data as depicted in Fig. 3.8(c) using amplitude encoding in the upper matrices and angle embedding in the lower. The left column uses 2D synthetic data and the right column uses 4D synthetic data. For the 2D data, we run the distance estimation circuits for 8192 shots on IBMQ Sydney. For the 4D data, the distance estimation circuits run for 2048 shots on IBMQ Bogota. The results are those of running 3 iterations of k-means, where in simulation, perfect results are produced with 2 iterations.

3.1.8. German Electricity Grid Data

In the energy operations sector, one major topic is of predictive maintenance. The ability to determine areas of the electrical grid that are susceptible to failing in some pre-determined timespan has many obvious benefits for customers downstream from any grid infrastructure which may fail. One possible approach to this problem is using data-driven analysis of different partitions of the full network grid to group and find similar types of subgrid assets by taking into account data features such as the amount of renewable and non-renewable electricity flowing through the grid subsection, the number of power lines within a subsection, and statistics about the ages of the assets contained in the subsection. Given such a collection of asset properties for electrical grid assets, we aim to employ unsupervised k-means clustering to classify the various subgrids.

The dataset consists of 81,350 low-voltage power lines from a Distribution System Operator (DSO) grid in Germany. Each power line has 7 numerical features as described in Table 3.3. Low-voltage subgrid networks are connected to high-voltage entry and exit points in the grid. For a given high-voltage transformer in the grid network, we collect the low-voltage lines which are part of its respective subnetwork and compute numerical features describing the entire subset of low-power lines. Specifically, for each subgrid, we compute: Number of Non-Powerline Assets, Total Number of Connected Assets, and then the minimum, maximum, and sum of each of the features listed in Table 3.3. There are 1,037 subgrids and therefore we have a final dataset of 1,037 feature vectors each of dimension 26.

Name	Unit
Conductor cross-section	cm^2
Operating Voltage	kV
Average Renewable Energy In-feed Load	MWh
Average Non-Renewable Energy In-feed Load	MWh
Number of Exits of Next Major Substation	#
Line Length	m
Sum MVA at closest HV exist	MVA

Table 3.3.: Description of the features for each powerline in the dataset.

3.1.9. Results

To cluster the 26-dimensional data created using the individual power line features (see Table 3.3) for all power lines in a given high voltage entry point, we first perform a preprocessing step to reduce the total feature vector dimension. To fit the data onto the quantum computers available for this work, we reduce the dimension to eight using Principal Component Analysis (PCA) which results in a 97.7% explained variance as well as a dataset with six dimensions using PCA which accounted for 91.4% of the variance. This second dataset was used for the angle embedding approach to fit in a 7-qubit quantum computer, the quantum computer topology we had the most access to

in this work.

With an initial classical analysis using the elbow method [102], the optimal number of clusters for this dataset was determined to be $k = 5$. From this dimension-reduced dataset of 1,087 points, we randomly selected 180 points to cluster, where 180 points allow us to send 900 circuits ($180 \cdot 5 = 900$) to IBM's cloud service in one job (an upper limit for some hardware). Important to any unsupervised clustering algorithm is the choice of initial centroid points. To ensure a quick convergence, and to reduce the number of quantum iterations, we ran the classical algorithm with a variety of random seeds such that convergence was reached within three iterations. The classical clustering results are depicted in Fig. 3.11(a), using t-Distributed Stochastic Neighbor Embedding (t-SNE) [103] on the high-dimensional data to generate a 2D projection. Using the initial centroids that achieved this, we then ran the quantum clustering experiment.

To validate the quantum approaches we used, we first compare the labeling output from noiseless simulation to the label outputs using the classical approach and then repeat the comparison running on real quantum devices. Because the classical approach converged in three iterations, we allow the quantum versions to run with a maximum of five iterations. In the simulation, the balanced accuracy of the experiments was 100% for amplitude embedding and 97.8% for angle embedding. Given that simulation produces high accuracy, we performed a series of tests on the quantum hardware.

Clustering using amplitude embedding

The first test we perform is to simply run the same logic as in the simulation. We use 12,000 shots per distance estimate and run the full clustering algorithm for five iterations. The clustering result using amplitude embedding is given in Fig. 3.11(b). The grid data is of relatively high dimension and the circuits to prepare the data are roughly 120 gate-depth with approximately 70 non-local gates for amplitude embedding. For angle embedding, the gate depth is expectantly shallower at approximately 86 but with roughly 103 non-local gates, depending on the randomization of the circuit transpilation step. With the level of noise occurring, five iterations do not improve the results, and indeed we speculate further iterations would not have led to improved results either. Here, our observation is that the labeling is essentially random due to the noise in the distance estimation circuits, never leading to a converging state.

Classification using amplitude encoding

As a second test, we implemented a pure nearest-neighbor classification application. We begin by training the model offline classically to determine optimal centroid locations, then, at runtime, we compute only the prediction step quantumly to determine which cluster test set data points belong to. Fig. 3.13(a) shows the accuracy results of the outcome, where we used 30,000 shots to estimate the distances using amplitude embedding in 8D. We see the majority of points were assigned to one class, similar to how the five iterations of clustering performed.

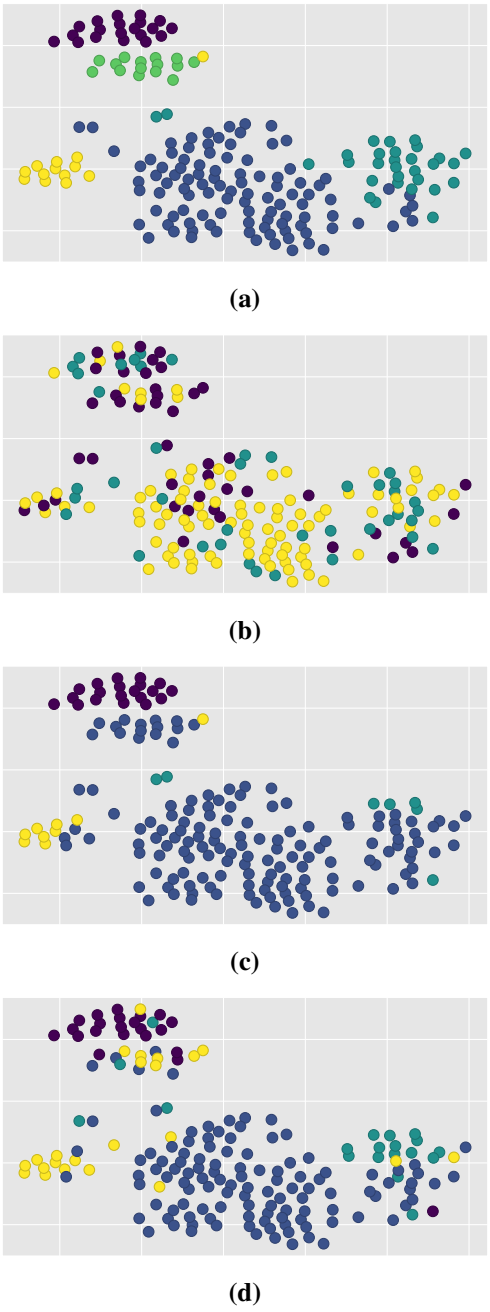


Figure 3.11.: (a) Classical clustering output after t-SNE is performed on the subgrid data. (b) Quantum labeling output using 8D amplitude embedding for five iterations with 12,000 shots per distance estimation on IBM Casablanca. (c) Quantum clustering output using the angle embedding split distance estimation (3.1.12) of the subgrid data. We decomposed the distance estimation to be one estimate per circuit, for five iterations with 12,000 shots per distance estimation on IBM Perth. (d) Quantum clustering using amplitude embedding split distance estimation (3.1.12) using a parallel execution process as described (c).

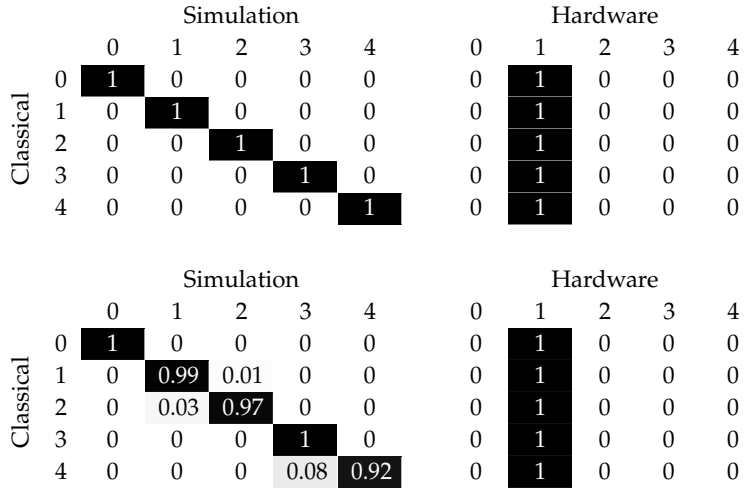


Figure 3.12.: Unsupervised clustering results compared to the classical outputs using energy grid data with 5 clusters on IBMQ Casablanca using 12,000 shots for amplitude embedding and 2,048 for angle embedding.

Distance estimation with vector subspace parallelization

So far, our approach to distance estimation has been to use the approach as stated in [84], however, the accuracy in practice using this approach has thus far been relatively low. From the benchmarking section, the highest accuracy was seen in the two-dimensional data experiments. Using this as motivation, we propose a new technique of parallelizing the distance calculation for high-dimensional vectors by using distances between two-dimensional subspaces of the full feature vectors.

Given input data vectors $a := (a_1, a_2, ..., a_n)$ and $b := (b_1, b_2, ..., b_n)$, the distance between them can be decomposed as

$$\begin{aligned} d(a,b) = d(a_{1,2}, b_{1,2}) + d(a_{3,4}, b_{3,4}) \\ + \cdots + d(a_{n-1,n}, b_{n-1,n}), \end{aligned} \tag{3.1.12}$$

where $a_{i,j} = \mathcal{P}_{i,j}(a)$ and $b_{i,j} = \mathcal{P}_{i,j}(b)$ are projections of the respective vectors to the (i,j)-th vector subspace. The circuit for this parallel distance estimate using angle embedding is depicted in Fig. 3.14.

This approach has various benefits in terms of mitigating noise. Firstly, it uses only low-dimensional projections. In this case, we use two-dimensional projections, aligned with our benchmarking results, but as hardware improves, we can extend this to larger dimensions to reduce the number of total independent measurements until we can eventually use the entire vector. Next, these low-dimensional circuits will be in general shallower and thinner, which will improve the accuracy and reduce the computation time, allowing for more shots within the same execution timespan. Because in some cases we observed a large standard deviation, with shorter execution time, one can also execute the circuit many times to produce an average distance estimate within the same period, mitigating Gaussian noise in the system.

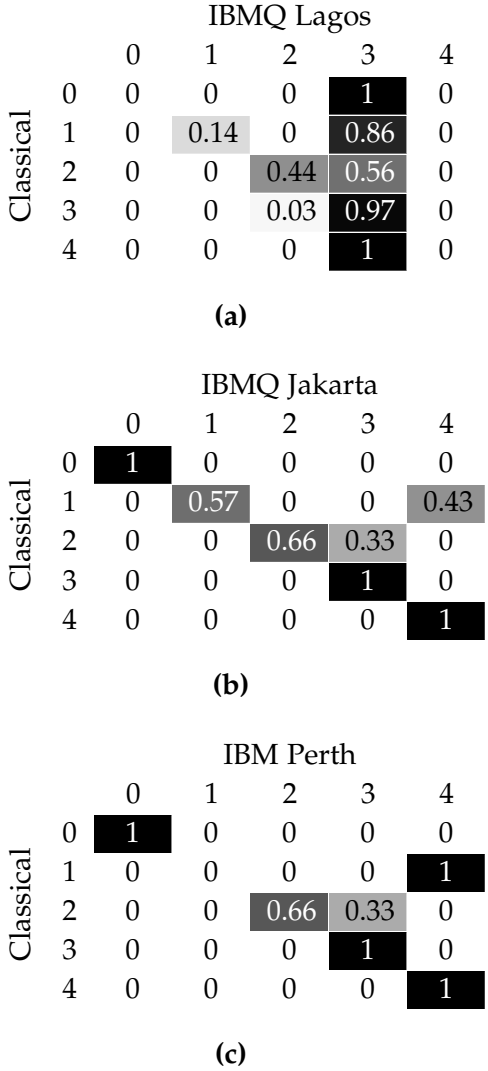


Figure 3.13.: The results of classifying the test set of 60 data points using a nearest-neighbor prediction. In (a), we use amplitude encoding in 8 dimensions and 30,000 shots per distance estimation. The balanced accuracy, in this case, is 17.0% and the raw accuracy 46.7%. Weighted precision cannot be computed since some classes are empty. In (b) we repeat the classification using 15,000 shots using the divided distance estimation in four 2D estimates, using an averaged of five estimates per distance. The results have a balanced accuracy of 84.8%, a raw accuracy of 90.0%, and a weighted precision score of 92.1%. In (c), the classification is done using the vector subspace parallelization circuit which uses two swap tests per circuit. This approach reduces the total number of circuits by 50%. The results improved significantly to have a balanced accuracy of 73.3%, and a raw accuracy of 83.3%.

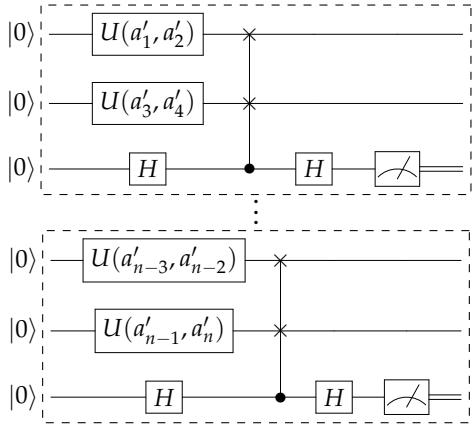


Figure 3.14.: Circuit for processing multiple distances in one circuit using angle embedding with swap tests. The dashed boxes separate each independent swap test.

Because the distance estimation circuits are indeed thinner, we can load multiple circuits into one QPU proportional to the number of qubits. For example, because the swap test with angle embedding in 2D uses three qubits per swap test, we can load two distance calculations at a time in a seven-qubit quantum computer, reducing the number of total circuits to execute by 50%. This approach could be generalized to contain as many swap-tests as there are (the floor of) one-third the number of qubits, which could in turn result in again using one circuit for distance estimation, simply with a modified pre- and post-processing step. These benefits make this much more NISQ compatible than performing the distance estimation with all dimensions considered at once.

After verifying this approach produced accurate results in simulation, to test how well it mitigates the effects of noise in the classification task, we ran the circuit implementing (3.1.12) using two approaches. For the first approach, we used amplitude encoding and executed each distance estimation circuit independently five times, averaging the results and using the average as the distance estimate. We used 15,000 shots per execution and since the circuit uses four qubits, we could fit just one circuit at a time on the 7-qubit device. The confusion matrix of the results is in Fig. 3.13(b), showing a vast improvement.

For the second approach, we perform classification again, now using angle embedding, but in this case, since just three qubits per swap test are required, we could load two distance estimates in parallel into the 7-qubit device. Using 15,000 shots with, in this case, two repetitions per circuit, we again use the average for the estimate. The results of the classification are shown in Fig. 3.13(c). Again, we see a strong improvement for the classification problem over the standard method, increasing the balanced accuracy from 17% in the standard approach to 73.3% using this novel approach, a difference of 56.3%.

Clustering with vector subspace parallelization

Given the promising results from the classification task using the vector subspace parallelization, we again perform the full clustering algorithm using the angle embedding approach. We use the distance estimation (3.1.12) with 12,000 shots and one repetition on the IBM Perth machine with a total of five iterations for the clustering algorithm. The clustering results are shown in Fig. 3.11(c). Although the labels were reduced to three classes, two fewer than in the classical algorithm, we see a much clearer separation of the classes in comparison to using amplitude encoding for all eight dimensions.

3.1.10. Conclusion and Discussion

In this work, we thoroughly investigated the potential of using quantum k-means clustering practically on current NISQ quantum hardware. In terms of distance estimation comparison between classical and quantum distance calculations, we observed a high level of difference between simulation and running on physical devices—especially comparing the distance estimation results using angle embeddings. The results which used batched job submission via `Qiskit Runtime` showed to vastly improve performance, allowing for more circuit executions, improving reliability, as well as drastic speed improvements when dealing with large datasets due to the reduction in job-queuing time.

The best k-means clustering results observed were from clustering datasets of 2-dimensional data points. When we increased the number of clusters from four to a more complex scenario of eight clusters and changed the input vector dimension from two to four, the results worsened. We experimented with an industrial unsupervised learning problem, labeling high-dimensional energy grid data using k-means clustering. Using the state-of-the-art approach, the clustering and classification results proved inaccurate when executed over real hardware. When we changed the distance metric and used our vector subspace parallelization approach, we saw a significant improvement in both our classification and clustering experiments. For amplitude embeddings, the balanced accuracy of the classification went from 17% using the standard approach to 84.8% with this novel distance estimation method. With angle embedding, loading two swap-tests into one circuit to execute in parallel, albeit an overall wider circuit, proved to also have a large performance improvement over the amplitude encoding approach, with a balanced classification accuracy of 73.3% and raw accuracy of 83.3%.

This work provides a first step into quantum clustering for practical, industrial use cases, but still, there are questions to be answered. Future work will be to consider other clustering algorithms such as k-medoids which use alternative distance metrics, considering other quantum approaches for distance estimation. Indeed, many algorithms require a distance calculation step, and so benchmarking their quantum performance leaves many possibilities for future work.

Although it is well known that quantum computing is in its early stages of development, it is important to investigate what boundaries exist in relation to non-trivial problems that move beyond fundamental algorithm proofs-of-concept. Clustering, and particularly distance estimation, is widely used in various industrial applications. With

this work, we have tested a large set of experiments that can be performed on the quantum cloud using only the core features of the platform.

Quantum technology is continuously and rapidly improving, and we expect that as NISQ-era quantum computers mature, these types of analysis and industry-driven use-case studies will continue to be necessary to provide valuable insight into how they will be used for real-life applications.

3.2. Distributed Quantum Computing

Section based on the article: "Quantum Algorithms and Simulation for Parallel and Distributed Quantum Computing"

Scaling quantum computers up to levels where practical quantum algorithms can be executed will require several technological breakthroughs. In the present state of technology, scaling quantum computers past the 100-qubit mark has proven challenging [104]. Even when quantum computers can support a large number of qubits in a single system, if current error correction methods like surface codes are used, the number of control signals required to perform error correction will scale with the number of qubits, potentially bottle-necking logical instructions for an algorithm's execution[7]. To overcome these obstacles, a potential solution is to instead create smaller-scale quantum computers and interlink them using a quantum network to perform quantum algorithms over a distributed system. The benefit of using smaller, interlinked quantum processors is the ability to perform larger quantum circuits on more robust and controllable quantum processors albeit with the added—potentially easier— problem of using distribution methods. When one can use networked quantum computers, an additional ability to use parallelism in algorithm design is enabled.

When moving from monolithic to distributed quantum computers, a variety of challenges arise. Indeed, there are many technological challenges to overcome in building distributed quantum computers. A naturally arising problem to consider in this perspective is performing two-qubit operations between qubits that are physically separated between two quantum computers. To perform two-qubit operations with monolithic quantum technologies, generally, the two qubits are physically near each other, and if not, swap gates are applied to bring them near enough, known as the qubit routing problem [105]. On the other hand, for two-qubit operations between distributed qubits, one needs a new technique for transporting the control information between devices. Possible options are to physically transmit qubits via a potentially noisy and lossy medium [106], using quantum teleportation [107, 108], transferring control information to a flying qubit [109, 110], or using the method introduced in [111] using one entangled pair and two bits of classical communication as seen in Fig. 3.16.

Once a method of performing non-local two-qubit gates is selected, quantum circuits designed for monolithic systems need to then be remapped to a logically equivalent distributed version. To perform the remapping, one starts with the topology of the networked quantum computers, each with its own quantum processor chip structures. A monolithic circuit is converted such that any multi-qubit operation involving qubits

located on different processors is replaced with a logically equivalent set of instructions orchestrating the additional tasks needed for the non-local operation. This remapping problem has been addressed in a variety of ways [112–117], but until distributed quantum computing becomes more standardized, the most applicable method for generating and optimizing distributed circuits remains an open problem.

The next problem arising is how to design and develop a control system for a distributed system of quantum computers. Already a step in this direction is the concept of cloud quantum computing which takes user input—usually as a circuit—and a software layer converts the input into control instructions for a single quantum computer [100, 118]. The quantum computer performs the computation, and the results are sent back to the user via a communication network. For a distributed system of quantum computers, additional network connections are needed between the quantum computers. Moreover, the connections cannot simply be classical channels, but quantum channels will be needed for either distributing entanglement or moving data-containing qubits. Networked control systems for classical distributed systems have been developed in various scenarios [119], for example in GPU clusters [120], but a key problem that is not as critical for classical systems for computing is that the quantum computers need to be highly time-synchronized to perform joint measurements, for one. It is therefore a unique problem to design networked control systems for distributed quantum computers. Proposals addressing such control systems are found in [112, 121].

Finally, once the ability to perform distributed quantum algorithms is enabled, one can then start to consider the various quantum algorithms that can benefit from being distributed and parallelized. Such examples have been considered such as distributed Shor's algorithm [111], Quantum Phase Estimation (QPE) [122], and accelerated Variational Quantum Eigensolver (VQE) [112]. Further, a mathematical framework for expressing and analyzing distributed quantum algorithms has been developed in [123]. Now that the hardware technology is beginning to catch up with the theory, what remains is to better understand what advantages exist if any, especially while considering the cost of execution in a distributed setting.

In this work, we investigate two angles for distributed quantum computing. We consider firstly a formalization of parallel and distributed quantum programs and consider a collection of quantum algorithms fitting this formalization. Next, we introduce a novel software simulation tool for simulating distributed quantum algorithms called Interlin-q. Interlin-q is a Python library built on top of QuNetSim [56]—a quantum network simulator—which generates and simulates the control steps needed in an asynchronous setting to simulate distributed quantum algorithms. The overall goal of the platform is to provide a tool for validating algorithms for distributing quantum circuits and testing control systems. In addition, one can use Interlin-q to simulate parallel and distributed algorithms to then benchmark the approaches for their distribution and parallelization efficiency. In this thesis, we provide an overview of the software library in its current state and some demonstrations. Overall, interlinking quantum computers to perform distributed quantum algorithms will inevitably be an important part of quantum computing in the coming future. This thesis aims to shed light on the open problems and foreseeable benefits of distributed quantum computing, an increasingly important topic for the future of quantum computing.

When moving from monolithic to distributed quantum computers a variety of challenges arise. Indeed, there are many technological challenges to overcome in building distributed quantum computers. A naturally arising problem to consider in this perspective is performing two-qubit operations between qubits that are physically separated between two quantum computers. To perform two-qubit operations with monolithic quantum technologies, generally, the two qubits are physically near each other, whereas for two-qubit operations between distributed qubits, one needs a new technique for transporting the control information. Possible options are to physically transmit qubits via a potentially noisy and lossy medium, using quantum teleportation, transferring control information to a flying qubit as in the recent experiment [109], or using the method introduced in [111] using one entangled pair and two bits of classical communication as seen in Fig. 3.16.

With the ability to perform distributed two-qubit gates, quantum circuits designed for monolithic systems need to be remapped to the logically equivalent distributed version. Additionally, the control system for a monolithic system needs to be modified to support a network interface to communicate control instructions within the distributed network. The ability to synchronize and control a distributed system of quantum computers to orchestrate them on a very precise schedule will also be required. Proposals to solve these problems have been developed [112–114, 121], where communication protocols and distributed circuit compilation are considered. With the ability to generate equivalent non-local circuits, one can then start to consider quantum algorithms that can benefit from being distributed and parallelized. Such examples of distributed Shor's algorithm [111], Quantum Phase Estimation (QPE) [122], and α-VQE [112] have been considered.

In this thesis, we investigate two angles for distributed quantum computing. We consider firstly a formalization of parallel and distributed quantum programs and consider a collection of quantum algorithms fitting this formalization, all gaining a potential run-time speed up when parallelized. Next, we introduce a novel simulation tool for simulating distributed quantum algorithms. We introduce our software framework Interlin-q which enables the simulation of distributed algorithms. Interlin-q is a Python library built on top of QuNetSim [56], which generates and simulates the control steps needed in an asynchronous setting to simulate distributed quantum algorithms. We provide an overview of the software library in its current state and a demonstration overview. Interlinking quantum computers to perform distributed quantum algorithms will inevitably be an important part of quantum computing in the coming future. This thesis aims to shed light on the open challenges and foreseeable benefits of distributed quantum computing, an increasingly important topic currently seeing relatively little research attention.

3.3. Monolithic to Distributed Algorithms

Section based on articles: "Distributed Quantum Computing and Network Control for Accelerated VQE" and "Quantum Algorithms and Simulation for Parallel and Distributed Quantum Computing"

To start our investigation of distributed quantum algorithms, we generalize the concept of mapping monolithic quantum algorithms to distributed quantum programs and scheduling them for execution. Executing a distributed quantum algorithm on a distributed quantum computer has general preparation and execution stages: 1) Allocate logical qubits within the network of quantum computers; 2) Remap circuits for the possibly distributed qubit assignment; 3) Generate a schedule for the control operations; 4) Distribute and execute the schedule; and 5) Merge the outputs. Some quantum algorithms, which we investigate in the next section, have a particular structure that allows them to gain large "horizontal" speedups when parallelized, whereas other quantum algorithms requiring many logical qubits can more readily be executed on nearer-term quantum computers via a distributed quantum computer. To model this staged process of preparation and execution, we start with a QPU structure as a collection of integers $Q = [q_1, ..., q_k]$ representing a network of k QPUs where each QPU i has $q_i \in \mathbb{N}$ logical qubits. In this model, it is implied that the quantum network topology is completely connected and entanglement units are created during runtime. With this, we define a quantum parallel program.

Definition 17 (Parallel Program)**.** *A program P is the instruction set needed to perform a monolithic execution of a quantum circuit including the logical circuit and the number of times to repeat the execution of the circuit. A schedule $S(i)$ is a mapping from an execution-round number i to sets of integers, where $|S(i)|$ is always the number of QPUs in the network. The k-th set of $S(i)$ represents the programs $\mathcal{P}_i \subset \{P_j\}_{j=1}^n$, where there are n programs total to run, executing at time i on QPU k where two distinct sets in $S(i)$ are not necessarily disjoint. A collection of programs $\{P_j\}_{j=1}^n$, a function $M : O^n \mapsto O$ for O the output of a program which acts as a central merging function, and a schedule forms a parallel program $\mathcal{P} = \{\{P_1, ..., P_n\}, S(i), M\}$.*

Definition 18 (Distributed Program)**.** *Given QPUs $Q = [q_1, ..., q_k]$, a distributed program dP is a program P where the circuit execution instructions of P are assigned to qubits from multiple distinct QPUs from Q. In this framework, it implies there exists an i where there are at least two distinct sets both containing P.*

To generate $\mathcal{P}$, the collection of programs and schedule, Algorithm 8 is used. Input to Algorithm 8 is 1) The specifications of the distributed quantum computers $Q = [q_1, ..., q_n]$; 2) The circuit input to the program with width w, that is, the number of qubits simultaneously needed to run the circuit; 3) An algorithm $\mathcal{A}$ which takes Q as input and determines an allocation for w logical qubits or determines no allocation exists; 4) A collection of monolithic programs $\{P_i\}_{i=1}^n$. The output of the algorithm is a schedule for executing a distributed program $\{\{dP_i\}_{i=1}^n, S(i), M\}$. In Fig. 3.15 is a depiction of how such a system could perform.

Example 1. *Let $\{P_1, ..., P_{10}\}$ be a collection of programs that run circuits with width $w = 4$ and $Q = [10, 10]$. If $\mathcal{A}$ is an algorithm that greedily allocates qubits, then the output of Algorithm 8 is: $S(0) = \{\{1, 2, 3\}, \{3, 4, 5\}\}$, $S(1) = \{\{6, 7, 8\}, \{8, 9, 10\}\}$ and $\{dP_1, ..., dP_{10}\}$, where dP_3 and dP_8 are distributed between the two QPUs and the other programs run monolithically.*

To generate $\mathcal{P}$, the collection of programs and schedule, Algorithm 8 is used. Input to Algorithm 8 is 1) The specifications of the distributed quantum computers $Q =$

Algorithm 8 Distributed Quantum Algorithm Scheduler

Input: QPUs $Q = [q_1, q_2, ..., q_k]$, w the circuit width, qubit allocation algorithm $\mathcal{A}$, programs $\{P_1, ..., P_n\}$. Assume $\forall i \leq k, w \leq q_i$.
Output: $\mathcal{P} = \{\{dP_1, ..., dP_n\}, S(i)\}$, dP_j the distributed program for circuit execution j, $S(i)$ the schedule for r rounds.

$a \leftarrow 0; i \leftarrow 0; dP \leftarrow \{\}; A \leftarrow \{\};$
for circuit $c \leq n$ **do**
  Allocate w qubits within current Q with $\mathcal{A}$
  if an allocation exists **then**
    $A \Leftarrow$ allocation ▷ Append the allocation to A
    reduce the available qubits in Q based on allocation
    $a \leftarrow a + 1$
  else if no allocation exists or $c = n$ **then**
    Use allocations A to distribute a circuits
    $dP \Leftarrow$ Generate a distributed programs
    $S(i) \leftarrow \{c - a, ..., c\}$
    Reset Q; $A \leftarrow \{\}$; $a \leftarrow 0$; $i \leftarrow i + 1$;

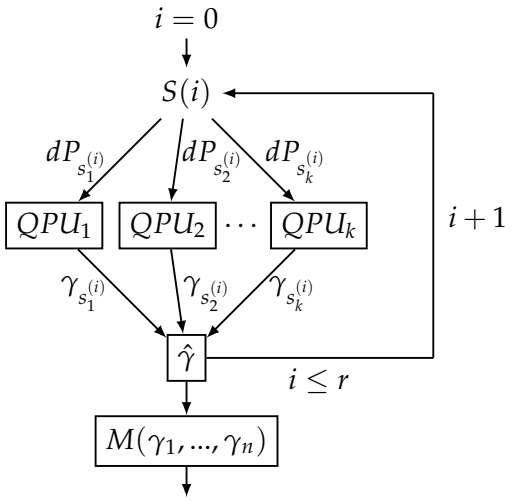


Figure 3.15.: A depiction executing a parallel program. The system starts at time $i = 0$, loading the programs specified by $S(i)$ to the respective QPUs until all r rounds are run. The outputs of the distributed programs are accumulated in an output vector $\hat{\gamma}$ during execution. Finally, M maps the collection of n outputs $\hat{\gamma}$ to a single output.

$[q_1, ..., q_n]$; 2) The circuit input to the program with width w, that is, the number of qubits simultaneously needed to run the circuit; 3) An algorithm $\mathcal{A}$ which takes Q as input and determines an allocation for w logical qubits or determines no allocation exists; 4) A collection of monolithic programs $\{P_i\}_{i=1}^{n}$. The output of the algorithm is a schedule for executing a distributed program $\{\{dP_i\}_{i=1}^{n}, S(i), M\}$. In Fig. 3.15 is a depiction of how such a system could perform.

Iterative quantum algorithms mapped to this model and scheduled using Algorithm 8 stand to face the same "horizontal speedup" as mentioned—a run-time speedup achieved by allocating more quantum processors to run in parallel. Influencing this speedup is the algorithm used to allocate qubits—$\mathcal{A}$ in Algorithm 8—for distributed processing. The choice of algorithm that solves this problem can come in a variety of flavors. For example, an allocation algorithm that simply chooses qubit allocations randomly will likely introduce more non-local gates, potentially diminishing potential speedups due to the needed additional logic, whereas one which considers the topology and connectivity of the quantum processor can minimize the number of non-local gates. Alternatively, [124] addresses qubit allocation to reduce the circuit width in a circuit using a technique called "circuit cutting" to run parts of a circuit independently and then uses classical post-processing to combine outputs. Their algorithm further aims to minimize the classical post-processing overhead. One can use the technique to define parallel programs to then execute the overall circuit over a cluster of QPUs.

With an optimal allocation algorithm, the speedup of the parallelization for the algorithms we investigate is not found by a reduction in algorithm complexity, but from running multiple iterations of an algorithm simultaneously reducing the run-time of execution. This type of speedup is commonly defined as the ratio between the run-time of one process running an algorithm and the run-time of p parallel processes running a parallelized version [125]. We also note that in classical distributed computing, the concept known as Amdahl's law is used to predict the theoretical speedup [126, 127]. The law predicts that eventually, the communication latency between many processors will diminish the reduced runtime of parallel processing. Indeed this applies to distributed quantum computing as well, but still, there are advantages to be gained as analyzed in [128].

Future work will require a deeper investigation into how much of a horizontal speedup can be gained in the purely quantum setting, and which parameters influence the speedup. One of the parameters that will play a large role will be—as with classical distributed computing—the topology of the network, but indeed some parameters will exist only in the quantum setting for distributing algorithms. We plan to further investigate the effects of two such parameters: 1) The quality of created entanglement and 2) Entanglement distribution protocols; each of which will affect the performance of non-local control gates. If the entanglement generation rate is low—which could be the case when using deterministic entanglement generation (roughly in the ~ 100 ms regime [66])—then there could be long waiting times during execution. Moreover, when the quality of the entanglement is low (but high enough to be useful), many repetitions of the algorithm could be needed to produce meaningful results. A full investigation will be necessary especially for developing optimized distributed circuit compilation algorithms.

3.3.1. Parallel and Distributed Quantum Algorithms

In this section we describe some examples of quantum algorithms that can be mapped to the parallelized model from the previous section, and hence can benefit from horizontal speedup. The property that each of the following algorithms has in common is the quantum part of the algorithm can be split up across multiple QPUs to run in parallel and the classical outputs can then be merged to produce the same result as if the quantum part was instead run iteratively. The types of algorithms use techniques like output counting or have linear properties that distribute straightforwardly. We investigate some such examples and explore some of the expected benefits and disadvantages.

Variational Quantum Eigensolver

Computing the eigenvalues of certain quantum operators can be challenging for classical computers due to the exponential scaling in the dimensions of the operators with the increase in the number of quantum states of the system. QPE allows one to compute such eigenvalues in a much more efficient manner but requires a coherent fully-connected quantum computer to produce good estimates. Consequently, the Variational Quantum Eigensolver [129] (VQE) was proposed as a low-depth alternative, using a hybrid model containing classical optimization and quantum computing. As per the name, VQE belongs to the family of Variational Quantum Algorithms (VQA) [130], a group of hybrid algorithms that include a quantum circuit as a subroutine. VQE uses a classical computer to fine-tune the parameters of the “ansatz" preparation circuit. In VQE, by tuning the parameters to the ansatz circuit one can minimize an expectation value and use this as an estimate for the minimum eigenvalue. The algorithm is built on the fact that certain Hamiltonian operators can be decomposed into a polynomial number of terms of simpler Pauli operators. As a result, the evaluation of the expectation value of such Hamiltonians reduces to a linear combination of the expectation values of these simpler operators. With this, one can simply measure the different qubits as per the observables in each term to obtain the term’s expectation value in constant time and then recombine to find an overall estimate.

In its standard form, VQE iteratively performs this, but it can benefit from using a cluster of quantum processors in two different ways. The general workflow would be to dispatch some terms as well as the respective parameters to each quantum processor, which would then compute the expectation value of the terms, and then the dispatching node would aggregate the results from each of the different processors. After carrying out the classical optimization step generating the new parameters for the ansatz, the dispatcher would then send the new parameters to the quantum processors to repeat the process. The second advantage comes from the fact that the Hamiltonian governing a molecule requires more quantum systems to simulate as the complexity of the molecule increases. By using a cluster of interlinked quantum computers, one can simulate larger Hamiltonians using the interlinked smaller quantum computers. Indeed, with an approximately 48 qubit Hamiltonian, it is predicted to be infeasible for a classical computer to simulate [131], which could be achievable with an interlinked cluster of quantum computers in the future.

We now frame this algorithm in the setting of the previous section. For a Hamiltonian $H = \sum_{i=1}^{n} c_i L_i$, where each $c_i \in \mathbb{R}$ and $L_i \in \{I, \sigma_x, \sigma_y, \sigma_z\}^{\otimes w}$, is a tensor product of w Pauli matrices (or identity), we can form a collection of programs $\{P_i\}_{i=1}^{n}$ where each P_i is the combined w width ansatz preparation circuit, generating $|\psi_j\rangle$ for the jth iteration, prepended before the respective circuit for L_i with N repetitions. The merging function M is simply to reassemble the linear combination, where M applies the respective the coefficient c_i, computing the estimate for $\sum_{i=1}^{n} c_i \langle\psi_j|L_i|\psi_j\rangle$. This type of parallelization has been investigated in depth in [132], showing up to a 100-fold improvement in algorithm execution efficiency in experiment in comparison to iterative methods. Indeed many variational quantum algorithms have this same structure [130, 132] and can be parallelized similarly, making it more feasible for executing the class of algorithms on near-term devices.

Low-Depth Quantum Amplitude Estimation

Already considered in 2002 by Bassard et al. [133], Quantum Amplitude Estimation (QAE) remains one of the fundamental algorithms for quantum computing, as it adds, for one, a significant performance speed-up for Monte-Carlo methods [134]. An issue to overcome to use QAE with near-term quantum computers is to greatly limit the circuit depth. In its original form, QAE uses a combination of QPE and Grover's search [86], where QPE, with no additional assumptions, uses circuits that deepen proportionally to the inverse of the precision [133]. Moreover, QPE requires an application of the inverse-QFT algorithm requiring a high-depth and highly-connected quantum processor. To overcome these issues, proposals for low-depth, QFT-free QAE have been proposed [135–137].

From these approaches, we focus on the algorithm called the "Power Law Amplitude Estimation" (PLAE) algorithm proposed in [136, Algorithm 2.1]. PLAE works by using a maximum likelihood estimation routine where for each number of queries $m_k \in \mathcal{K} \equiv \{\lfloor k^{(1-\beta)/2\beta} \rfloor : k \leq K\}$—with K bounded above by a constant that grows depending on the desired accuracy and $\beta \in (0, 1]$—a circuit making m_k sequential oracle calls is executed on a quantum computer. For each m_k, the circuit making m_k queries executes N times, measuring the output of a single qubit, essentially performing tomography. Once all of the K circuits execute, a Bayesian update step is performed iteratively on the K statistics outputs.

In the framework of the previous section, there is a clear parallelization to make for this problem. We can define a program P_k for each $k \leq K$ to be the oracle circuit of width w with m_k oracle queries and N repetitions. The output γ_k of P_k is the accumulated statistics of performing m_k oracle queries. The Bayesian update task is used for the merging function M. Once all $\{\gamma_k\}_{k=1}^{K}$ are collected, a phase estimate is made based on the original algorithm. In this way, one can split the load of executing the K circuits across multiple quantum computers, thereby gaining a horizontal speedup. A further parallelization that can be made is to duplicate programs P_k on multiple QPUs, using the same oracle query but dividing the number of circuit repetitions across the QPUs to then merge the counting statistics for each oracle type. Algorithms using Bayesian update methods via counting as with this version of QAE have been proposed in other

modified quantum algorithms [138], and further investigation for this algorithm class could prove fruitful.

Quantum k-Means Clustering

Clustering data into groups based on the properties of the data can be used to find correlations between the data features. k-Means clustering is an unsupervised machine learning algorithm used to perform such clustering [48]. The k-Means algorithm takes as input a collection of unlabeled data, or feature vectors, and outputs k clusters, where in each cluster are the data points that minimize the distance to a computed centroid point. The algorithm runs for several iterations, improving the centroid locations to minimize the average distance between the points in the cluster at each step. A distance metric is used to determine how far apart two data points are from each other. Classically, the usual method for measuring the distance is to simply take the Euclidean distance. For feature vectors of length N, computing Euclidean distance requires $O(N)$ steps. Using the quantum encoding known as amplitude encoding, one can encode N length vectors into $O(\log_2 N)$ qubits, an exponential decrease for encoding, assuming one can load quantum states into a quantum random access memory [84]. With this encoding, one can perform a swap test to compute an estimate for the Euclidean distance between two feature vectors. The swap test performs proportionally to the number of qubits used in the encoding and can lead to—in theory—an exponential decrease in the number of operations used to compute distance. Quantum k-means clustering is especially interesting as it is suitable for near-term quantum devices [90, 139].

Because each feature vector is compared to each of the k clusters based on the algorithm of [48], n distance estimates are made for each of the k centroids. To parallelize this we can consider programs $\{P_{ij}\}_{i=1,j=1}^{n,k}$ where each program computes the distance between feature vector i and centroid j. The circuit for each P_{ij} is the one described in [99], which loads two feature vectors using amplitude encoding and an additional ancilla qubit for performing the swap test. The merging function M collects the outputs of $i \cdot j$ programs grouping the circuit outputs in i vectors of length j such that the closest centroid can be determined. With this, one can then update the centroid positions classically and repeat the process until convergence is reached, or a maximum number of iterations are performed.

For a purely parallel version of k-Means clustering, the horizontal speedup will scale linearly according to the number of quantum processors until the scale of connectivity comes into play according to Amdahl's law. When moving to the distributed setting, where the number of features cannot be encoded in a single QPU, it becomes important for determining the overall run time to consider how the classical data is encoded in the quantum computer. Indeed, depending on how one performs encoding it could be that no quantum advantage is achieved for clustering [88]. If a standard quantum state preparation algorithm is used to perform amplitude encoding across a distributed system of quantum computers, then an exponential number of control gates will be used for state preparation with the number of features, but only a logarithmic number of control gates for performing the swap-test. Alternatively, with an angle

encoding, only a linear number of control gates are needed for state preparation, but also a linear number of control gates for the swap test, hence no quantum advantage. Moreover, the more control gates needed across a distributed system will result in more classical communication and entanglement generation. The full effect of quantum state preparation across distributed systems will be of interest for future work, especially adapting novel preparation methods as in [140] for distributed systems.

3.3.2. Decomposing α-VQE

As an example of using this approach to parallelize and distribute a quantum algorithm explicitly, we take the α-VQE [138] and remap it for a distributed architecture. A problem to overcome when dealing with near-term quantum computing devices is that the ability to run deep circuits is greatly reduced due to the low coherence time of qubit systems without error correction. A classical-quantum hybrid class of algorithms called "variational quantum algorithms" allows running reduced depth circuits performing some of the algorithms on near-term quantum hardware and some on classical hardware. In particular, the variational quantum eigensolver (VQE) algorithm is a variational hybrid-quantum algorithm that can be used to find the minimum eigenvalue of a chemical Hamiltonian. It uses a quantum portion of the hardware to estimate the eigenvalues for a particular Ansatz of Pauli operations combining to form the Hamiltonian. VQE uses the quantum system to determine an expectation value and these expectation values are then combined to find an expectation value of the full Hamiltonian [129].

Using classical optimization techniques, various Ansätze – plural of Ansatz – are prepared with the goal of finding an estimate of the eigenstate with the lowest eigenvalue. The drawback of VQE is that the number of times the Ansatz state and expectation value needs to be prepared is proportional to $1/\epsilon^2$, where ϵ is the desired precision, which could lead to long run-times [129]. Another way to estimate eigenvalues of unitary operations is using the quantum phase estimation (QPE) algorithm explained more in-depth in Section 3.3.4. The advantage of using QPE is that the number of times the experiment is conducted to find the estimate is proportional to a constant. The downside is of course that the circuit depth grows proportionally to $1/\epsilon$.

As quantum hardware technologies improve, it will allow for longer coherence times of qubits and in turn, allows for deeper quantum circuits. To make use of this ability, and to "squeeze" as much power out of the available quantum hardware, Wang et. al proposed the Accelerated VQE (α-VQE) algorithm [138]. We again attempt to squeeze more power out of our quantum hardware by considering how one could implement α-VQE for a distributed quantum computer.

When using VQE for quantum chemistry applications, it is common to prepare parameterized circuits that generate entangled Ansatz states. A commonly used Ansatz is the unitary coupled cluster Ansatz [141], which grows in the number of qubits required to prepare the Ansatz as Hamiltonian complexity increases. A critical part of using a distributed quantum computer for quantum chemistry is therefore preparing Ansatz states over an array of quantum computers. When distributing any quantum circuit across devices, the main complication that arises is when a controlled two-qubit

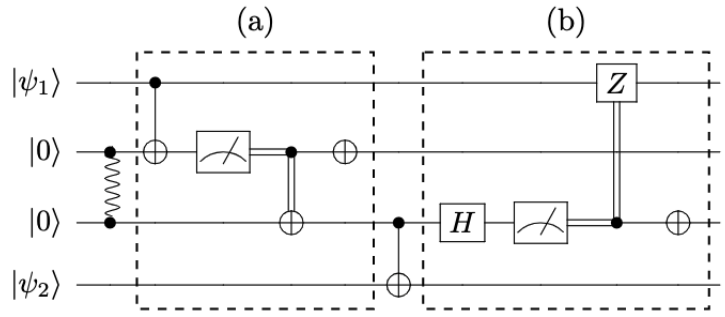


Figure 3.16.: Circuit diagram for a non-local CNOT gate between $|\psi_1\rangle$ and $|\psi_2\rangle$ where (a) is the Cat-Entangler sequence and (b) the Cat-Disentangler sequence.

gate needs to be applied across two QPUs. There are two approaches we consider here. We assume that only entanglement and classical communication are used to achieve this. Alternatively to this, we could consider physically moving qubits between QPUs but this is a much noisier task and we ignore this option. We consider two approaches: Teleporting one of the two qubits to the other QPU so that they are on the same QPU and then performing the two-qubit gate on one QPU locally, the second approach is to use the mechanism introduced in [142] where Yimsiriwattana et. al introduce "cat-entangle" and "cat-disentangle" protocols seen in Fig. 3.16.

Comparing these two approaches in terms of the number of operations needed, we find that using the approach of Yimsiriwattana et. al is more efficient. To use teleportation in a distributed system, we would require 2 Bell pairs to teleport the qubit from one QPU and back again. Using the method of Yimsiriwattana et. al requires just 1 Bell pair to perform a non-local control gate and this Bell pair can also be used to perform multiple control gates when the control qubit is the same as is done in [122] for distributed quantum Fourier transform.

Using the approach of Yimsiriwattana et. al, in the first subsection we consider how, given a collection of QPUs and an electronic molecular Hamiltonian, we can generate a schedule that can be used to estimate the expectation value of the Hamiltonian. We develop two approaches for solving this: the first is a greedy algorithm and the second uses constraint programming. In the next subsection, we consider how we can perform the needed α-QPE step that is required for estimating expectation value over a distributed system and merge the ideas to produce a complete version of a distributed α-VQE.

3.3.3. Scheduling Hamiltonians

An electronic molecular Hamiltonian H can be written as a sum of a polynomial number (with respect to the system size) of Pauli matrices in the form of Eq. (3.3.1), where each $P_i \in \{I, \sigma_x, \sigma_y, \sigma_z\}^{\otimes n}$ is a tensor product of qubit n Pauli operators (or the identify), called a Pauli string, and each $a_i \in \mathbb{R}$,

$$H = \sum_i a_i P_i. \tag{3.3.1}$$

To use a networked quantum computer more effectively, we wish to use a parallelized and distributed approach to expectation value estimation. We motivate the approach as follows. Given the linear nature of estimating $\langle\psi|H|\psi\rangle$, we can break up the summation into its pieces. We need to prepare an n qubit Ansatz for each piece of the sum to estimate each $\langle\psi|P_i|\psi\rangle$ independently to later rejoin the expectation values to estimate $\langle\psi|H|\psi\rangle$. Given the distributed QPU architecture, we need to allocate the qubits in such a way that Ansatz states can be prepared for each P_i in the sum. Later, the coefficients a_i can be merged to produce a single value for $\langle\psi|H|\psi\rangle$.

For Hamiltonians that require a large number of qubits, in this subsection, we consider methods that distribute the expectation calculation of the Pauli strings between a given distributed quantum computer. Here we model a collection of quantum processors {QPU_1,...,QPU_m} as a collection of $q_i \in \mathbb{N}$ qubits (respectively), all of which are located in the same device. Given a set of QPUs and a Hamiltonian in the form of a summation of Pauli strings, a distributed layout of the qubits with the required allocation of communication qubits is produced.

We enforce the following restrictions. Because the goal is to run α-VQE, we know ahead of time that one additional qubit (additional to the qubits in the Ansatz) is reserved for each Ansatz to perform α-QPE. On top of this, we need to reserve qubits for entanglement between QPUs which is necessary when an Ansatz is split between QPUs. The worst case for this occurs when there is a three-qubit control gate (equivalent to a Toffoli gate) where the chain qubits are allocated on different QPUs while performing α-QPE. In this case, since we are using the method of cat-entangling and disentangling, we need to reserve 2 qubits from each QPU for entanglement. We depict such a distribution in Figure 3.17. We formalize this as a problem:

Problem 1 (Ansatz Distribution Problem)**.** *Given a Hamiltonian $H = \sum_{i=1}^{n} a_i P_i$ where each Pauli string $P_i \in \{I, \sigma_x, \sigma_y, \sigma_z\}^{\otimes n_i}$ and a collection of m QPUs described by the number of qubits on the system $[q_1, q_2, ..., q_m]$, output a series of rounds that can be used to estimate, for a given Ansatz $|\psi\rangle$, the expectation $\langle\psi|H|\psi\rangle$. To prepare an Ansatz, when P_i is split between two QPUs, 2 qubits from each QPU have to be allocated to perform non-local operations for preparing the Ansatz $|\psi\rangle$ across two or more QPUs. Moreover, 1 qubit needs to be reserved for α-QPE. The solution to this problem outputs a schedule of distributions in which one can run over the distributed system to obtain an estimate to $\langle\psi|H|\psi\rangle$.*

For the task of distributing the qubits, we take various approaches to this problem. In its essence, this problem is a resource allocation problem. We can therefore gain insight from common solutions to such problems. Common approaches for resource allocation problems are greedy algorithms and constraint programming. We propose an algorithm for each approach in this section.

Greedy Ansatz Distribution

In the greedy algorithm approach, we greedily fill the QPUs with as many Ansatz states as can possibly fit and for the remaining needed qubits, we split them across the QPUs reserving the needed qubits as needed. When the QPUs cannot fit any more Ansätze, the execution of those estimations is moved to the next round. In detail, we propose

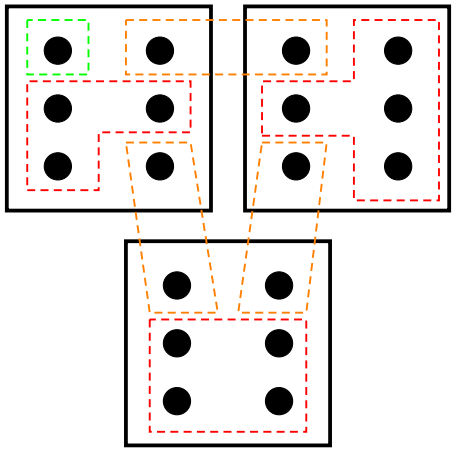


Figure 3.17.: Distribution of a 11-qubit Ansatz on three QPUs with 6 qubits each. One qubit is reserved for α-QPE in green. Communication qubits are reserved in orange. The Ansatz qubits are in red. Two qubits are reserved for communication to accommodate for any control-control gates that could occur when running α-QPE that need to cross QPUs.

Algorithm 9. We refer to an algorithm called **doesNotFit** which simply runs a similar logic as the main algorithm but just ensures a distribution exists for one particular Ansatz. We refer the reader to Appendix A, Algorithm 15 for the detailed algorithm.

Constraint Programming Approach

As another approach to solving Problem 1, we use constraint programming. The trade-off with constraint programming is that setting up a collection of constraints is generally straightforward forward but solving constraint problems on a finite domain is generally NP-complete, trading simplicity for time. We construct the multi-objective constraint program in detail in Constraint Program 8. Using this constraint program repeatedly, we can produce a schedule by running the constraint program on the maximum number of Ansätze that fit in the system and using a solution from the output, once per round, until all Ansätze are covered.

3.3.4. Distributing α-VQE

As discussed in earlier sections, The variation quantum eigensolver (VQE) is a variational algorithm that uses a combination of quantum and classical components and can be used to estimate ground state energies in electric molecular Hamiltonians. To perform chemical calculations, VQE is used with a statistical sampling sub-routine to estimate expectation values with a given Ansatz with a classical optimizer to pick the parameters to minimize the expectation value. In [138], a generalization of VQE is proposed, called α-VQE. The generalization replaces the statistical sampling step with a subroutine called α-QPE, which for the selection of $\alpha \in [0, 1]$ can behave as VQE does,

Algorithm 9 Greedy Ansatz Distribution

Input:

- List of QPU sizes $Q = [q_1, q_2, ..., q_m]$.
- n the qubits for Ansatz
- p the number of Pauli strings to distribute
- Parameters for recursion defaulted to $schedule = \{\}$ and $round = 1$

Output: An Ansatz distribution schedule used to compute $\langle\psi|H|\psi\rangle$ for an Ansatz $|\psi\rangle$ of size n qubits.

GreedyDistribution$(Q, n, schedule, round)$:

if $p = 0$ or $n = 0$ **then**
  return *schedule*
$Q' \leftarrow$ **copy**$(Q) = \{q'_1, ..., q'_m\}$ ▷ Copy Q for modification
$schedule[r] \leftarrow [\,]$ ▷ Initialize the schedule for this round
$couldNotFit \leftarrow 0$
for $i \in 1, ..., p$ **do**
  sort(Q')
  if doesNotFit(n, Q') **then**
    if $round = 1 \wedge i = 1$ **then exit** ▷ The Ansatz does not fit, problem cannot be solved
    $couldNotFit \leftarrow coundNotFit + 1$
    continue
  $distribution \leftarrow [0 \text{ for } \_ \in \{1, .., m\}]$ ▷ A vector of m zeros
  for $j \in \{1, ..., |Q'|\}$ **do**
    $curAllocation \leftarrow [0 \text{ for } \_ \in \{1, .., m\}]$
    $possibleQPUs \leftarrow Q'|_{\{1,...,j\}}$ ▷ Restrict to the first j available QPUs
    if $j = 1$ **then** ▷ No split needed
      $k \leftarrow$ **QPUNumber**$(possibleQPUs[1])$ ▷ The QPU index
      $curAllocation[k] \leftarrow possibleQPUs[1] - 1$
    else
      $k \leftarrow$ **QPUNumber**$(possibleQPUs[1])$ ▷ The QPU index
      $curAllocation[k] \leftarrow possibleQPUs[1] - 3$
      for $q'_s \in possibleQPUs|_{\{2,...,j\}}$ **do**
        $curAllocation[s] \leftarrow q'_s - 2$ ▷ Reserve 2 qubits from the QPUs
    if $\text{sum}(curAllocation) \geq n$ **then** ▷ An allocation is possible
      $remaining \leftarrow n$
      $iteration \leftarrow 1$
      for $q'_s \in possibleQPUs$ **do**
        $t \leftarrow \min\{remaining, curAllocation[s]\}$
        $distribution[s] \leftarrow t$
        $remaining \leftarrow remaining - t$
        if $iteration = 1$ **then** ▷ Remove the respective qubits from the first QPU
          if $j = 1$ **then**
            $q'_s \leftarrow q'_s - t - 1$
          else
            $q'_s \leftarrow q'_s - t - 3$
        else
          $q'_s \leftarrow q'_s - t - 2$
        if $remaining = 0$ **then break**
        $iteration \leftarrow iteration + 1$
      break
  for $q'_s \in Q'$ **do**
    if $q'_s = 0$ **then delete** q'_s
  $schedule[r]$.**add**$((i, distribution))$
return GreedyDistribution$(Q, n, couldNotFit, schedule, round + 1)$

but also can become more efficient by choosing $\alpha > 0$, which requires the ability to run deeper circuits on quantum hardware.

In this section, we take the proposed α-VQE in [138] and map it to a distributed system. The main theme in this section is applying non-local control gates over separated QPUs. We follow the approach of Refs. [114, 142] using entanglement and classical communication to perform control gates across distributed systems, relying on the pre-allocated qubits from the previous section to hold the entanglement across devices.

Distributing α-QPE

The quantum phase estimation (QPE) algorithm is an essential ingredient to many popular quantum algorithms – one such being Shor's algorithm. First discussed by Kitaev in [143], QPE is used to estimate the phase of a quantum state $|\psi\rangle$ that appears after applying a specific unitary operation U to it, where $|\psi\rangle$ is an eigenstate of U. Specifically, QPE aims to estimate the phase ϕ in $U|\psi\rangle = e^{2i\pi\phi}|\psi\rangle$ with high probability. In Fig. 3.18, we depict a circuit representation of QPE applied to a qubit $|\psi\rangle$ where n qubits are used to estimate ϕ.

Here, we adapt a modified version of QPE developed in [138] called α-QPE for a distributed system. α-QPE is a modified version of rejection filtering phase estimation (RFPE) whose circuit diagram is given in Fig. 3.19. α-QPE uses a free parameter α that is chosen depending on the available circuit depth on the specific hardware running the algorithm. With this α, M and θ are selected as $M = 1/\sigma^\alpha$ and $\theta = \mu - \sigma$. Here, σ and μ are parameters for a normal $\mathcal{N}(\mu, \sigma^2)$ prior distribution in the first round of α-QPE for sampling values of ϕ, the "eigenphase" in $U|\phi\rangle = e^{\pm i\phi}|\phi\rangle$. Here U is modified to be a rotation operator that rotates an Ansatz $|\psi\rangle$ by an angle ϕ in the plane spanned by $\{|\psi\rangle, P|\psi\rangle\}$, where P is a Pauli string. More precisely, with the goal of estimating $|\langle\psi|P|\psi\rangle|$, given an Ansatz preparation circuit $R := R(\lambda)$ for some parameter vector $\lambda \in \mathbb{R}^n$ and a reflection operator $\Pi := \mathbb{I} - 2|0\rangle\langle 0|$, $U := R\Pi R^\dagger P R \Pi R^\dagger P^\dagger$ and the circuit depicted in Fig. 3.19 is executed to obtain a value E. When E is obtained, rejection sampling is performed to produce a posterior distribution, which can be shown to again be normal, in which to again sample values of ψ. This process is repeated until sufficient accuracy is reached. Once an estimate for ϕ is obtained, one can recover $|\langle\psi|P|\psi\rangle|$ using the relation $|\langle\psi|P|\psi\rangle| = \cos(\phi/2)$. In [138], mechanisms to recover the sign of $\langle\psi|P|\psi\rangle$ are provided.

In this subsection, we tackle three key steps in to adapt α-QPE for a distributed system: The first is mapping the state preparation circuit $R(\lambda)$ across multiple QPUs, the second is then to map U to a distributed system, and the third, performing the controlled operation in Fig. 3.19. We solve these in order. The solution to the first task takes Ansatz preparation circuit $R(\lambda)$ and develops a mechanism such that it can be applied when some qubits are physically separated. Here we consider $R(\lambda)$ a variational form, a parameterized circuit used to prepare an Ansatz. We give an algorithm to achieve this in Algorithm 10.

The high-level idea of Algorithm 10 is, given the circuit representation of $R(\lambda)$ as a series of layers, where each layer is a collection of gates in a layer of the circuit, and

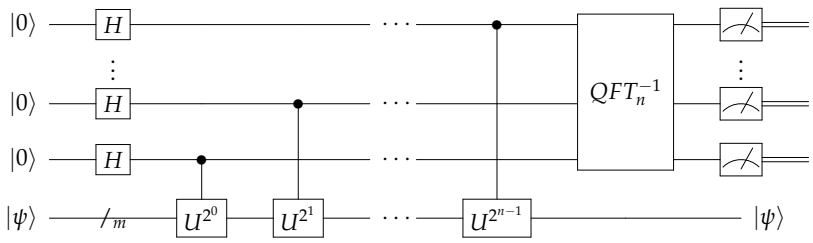


Figure 3.18.: Circuit diagram for QPE with unitary operation U and eigenstate $|\psi\rangle$.

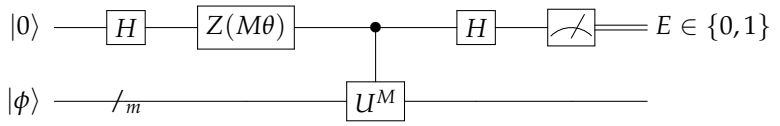
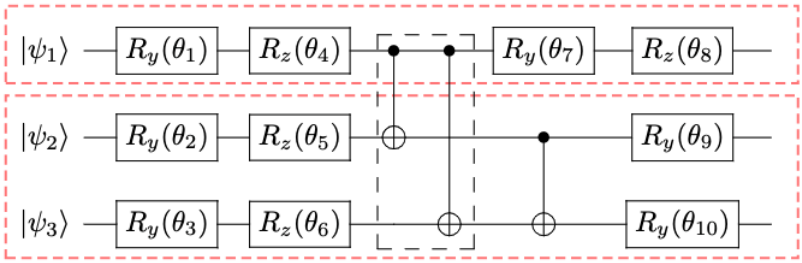


Figure 3.19.: Circuit diagram for RFPE. $Z(M\theta) \coloneqq \text{diag}(1, e^{-iM\theta})$.

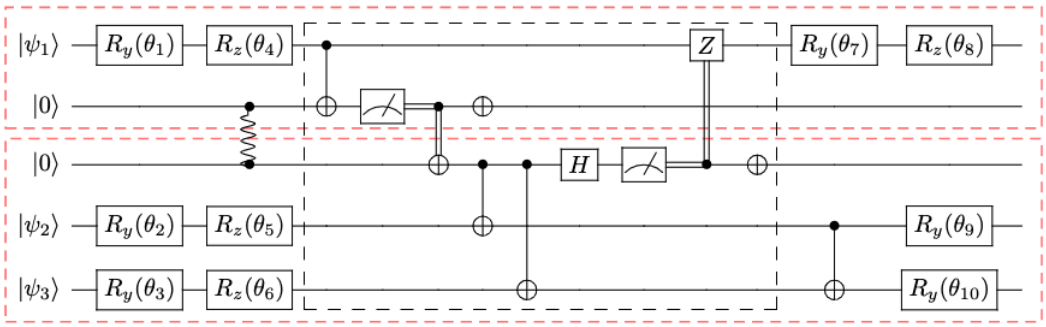
a mapping of qubits, to search for any control gates where the control and target are physically separated between two QPUs. When found, insert, between the current layer and the next layer in the circuit, the necessary steps to perform the control gate in a non-local way using the cat-entangling method. We also ensure that entanglement is established between the two QPUs ahead of time by pre-pending an entanglement generation step. As an optimization, the cat-disentangler step can be shifted to a later layer if the non-local control gate has the same control qubit and no operations on that control qubit in between controlled gates. Note that we can generate a distributed $R(\lambda)^\dagger$ in the same way. From the previous subsection, the proposed solutions to Problem 1 ensure that there are two qubits reserved on each QPU for the entanglement qubits needed for non-local operations. Producing the layering of a circuit can be done in a straightforward way and we assume that this structure is the input to the algorithm. We depict an example of running the algorithm in Fig. 3.20.

The next step is to map $U \coloneqq R\Pi R^\dagger P R \Pi P^\dagger R^\dagger$ to a distributed system. One observation that can be made immediately is, since P is a Pauli string, $P^\dagger = P$, so there are no additional steps needed to map $P^\dagger$. P is a separable operation (i.e. there are no 2 qubit gates) and therefore we can apply each piece of P in a single layer with no added inter-QPU communication. For mapping $R(\lambda)^\dagger$ to a distributed system, as discussed, given an $R(\lambda)$ as a circuit that is not distributed, we can obtain $R(\lambda)^\dagger$. To obtain the mapping, we can run Algo. 10 with $R(\lambda)^\dagger$ as the input with the same Ansatz distribution. Next, we consider the n qubit reflection operator Π which can be decomposed (locally) as a series of single qubit gates and $CNOT$ operations. We can therefore again use Algo. 10 to map a provided reflection Π to a distributed architecture given the Ansatz distribution as input.

For the control part of α-QPE, we consider the controlled version of U, $c-U$, because of the structure of U, one can see that the only operation that needs control is in the reflection Π since if Π is not applied, $c-U$ is reduced to the identity. Here it will be the case that we need to execute control-control gates (CC-gates). If the Ansatz is split between QPUs, then two qubits need to be reserved on each QPU to accommodate for



(a) 3 qubit variational form.



(b) 3 qubit variational form distributed across two devices.

Figure 3.20.: An example of running the **DistributedRemapper** algorithm.

CC-gates. This is guaranteed by the scheduling algorithm in the previous subsection and there will always be two free qubits reserved such that we can apply Algo. 10 again after adding a control connection to each gate of the circuit representing Π the distributed form, excluding the previously added non-local steps, to produce a circuit that achieves the controlled version of Π.

The remaining steps of α-QPE are the two complications that arise which are discussed in [138] Section 2b. At each iteration of α-QPE the Ansatz $|\psi\rangle = 1/\sqrt{2}(|\phi\rangle + |-\phi\rangle)$ needs to be collapsed into either $|\phi\rangle$ or $|-\phi\rangle$. In [138], Wang et. al propose a statistical sampling method in which one can apply a constant number of iterations to, with high confidence, both estimate the sign of $\langle\psi|P|\psi\rangle$ and ensure that $|\langle\psi|P|\psi\rangle| > \delta$. When this bound holds, then with high confidence, $|\psi\rangle$ can be efficiently collapsed to either one of $|\phi\rangle$ or $|-\phi\rangle$. Once this is performed, we apply the α-QPE procedure as normal. If high confidence cannot be achieved, then instead of using the α-QPE circuitry, statistical sampling continues. Statistical sampling in this setting implies repeatedly preparing $|\psi\rangle$, applying the single layer Pauli string P, to estimate $\langle\psi|P|\psi\rangle$. When the bound does not hold, statistical sampling is performed until $\langle\psi|P|\psi\rangle$ is estimated with sufficient precision in the normal VQE sense. We follow the method of Wang et. al but use the modified $R(\lambda)$ circuit needed to prepare the Ansatz over a distributed quantum computer. We write this whole procedure in Algorithm 12.

Definition 19 (Schedule)**.** *A schedule S is a collection of r lists where each element of a list contains the distribution of qubits on the m QPUs. Each distribution is a list of qubit allocations on each QPU $q_i \in \{0, ..., Q_j\}$ where Q_j is the number of qubits on QPU j. If the Ansatz is not allocated in a round $r' \in \{1, ..., r\}$, it does not appear in the distribution list. The structure of a*

schedule is as follows:

$$S = \{ \\ 1 : [[q_1, ..., q_m]_1, ..., [q_1, ..., q_m]_{n_1}], 2 : [[q_1, ..., q_m]_{n_1+1}, ..., [q_1, ..., q_m]_{n_2}], \\ ..., r : [[q_1, ..., q_m]_{n_{r-1}+1}, ..., [q_1, ..., q_m]_{n_r}] \\ \}$$

The subscripts on the qubit count lists represent the index of the Pauli being estimated.

Distributed α-VQE

To conclude the mapping of a localized, monolithic version of α-VQE to the distributed version, we need to replace the α-QPE subroutine with the distributed α-QPE version from the previous section. For completeness, we write distributed α-VQE as an algorithm in Algorithm 13.

Analysis

In this section, we analyze the properties of the distributed quantum circuits in relation to the Ansatz size. First, we compare the duration of computation using three methods of performing the estimates of the expectation values: estimating in parallel, on one single QPU the size of the Ansatz, and using parallel and distributed computing. When running in parallel, one Pauli string is estimated per QPU. The limitation is that the Ansatz can be only as big as the smallest QPU, minus the qubit for α-QPE. In the single QPU case, we assume the full Ansatz can fit on the QPU, and therefore no gates are distributed. Finally, in the distributed and parallel case, Pauli strings are estimated similarly to the parallel case, but multiple Ansätze can be placed on a single QPU as well as split between multiple QPUs with distributed control gates.

To get an estimate for the number of gates used, we analyze the pieces of the U operator defined in the previous section. The reflection operator Π has the equivalent cost, up to $2n$ single qubit gates to an $(n+1)$-qubit Toffoli gate [138, Section II.B]. Without ancilla qubits, currently, the circuit depth to implement such a gate grows linearly $O(n)$ [144] with improved linear scaling with 1 ancilla qubit [145]. When $\lceil \frac{n-2}{2} \rceil$ ancilla qubits are available, the depth can scale as $O(\log n)$ [146] to implement with $6n - 6$ CNOT gates. The additional ancilla qubits to decrease the circuit depth could be considered in the Ansatz distribution phase from Subsection 3.3.3, and we leave it to future work to analyze this change. Here we assume no additional ancilla qubits. For the Ansatz preparation $R(\lambda)$, in most of the applications to date, the circuit depth is $\Omega(n)$ [147], meaning it has a tight upper and lower bound proportional to the number of qubits, which could be the most significant overhead in this process.

We demonstrate the time trade-off. In Fig. 3.21 we assume we have a QPU cluster with 5 QPUs each with 10 qubits. We determine a rough upper bound on the number of gates needed to perform distributed computing and summarize the time weight and gate quantity scaling in Table 3.4. In Fig. 3.22, we show the maximum number of qubits that an Ansatz can be composed of using four different-sized QPUs and with respect to adding additional QPUs of that size to the distributed system.

Operation	Execution time weight	Quantity scaling
CNOT	5	$O(n^4 \cdot \log n)$
Single qubit gate	1	$O(n^4 \cdot \log n)$
Measurements	2	$O(n^4 \cdot \log n)$
Entanglement generation	8	$O(n^4 \cdot \log n)$
Classical communication	2	$O(n^4 \cdot \log n)$
Output merging	3	$O(m)$

Table 3.4.: The time scaling of gates. n represents the number of qubits in the Ansatz and m the number of QPUs. The execution time weights are derived from [148] for superconducting qubits. The quantity scalings are based on a Bravyi-Kitaev mapping [149].

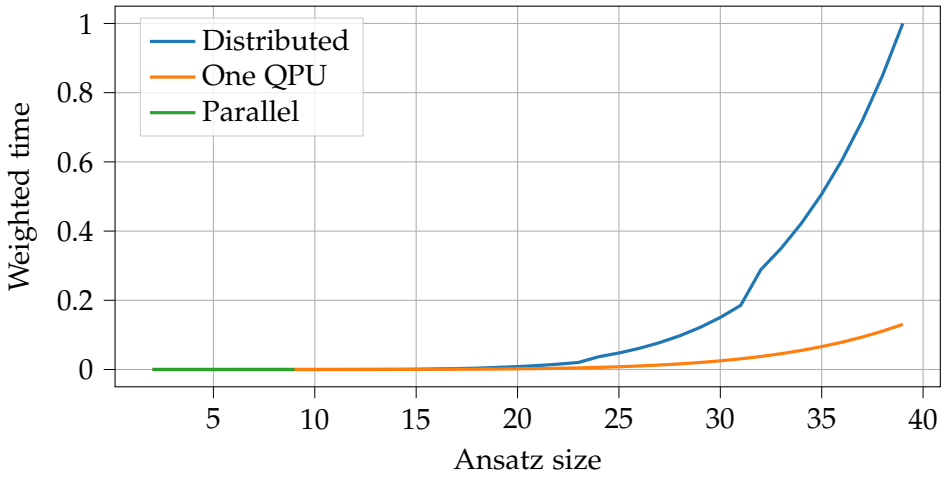


Figure 3.21.: This plot is of a weighted time using the greedy distribution of the Ansatz for growing Ansatz sizes with 5 QPUs each with 10 qubits. The green line shows the timing for running 1 Ansatz per QPU. It cuts off at 9 qubits. The orange line is if all 50 qubits were on 1 QPU. The blue line is if we use a distributed Ansatz over the 5 QPUs.

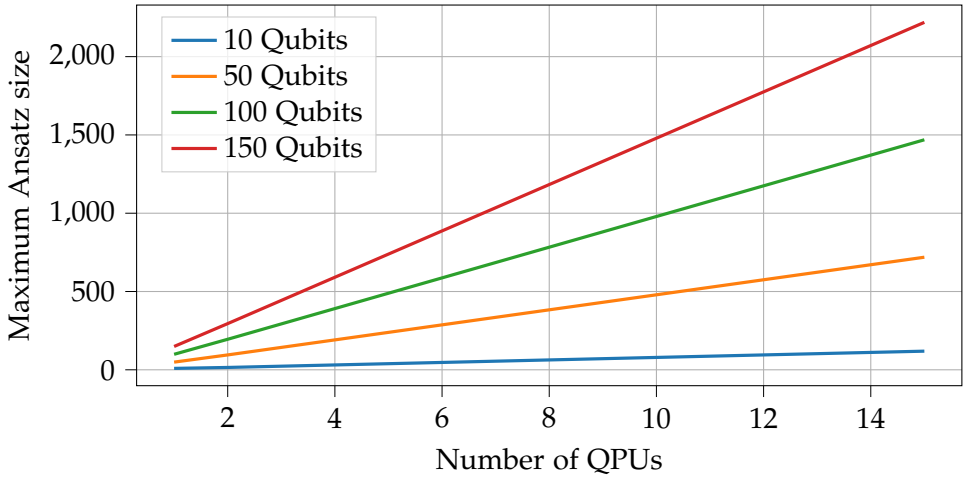


Figure 3.22.: The maximum Ansatz size that would fit on a distributed system of QPUs. The maximum Ansatz size is given by $\sum_{i=1}^{n} q_i - 2n - 1$ with n QPUs with $q_i > 2$ qubits on QPU i.

Applications for Quantum Chemistry

In this section, we take an example of an electronic molecular Hamiltonian for the chemical H_2. To estimate the Hamiltonian for this molecule with 2 electrons and 2 active orbitals, we require 4 qubits when using a Bravyi-Kitaev transformation. We can quickly obtain the Hamiltonian using the Pennylane Python library [150]. The Hamiltonian in this case, under the Bravyi-Kitaev transformation, is of the form,

$$H = \sum_{i=1}^{15} a_i P_i, \tag{3.3.2}$$

where we are concerned with the number of elements in the sum and less so about the constant factors and therefore to perform α-VQE, we will need to estimate 15 Pauli strings. In this example, we will consider a distributed quantum system of 3 QPUs each containing 9 qubits. If we use these parameters as input to the algorithms in Section 3.3.3, the output configuration would be the one depicted in Figure 3.23. In one round, 4 Ansätze can fit across this distributed system, and so at least 4 rounds need to be executed. We can use the same allocation for the first 3 rounds and in the last round eliminate the distributed Ansatz to reduce the need for cross-communication between QPUs.

For the Ansatz preparation, we use the circuit $R(\lambda)$ depicted in Fig. 3.24 (a). From the 4 Ansätze, three of them will be able to run the α-QPE step without distribution of the Ansatz. The fourth Ansatz is on the other hand distributed and will need to use the circuit in Fig. 3.24 (b) for preparation. For simplicity, we include arbitrary qubit rotations which are represented by the $R(\lambda_1, \lambda_2, \lambda_3)$ gates, where $\lambda_i \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ for $i \in \{1, 2, 3\}$. Next, we need to perform the reflection operation Π described in Section 3.3.4, whose circuit is shown in Figure 3.25 (a). An equivalent circuit is also shown which decomposes the 4-qubit Toffoli gate into a series of controlled and single-qubit

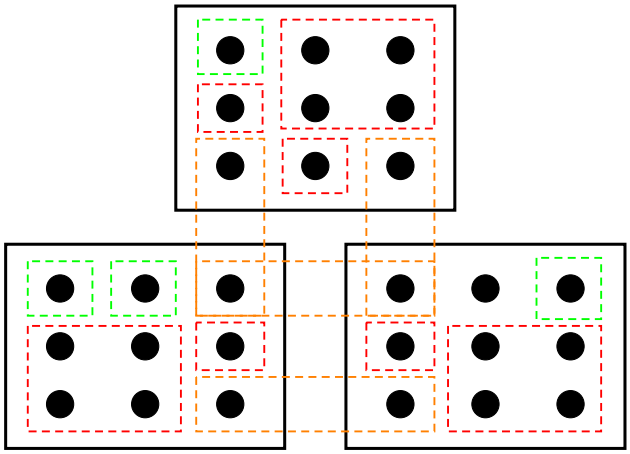


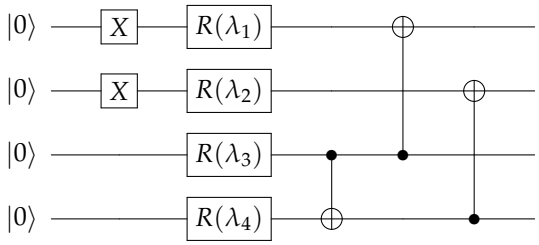
Figure 3.23.: A distributed Ansatz of size 4 on three QPUs with 9 qubits. The green outlined qubits are reserved for running α-QPE. The red outline qubits are for the Ansätze. The orange outlined are qubits reserved for entanglement between QPUs for non-local gates. One qubit is left idle.

gates. We again need a distributed version of the reflection operation to support the Ansatz which is distributed. We show this circuit in 3.25 (b). Here we introduce gates for the cat-entangler and cat-disentagler sequences. Here, 4 qubits are allocated for performing the non-local gates. Now, for running α-QPE, we need a circuit for $c-\Pi$, which is the control part of $c-U$. Here is where it is critical to have 2 entanglement qubits for each splitting of the Ansatz on each QPU since, as seen in Figure 3.26, there are control-control gates that occur across QPUs. With this collection of gates, we can run α-QPE and therefore using the algorithm in Section 3.3.4 run α-VQE.

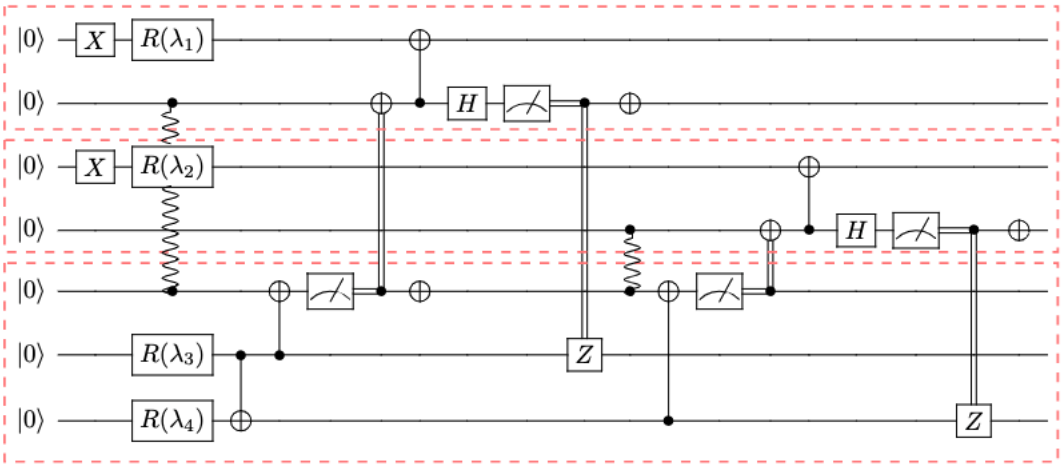
3.3.5. System Architecture and Engineering

Because it will be difficult to construct large, monolithic quantum computers in the near future, it will be a viable option to instead connect smaller quantum computers together using a network in a distributed manner. One can therefore consider networked control systems (NCSs) to manage the distribution of resources for running quantum algorithms. Such a system could allow for more flexibility regarding hardware configurations and the ability to add more devices while minimizing integration overheads dynamically. An NCS is a network of devices connected using the network to perform a specific mutual task orchestrated by a control system [151, 152]. Among the other thing, NCSs are used to perform distributed or parallel computing, control a fleet of robots or drones, or smart grid systems deployed in modern cities [153].

Networked control systems can have various architectures for the control system part. These systems can either have a centralized controller where communications amongst the nodes are restricted to a local area network (LAN) or a decentralized controller system that is connected via an internet or wide area network (WAN). These two scenarios resemble how distributed quantum computers could potentially be

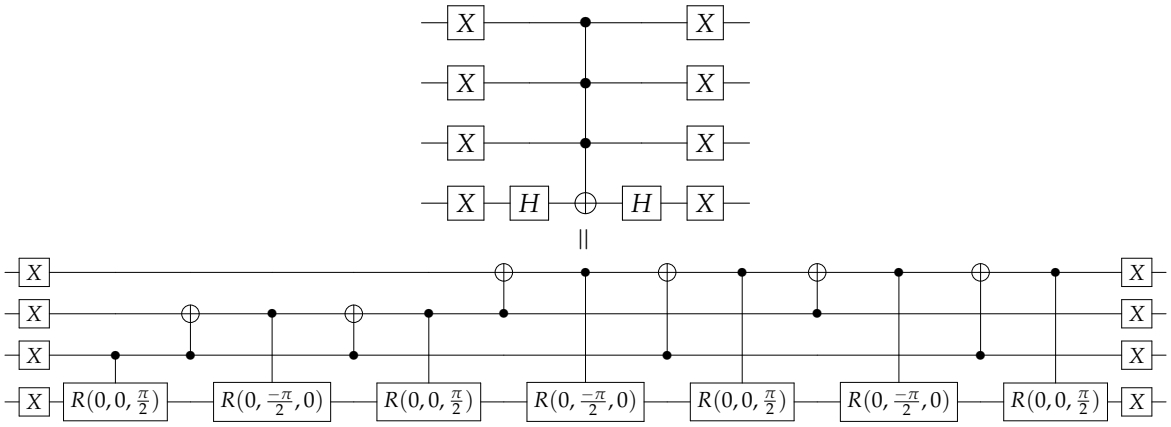


(a) Circuit for $R(\lambda), \lambda_i \in [-\pi, \pi]^3, i \in \{1, ..., 4\}$.

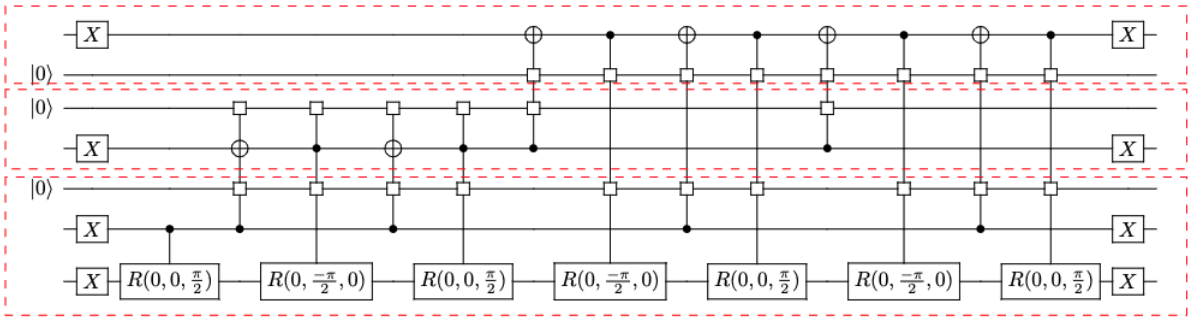


(b) Circuit for a distributed $R(\lambda)$. The red dashed lines represent the individual QPUs.

Figure 3.24.: Distributed circuit mapping for $R(\lambda)$.

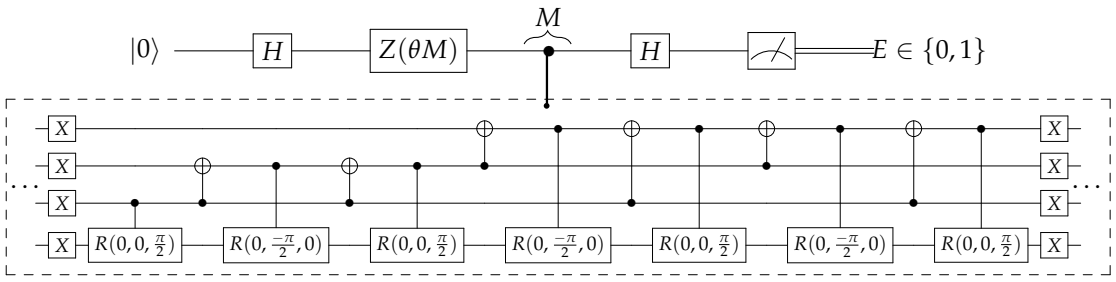


(a) Circuit representation of reflection Π.

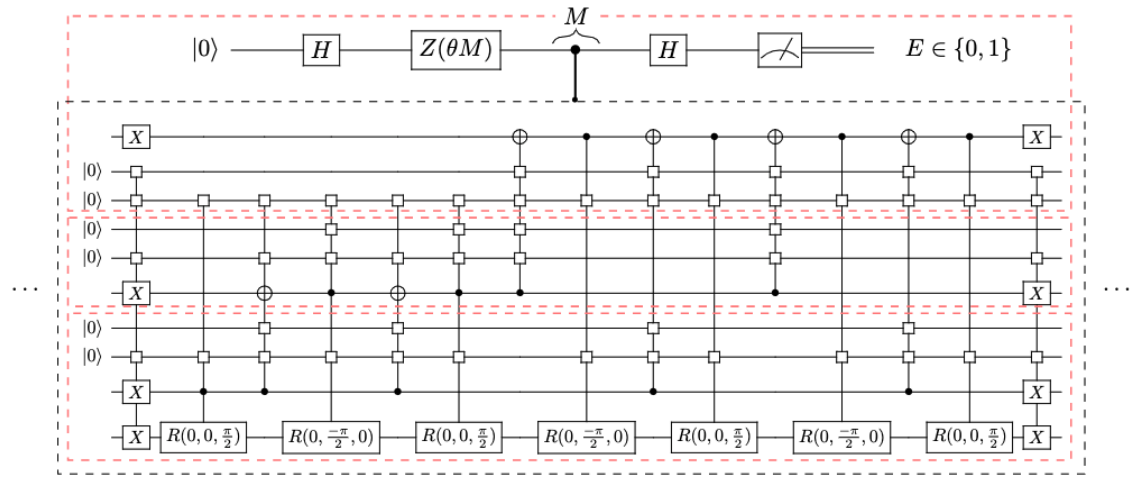


(b) Circuit representation of distributed Π. The square gates in the 4 qubit gates represent the cat-entangler/disentager sequence.

Figure 3.25.: Distributed circuit mapping for reflection Π.



(a) To run α-QPE, one needs to perform a controlled U operation M times, where $U = R\Pi R^{\dagger} P R \Pi R^{\dagger} P$. The control portion to consider is $c-\Pi$. We depict the $c-\Pi$ part, where the other parts of U are applied before and after what is depicted, which do not need to be controlled.



(b) Distributed $c-\Pi$. The square gates in the 4 qubit gates represent the cat-entangler/disentager sequence.

Figure 3.26.: Distributed circuit mapping for $c-\Pi$.

networked. In the first case, one can consider a single owner of multiple quantum devices where all of the quantum devices are located in the same room or building, specifically, the network owner would know the network topology and information about the hardware in the network. In the second setting, multiple quantum computers located possibly far apart are potentially connected by multi-hop connections where the owner of the hardware between the hops is possibly different. Here, more advanced protocols that consider security and robustness will be needed potentially leading to a fully-fledged quantum Internet.

To use a network of distributed quantum computers efficiently, one must develop robust communication protocols such that communication and control between the quantum devices in the network are efficient and reliable. In this section, we consider quantum systems with classical control and separated quantum processing units. We consider a QPU to be a combination of a three-layered system depicted in Fig. 3.27. The QPU in this case is a layered system with inputs and outputs to a communication network through a classical computer or a CPU. The CPU interfaces with the network as well as controls the FPGA based on the control instructions from the network which in turn controls the qubits to perform quantum operations on the qubit layer. Qubit measurements and other classical messages are transmitted back to the network via a reversed path.

We consider the two different network configurations described and get into more detail about how these systems could be implemented in practice. We list the communication requirements needed to perform distributed quantum computations. We explore some available protocols to achieve these requirements under two scenarios. In the first one, there is a centralized controller of the system, and communication to devices is classical information and quantum entanglement can be sent directly to other quantum processors without routing. The second case is when control over the network is not centralized but has a single user. We then propose a control system using Deltaflow.OS to orchestrate distributed quantum computing.

In this section, we discuss two possible network architectures for distributed quantum computing control systems. The major difference between the two systems is the centralization of the control. In the first system, we consider a distributed architecture with centralized control. In the second, the control is split such that each QPU in the system has its control. In this section, we describe these two systems in depth. In later sections, we go into detail regarding the communication requirements needed to run the systems and the potential protocols to achieve them.

Centralized-Controlled Distributed Quantum Systems

The first distributed quantum computing scenario we consider is depicted in Fig. 3.28. This scenario is one where there is a single controller, and the quantum hardware behaves only according to the instructions that are fed from this controller. The QPU systems are connected to the controller via a classical network and further, they are connected both classically and quantumly – so that they can generate entanglement amongst themselves. The main idea here is that the CPUs in the network have a static IP and can be accessed by centralized control. The finer synchronization between the

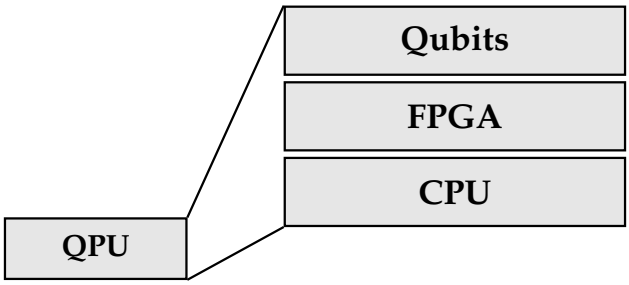


Figure 3.27.: Internal layering of a QPU. We assume there is a layered architecture. The CPU instructs the FPGA which in turn controls and measures the qubits. The CPU also interfaces with the network.

QPU nodes is delegated to the CPU controlling the FPGA layer of the QPU from the centralized control ahead of execution time. The CPUs control the FPGAs and the FPGAs communicate over fixed low-latency links. This latency can be accounted for the control instruction scheduling. At a small and medium size, this network scheme will be best suited, but when many nodes are added to the network, a system with a distributed control is better suited, which we discuss in the next subsection.

Decentralised-Controlled Distributed Quantum Systems

With a decentralized control system, the nodes in the network are no longer in a "master-slave" relationship because the hardware is no longer controlled by a single entity. Resources in this setting need to be requested from various parties and there is no guarantee that the requested resource will be available at the time of the request. Access to the controllers is hidden behind a firewall and their IP, MAC, and inner network configuration is potentially not exposed. We assume that the QPUs are offered by various vendors that have agreed to offer a base set of services: They provide access to quantum hardware for a maximum amount of time per instruction set execution, they offer classical communication input and output to a pre-specified IP address where the communication stream is established before execution, and they allow for remote entanglement to be established between quantum devices on specified quantum hardware. In this case, the control information between QPUs is needed and we will need a low-latency protocol that works in the network layer so that the control messages can be routed.

3.4. Networked Control and Algorithm Scheduling

3.4.1. Distributed Quantum Algorithm Scheduling

For networked quantum hardware to execute instructions synchronously, a method of dictating to the devices when the instructions should be executed is needed. In this section, we propose a temporal operation schedule, that is, a schedule of the operations with precise timestamps for execution. These schedules can then be sent to each QPU in the network with a time to begin execution. Because quantum gates generally have

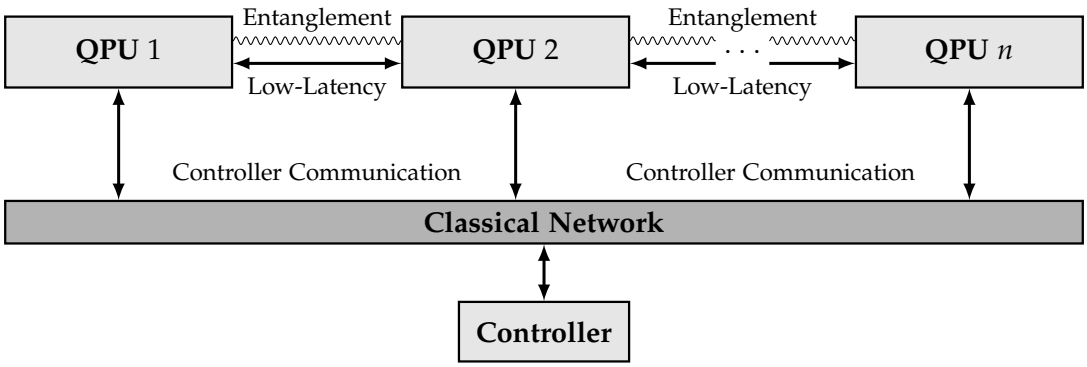


Figure 3.28.: A networked control system with a centralized controller. The triangular arrowheads represent classical connections, and the diamond-shaped arrowheads represent quantum connections used for establishing EPR pairs. Here we assume the network between the QPUs is completely connected in terms of quantum and classical connections, that is, each QPU has the same connections as any other QPU. In the completely connected network, this network is single-use for transmitting with low latency. Moreover, each QPU is connected to a common bus line that handles all the latency-tolerant message exchanges among the nodes.

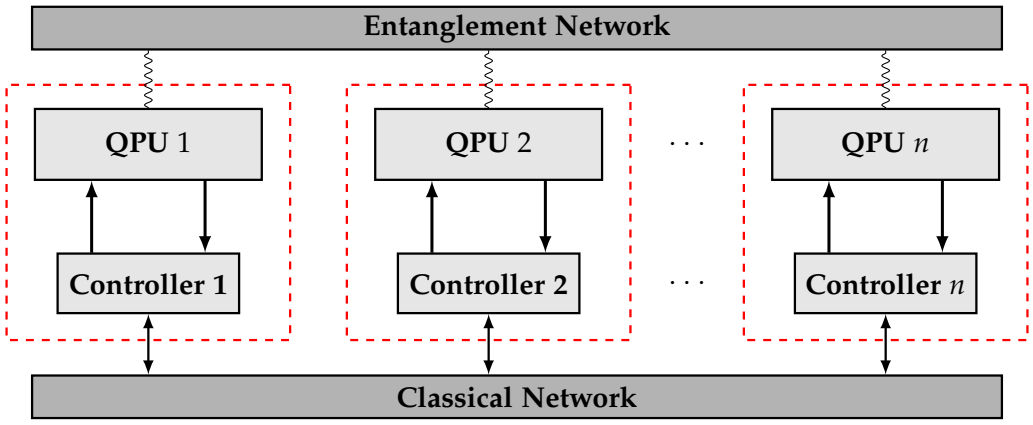


Figure 3.29.: An inter-networked distributed quantum computer with decentralized controllers. Here, some independent controllers control their respective quantum processing systems. Entanglement is generated with an entanglement network requiring possibly multi-hop entanglement routing. The controller is placed between the QPU and the quantum network since in this scenario, a quantum processing layer will be needed.

an upper bound for how long they take to execute, we can use this information when generating the schedule. Here, we assume that all gates have a known execution time as well and that latency times for classical communication and entanglement generation are known. We formalize the problem as follows:

Problem 2 (Distributed Quantum Algorithm Scheduling)**.** *Given a distributed circuit as a series of gate layers, where each layer contains a collection of gates to be applied on the qubits in the system, and the gate times (i.e. the amount of time it takes to perform the gate) of each gate for each QPU in the system, produce a temporal gate execution schedule such that the following constraints are obeyed:*

1. *Sending and receiving classical communication or entanglement between two parties occurs at the same time for the sending and the receiving parties.*
2. *One qubit operation occurs per time instance per qubit for the duration of the gate time (i.e. no overlapping gates).*
3. *At the start of a controlled operation, both qubits need to be available to perform the control gate (i.e. one qubit cannot have a gate operation ongoing).*

At the start of the problem, it is assumed that all nodes in the distributed system have synchronized clocks. We assume routes for any multi-hop communication or entanglement generation are already established and are already calculated into the communication time bounds. Moreover, it is assumed that swap gates are not considered in the scheduling and are assumed included in the worst-case gate times provided.

Comparing and creating a hybrid temporal planning approach with constraint programming for quantum circuit scheduling has been investigated in [154] for the max-cut problem. The difference here is the level of compilation is deeper as they include swap operations since they limit to nearest-neighbor interactions between the qubits. A temporal planning and constraint programming approach is therefore sensible. Here, we do not enforce nearest-neighbor interaction and assume this process is included in the worst-case timings for two-qubit gates for swapping qubits to their nearest neighbor if needed and remapping the index of the qubit so it does not have to be swapped back. We assume at the end of each layer of gates, each qubit will be free to be operated on and no swapping is needed therefore do not use any constraint programming.

The output of the schedule for each QPU will have the form of Table 3.6. Table 3.5 is an intermediate schedule that is used before splitting the schedules for each QPU. For a list of all possible commands and their descriptions, see Appendix A. We approach this problem as follows. We start with high-level instructions which are entanglement generation, single qubit gates, classical communication, and control gates. We generate an instruction list using this gate set. We then take the high-level circuits and break them down into finer control instructions. Once the full schedule is created, we can split the instructions such that the instruction schedule is for a single QPU. The instruction sets can then be sent to their respective QPUs and the algorithm can start. To ensure gates are performed in the correct order, we layer the circuits as done in the previous section and schedule the circuits layer by layer, iteratively constructing a full schedule. In complete form, we propose Algorithm 14.

Command	QPUs	Time
$CTRL[G, \text{qID}, \text{qID}]$	QPU1	T_1
$GEN\_ENT[\text{qID}, \text{qID}]$	QPU1, QPU2	T_2
$CLASSICAL[\text{cID}]$	QPU3, QPU2	T_3
⋮	⋮	⋮
$SINGLE[G, \text{qID}]$	QPU5	T_n

Table 3.5.: T_i is the time to execute the particular gate. The first step of Algorithm 14 generates a table of gates with QPU information before filtering the gates for each QPU for execution.

Command	Time
$CTRL[G, \text{qID}, \text{qID}]$	T_1
$SEND\_ENT[QPU, \text{qID}]$	T_2
$REC\_CLA[QPU, \text{cID}]$	T_3
⋮	⋮
$SINGLE[G, \text{qID}]$	T_m

Table 3.6.: T_i is the time to execute the particular gate. The output of Algorithm 14 will generate a collection of schedules in this form for each QPU.

3.4.2. Protocols

To run a distributed quantum algorithm with a distribute quantum computer using the architectures proposed in the previous section, certain communication requirements are needed to ensure execution is possible. Protocols for controlling networked systems exist in practice in the centralized and decentralized cases, and we explore some examples of them in this section.

The first requirement considered is the classical communication between the controllers and the QPUs. Here what is needed is a method for sending the computation instructions to the QPUs which can be done at slower latency, as well as a method for sending low-latency control bits between the QPUs. We explore methods for achieving this in the two cases. Clock synchronization is a commonly used scheme in distributed control systems. We consider an example of architectures using clock synchronization on a large scale. Lastly in the multi-vendor case, we discuss the certification steps needed to ensure multiple vendors can execute distributed quantum algorithms cooperatively.

Selecting the specific hardware that can execute these protocols is left to future work as tasks such as entanglement generation is still in a primitive state and may not exist to the extent we need for years to come. Also, as qubit technologies improve, the need for as-low-as-possible latency could be loosened, and other protocols could be used in replacement. Here we explore examples that could potentially achieve what is needed to perform distributed quantum computing.

Classical Communication

To run distributed quantum algorithms, there are specific non-local tasks that need to be carried out by the distributed system such as receiving control commands, sending measurement results to the controller, and sending qubit measurements between the QPUs at low latency to perform non-local control gates. In this section, we explore communication protocols that can be used by the control system to accomplish running distributed algorithms. Here we explore some examples of existing protocols that exist at an industry level.

For the centralized control case, we neglect the routing of information and assume each node is connected both classically and quantumly to another. As discussed, we propose that there is a classical network connecting the QPUs to a centralized controller forming a "master-slave" relationship with the additional network of dedicated connections between the QPUs for the sole purpose of low-latency communication. This communication does not go through the CPU of the QPU, but directly between the FPGAs to perform the non-local gates.

When using a centralized control system, to perform slower communications between the QPUs and the controller one could consider two options. The first is to simply connect the CPU portions of the network to the controller using a local area network and communicate using TCP/IP from the controller to the QPUs. In this case, one would need to closely monitor that communication traffic does not overwhelm the network. Another approach that has this feature built-in is to use a protocol often used in industrial control systems. The Modbus communication protocol [155]. Modbus is an open protocol used in a centralized controller master-slave setting as is this centralized controller setting. It is a messaging structure that allows heterogeneous devices to communicate with a centralized controller and receive control messages from the controller. The Modbus protocol can be used over a local network using TCP/IP making it easier to install into existing commonly used Ethernet networks. With Modbus, the controller can send the control instructions to the CPU portion of each QPU which can be sent to the FPGA to perform the portion of the quantum algorithm. With Modbus, the controller can also receive the qubit measurement results from the QPUs once the algorithm is complete.

For low-latency communication of short messages ($<$ 1 byte) between FPGAs there are existing methods that can be used to communicate at the ultra-low latency range ($<$ 1 ms). In the high-performance computing domain, FPGA networks for ultra-low latency, and high bandwidth communication are realized. Here we need to consider that the FPGAs may be meters apart. Connecting the FPGAs with, for example, 10 Gigabit Ethernet for sending short messages and using custom communication protocol and small form-factor pluggable (i.e. SFP+) transceivers, the latency of 300 nanoseconds is possible for each link [156]. With fiber, the latency can be even further reduced.

Another approach that can be integrated again comes from the industrial control domain. Industrial control, especially in the power sector faces issues where some devices need constant monitoring, and reacting to the changes needs to be done at very fast speeds. A method used is the Mirrored Bits [157] protocol. Mirrored Bits is a communication protocol for ultra-low latency communication adding additional

latency of approximately 200 μs for message processing in addition to the latency from transmitting signals over the communication link. Mirrored Bits could be used in this setting to transmit qubit measurement data. The devices that perform the Mirrored Bits protocol which is manufactured by Schweitzer Engineering Laboratories are programmable and can trigger different routines on the FPGA depending on the input bit. These devices are commonly used to frequently monitor sensor data to trigger emergency shut-offs as fast as possible, for example.

In the decentralized case, a dedicated wide area network could be established between the vendors such that a link-layer (of the OSI model) protocol is used for the classical control information between the QPUs. Low latency communication can be achieved using the link-layer protocol called the Generic Object Oriented Substation Event (GOOSE). In particular, IEC 61850 is a GOOSE Ethernet protocol meeting time-sensitive communications and high-speed performance requirements of automation applications. At the link layer over an Ethernet network experimental results show GOOSE can be used to transmit in the 0.5 ms scale [158]. When routing is involved, naturally, the latency will grow. If TCP/IP protocols used over the Internet are considered, then it is unlikely one could create any latency guarantees. If instead there are dedicated wide area networks with routing, one could consider the network-layer version of GOOSE called Routable GOOSE [159]. In [158], R-GOOSE is analyzed over a wide area network using a particular data distribution service and was shown to transmit with an average latency of around 8 ms.

Overall, there can be much to learn from looking into the power automation industry, as many low-latency and fast reaction systems have been developed which have carryover into distributed quantum computing. These protocols have been tested for robustness and security and could potentially fit well for doing distributed quantum computing over a wide area network. Moreover, networking FPGAs

Clock Synchronization

For centralized control, the assumptions of clock synchronization and full connectivity at a small scale are not overly restrictive. High precision clock synchronization can be achieved even in very large configurations (i.e. that comprise a large number of devices) using methods such as in the White Rabbit Project [160]. White Rabbit is used at CERN to synchronize over 1000 nodes with sub-nanosecond accuracy. This is achieved using Ethernet with lengths of up to 10 km, with experiments demonstrating an average of 160 ps skew between similar clocks – regarding clock environmental variables such as temperature – after several hours [161]. This protocol can be integrated into the centralized controller case so that all of the hardware used has synchronized clocks. Once the number of nodes in the network becomes large, routing and efficient network topologies become critical.

In the decentralized case, the controllers will need to perform a coarse-grain time synchronization via classical network synchronization protocols, and a fine-grain synchronization, a precise notation of time can be shared. GPSs directly connected to FPGAs can be a solution where shielding does not stop the incoming signals. This process is common in distributed physical experiments such as in the Super-

Kamiokande Detector [162] and the CERN-OPERA experiment [163]. Each node will need to implement extra steps to guarantee that the timing information is constantly accurate.

Entanglement Generation

To perform the non-local control gates needed in the distributed circuits, the ability to share high-quality entanglement between quantum processors is critical. Entangle generation has been achieved in various qubit technologies such as in optical photons, NV-centers, and superconducting qubits [164], but entanglement generation in quantum networks is an ongoing research topic. We consider deterministic entanglement generation schemes such that there is a guaranteed entangled pair available and shared between the quantum processors when it is needed. Experimental results demonstrating deterministic delivery of entanglement using NV-centers in diamond as qubits have been shown in [66], generating heralded entanglement at a rate of 39 hertz, three orders of magnitude better than previously known results and guaranteeing an entangled pair every 100 milliseconds with fidelity greater than 0.5 without pre- or post-selection. Methods for improving the results further are also proposed. This gives evidence that using entanglement to perform distributed quantum computing can become more feasible using various qubit technologies. As technology regarding entanglement generation and qubit stability improves, the deterministic entanglement generation rate can be improved.

Customer-Vendor Certification

In the case of a decentralized controller, additional protocols are required to ensure that all parties can execute the distributed quantum algorithms and an execution schedule can be made such that non-local operations are performed synchronously. In this setting, the user has no control over the quantum hardware, and therefore a protocol for ensuring the user's instructions can be executed is needed. We write in Protocol 10 a protocol for creating contracts between vendors and the user to ensure the desired instructions are carried out as specified.

Constraint Program 8 Constraint Programming Distribution

Input:

- $Q = [q_1, ..., q_n], \forall i, q_i \in \mathbb{N}$ a list of the number of qubits for each QPU in the system
- $A \in \mathbb{N}$ the number of qubits in the Ansatz
- $m \in \mathbb{N}$ the number of Ansätze to fit

Variables:

- $x_{ij} \in \{0, ..., A\}$: The number of qubits from Ansatz $0 \leq i \leq m$ placed on QPU j
- $y_{ij} \in \{0, 1\}$: The QPE qubit for Ansatz i on QPU j
- $z_{ijk} \in \{0, 2\}$: The number of qubits used to split Ansatz i between QPUs j and k

Objective Functions:

$$\text{maximize } \textstyle\sum_{ij} x_{ij}, \text{ minimize } \sum_{ijk} z_{ijk}$$

Constraints:

1. There's only one QPE qubit per Ansatz:

$$\sum_{j=1}^{n} y_{ij} = 1, \quad \forall i \in \{1, ..., m\}$$

2. If the Ansatz is split, then both QPUs use qubits:

$$z_{ijk} = z_{ikj}, \quad \forall i \in \{1, ..., m\}, j, k \in \{1, ..., n\}, j \neq k, j < k$$

3. Ansatz is completely covered with one QPE qubit:

$$\sum_{j=1}^{n} x_{ij} + y_{ij} = A + 1, \quad \forall i \in \{1, ..., m\}$$

4. Qubits allocated do not exceed the number of qubits on the QPU. Note we can recycle the splitting qubits for multiple splits of the same Ansatz.

$$\sum_{i=1}^{m} x_{ij} + y_{ij} + \max_{\substack{k \in \{1,...,n\} \\ k \neq j}} z_{ijk} \leq q_j, \quad \forall j \in \{1, ..., n\}$$

5. The Ansatz fits on one QPU or it is split:

$$\max_{i \in \{1,...,A\}} x_{ij} = A \wedge \sum_{j=1}^{n} z_{ijk} = 0 \ \vee \qquad \forall i \in \{1, ..., m\}$$
$$\left|\{x_{ij} : j \in \{1, ..., n\}, x_{ij} \neq 0\}\right| - 1 = |\{z_{ijk} : j, k \in \{1, ..., n\}, j \neq k, z_{ijk} = 2\}|/2,$$

6. The QPE qubit exists on a QPU with Ansatz qubits:

$$\exists j \in \{1, ..., n\} \ x_{ij} \neq 0 \wedge y_{ij} \neq 0, \quad \forall i \in \{1, ..., m\}$$

Algorithm 10 Local to Distributed Circuit

Input:

- A circuit representation of unitary U where U is a list of gates. Each list represents a layer in the circuit. Gates have the form $Gate(\text{ID})$ or $CONTROL(G, \text{ID}_1, \text{ID}_2)$ where G is the gate to applied using control qubit with ID_1 and target qubit ID_2. The ID in the form (i, j) where i is the QPU and j the qubit on that QPU.
- A qubit layout map $qubitMap$ on a collection ID tuples of the form $(ID_1, ID2)$.

Output: An equivalent circuit that accommodates non-local controlled gates.

DistributedRemapper$(U, qubitMap)$:

$remappedCircuit \leftarrow [[\,]]$ ▷ Add a placeholder layer in case an extra first layer is needed
for $layer_l \in U$ **do**
  $modifiedLayers \leftarrow [[\,] \times 8]$
  for $gate \in layer_l$ **do**
    if $gate$ **is** $CONTROL$ **then**
      $((i, j), (s, t)) \leftarrow gate$.**qubits**
      if $i = s$ **then**
        $modifiedLayers[0]$.**add**$(gate)$ ▷ Same QPU, no need to non-localize
        continue
      if Entanglement is not established between QPUs i and s **then**
        $remapping_{l-1}$.**add**$(Ent((i, e_1), (s, e_2))$ ▷ Add ent. gen. as previous layer
      $modifiedLayers[0]$.**add**$(CNOT((i, j), (i, e_1)))$
      $modifiedLayers[1]$.**add**$(c_i \leftarrow measure((i, e_1)))$
      $modifiedLayers[2]$.**add**$(c_s \leftarrow classicalCommunication(i, s, c_i))$
      $modifiedLayers[2]$.**add**$(classicalCtrlX(c_i, (i, e_1))$
      $modifiedLayers[3]$.**add**$(classicalCtrlX(c_s, (s, e_2))$
      $modifiedLayers[4]$.**add**$(Control - G((s, e_2), (s, t))$
      $n \leftarrow 0$
      ▷ Get series of control gates with same control qubit, target qubits on QPU s
      for $seriesGate \in$ **GetSeriesCGates**$(U, layer_l, s, (i, j))$ **do**
        $n \leftarrow n + 1$
        $((\_, \_), (\_, t')) \leftarrow seriesGate$.**qubits**
        $modifiedLayers$.**add**$([\,], 4 + n)$ ▷ Add empty list at index $4 + n$
        $modifiedLayers[4 + n]$.**add**$(Control - G'((s, e_2), (s, t'))$
        remove $seriesGate$ ▷ No need to distribute in next iterations of parent loop
      $modifiedLayers[5 + n]$.**add**$(H(s, e_2))$
      $modifiedLayers[6 + n]$.**add**$(c_s \leftarrow measure(s, e_2))$
      $modifiedLayers[6 + n]$.**add**$(classicalCtrlX(c_s, (s, e_2)))$
      $modifiedLayers[6 + n]$.**add**$(c_s \leftarrow classicalCommunication(s, i, c_s))$
      $modifiedLayers[7 + n]$.**add**$(classicalCtrlZ(c_i, (i, e_1))$
    else
      $modifiedLayers[0]$.**add**$(gate)$
  $remappedCircuit$.**addAll**$(modifiedLayers)$ ▷ Assume empty layers are ignored
return $remappedCircuit$

Algorithm 11 GetSeriesControlGates

Input:
- U the circuit as described in Algorithm 10
- $layerl_l$ the current layer to decompose in U
- s the QPU for the target qubit
- (i, j) the control qubit

Output: The series of gates directly following from layer l that are control gates with with control qubit (i, j).

GetSeriesCGates$(U, layer_l, s, (i, j))$:

1: $layers \leftarrow \{layer_{l+1}, ..., layer_n\} \subseteq U$ ▷ Skip current layer
2: $gates \leftarrow [\,]$
3: **for** $layer_s \in layers$ **do**
4: **for** $gate \in layer_s$ **do**
5: **if** $gate$ **is** $CONTROL$ and $gate = (\_, (i, j), (s, \_))$ **then** ▷ Same control and target qubit
6: $gates$.**add**$(gate)$
7: **else**
8: **return** $gates$
9: **return** $gates$

Algorithm 12 Distributed α-QPE

Input:
- S: A schedule (defined in Definition 19) providing the qubit mapping of an Ansatz of size n for p Pauli strings on m QPUs.
- $A = [a_1, ..., a_n]$: The vector constants for each Pauli string.
- $U = [U_1, ..., U_n]$: The circuits for α-QPE
- $P = [P_1, ..., P_n]$: The Pauli operators associated with each U_i

Output: The value of the expectation value estimations of the p Paulis

Distributed α-QPE(S, A, U, P):

1: $estimates \leftarrow [\,]$
2: **for** $r \in S$ **do**
3: $roundOfEstimates \leftarrow$ **RunAQPERound**$(S(r), U)$
4: $estimates$.**addAll**$(roundOfEstimates)$ ▷ Order of estimates is fixed
5: **return** $A \cdot estimates$ ▷ Return the scalar product

RunAQPERound$(S(r), U)$:

1: $estimates \leftarrow Array(|S(r)|)$ ▷ Initalize $|S(r)|$ length array
2: **in parallel for** $p \in S(r)$ **do**
3: $success \leftarrow$ Bound $|\langle\psi(\lambda)|P_p|\psi(\lambda)\rangle|$ away from 0 and 1 using [138, Appendix C, Stage I]
4: **if** $success$ **then**
5: Perform [138, Appendix C, Stage II] to collapse $|\psi\rangle$ to either $|\phi\rangle$ or $|-\phi\rangle$
6: $estimates[p] \leftarrow$ Perform α-QPE using distributed circuits with $c - U_p$ or $c - U_p^\dagger$ depending on collapsed $|\psi\rangle$
7: **else**
8: $estimates[p] \leftarrow$ Estimate $\langle\psi(\lambda)|P_p|\psi(\lambda)\rangle$ with statistical sampling using constant distribution circuits using the Ansatz distribution p
9: **await all**
10: **return** $estimates$

Algorithm 13 Distributed α-VQE

Input:

- A list of QPU sizes $Q = [q_1, q_2, ..., q_m]$
- H the Hamiltonian $H = A \cdot P = \sum_{i=1}^{n} a_i P_i$
- $R(\lambda)$ The Ansatz preparation circuit

Output: An estimate for $\langle\psi(\lambda)|H|\psi(\lambda)\rangle$, $|\psi(\lambda)\rangle$ the state prepared by circuit $R(\lambda)$.

Distributed α-VQE$(Q, H, R(\lambda))$

$q \leftarrow$ Number of qubits needed for $R(\lambda)$
$p \leftarrow$ Number of Paulis for H
$S, map \leftarrow$ Ansatz schedule from an algorithm proposed in Section 3.3.3
$dR(\lambda) \leftarrow$ **DistributedRemapper**$(R(\lambda), map)$
$dR(\lambda)^\dagger \leftarrow$ **DistributedRemapper**$(R(\lambda)^\dagger, map)$
$d\Pi \leftarrow$ **DistributedRemapper**(Π, map)
$c - \Pi \leftarrow$ Add control connections to $d\Pi$ from pre-allocated α-QPE qubit
$c - d\Pi \leftarrow$ **DistributedRemapper**$(c - \Pi, map)$
for $P_i \in P$ **do**
  $dP_i \leftarrow$ **DistributedRemapper**(P_i, map)
  $c - dU_i \leftarrow$ Combine distributed circuits $dR(c - d\Pi)dR^\dagger dP_i dR(c - d\Pi)dP_i dR^\dagger$
$c - dU \leftarrow [c - dU_1, ..., c - dU_n]$
$dP \leftarrow [dP_1, ..., dP_n]$
$\langle\psi(\lambda)|H|\psi(\lambda)\rangle \leftarrow$ **Distributed α-QPE**$(S, A, c - dU, dP)$
return $\langle\psi(\lambda)|H|\psi(\lambda)\rangle$

Algorithm 14 Distributed Scheduler

Input:

- $QPUs$ the collection of QPUs in the system
- $C = \{l_1, ..., l_n\}$ the circuit to schedule as a series of layers where each $l_i = \{g_1, ..., g_m\}$.
- $gateTime$ a mapping of gate names to time the gate takes to execute for each QPU.

Output: A schedule of gate operations for each QPU to run.

$layerEndtime \leftarrow 0$
$gateSchedule \leftarrow [\,]$
for $l_i \in C$ **do** ▷ Make a first pass schedule based on the layers of the circuit
  for $g_j \in l_i$ **do**
    $gateSchedule$.**add**$((g_j, \textbf{QPUs}(g_j), layerEndtime))$
  $layerEndtime \leftarrow \max_j(gateTime(g_j)) + layerEndtime$
$QPUSchedules \leftarrow \{\}$
for $QPU \in QPUs$ **do** ▷ Split the schedules so each QPU has its own
  $QPUSchedule \leftarrow [\,]$
  for $step \in gateSchedule$ **do**
    if $QPU \in \textbf{QPUs}(step) \wedge |\textbf{QPUs}(step)| = 1$ **then**
      $QPUSchedule$.**add**$(step)$
    else if gate$(step) = GEN\_ENT$ **then**
      if $QPU = \textbf{QPUs}(step)[0]$ **then** ▷ Sender QPU
        $QPUSchedule$.**add**$(SEND\_ENT[\textbf{QPUs}(step)[1], \text{qID}], \textbf{time}(step))$
      else ▷ Receiver QPU
        $QPUSchedule$.**add**$(REC\_ENT[\textbf{QPUs}(step)[0], \text{qID}], \textbf{time}(step))$
    else ▷ The other non-local gate is classical transmission
      if $QPU = \textbf{QPUs}(step)[0]$ **then** ▷ Sender QPU
        $QPUSchedule$.**add**$(SEND\_CLA[\textbf{QPUs}(step)[1], \text{cID}], \textbf{time}(step))$
      else ▷ Receiver QPU
        $QPUSchedule$.**add**$(REC\_CLA[\textbf{QPUs}(step)[0], \text{cID}], \textbf{time}(step))$
  $QPUSchedules[QPU] \leftarrow QPUSchedule$
return $QPUSchedules$

Protocol 9 Entanglement Validation

Vendor 1

1: Generate N entangled pairs with Vendor 2
2: Measure all of the owned halves
3: Send $t < N$ randomly selected measurements to Vendor 2 without stating which qubits were measured
4: Receive t bits from Vendor 2
5: If t bits not received, abort
6: Send positions of measurements to Vendor 2
7: Receive positions of measurements from Vendor 2 and compare measurements
8: Send acknowledgment if comparison passes, else send negative acknowledgment

Vendor 2

1: Generate N entangled pairs with Vendor 1
2: Measure all of the owned halves
3: Send $t < N$ randomly selected measurements to Vendor 1 without stating which qubits were measured
4: Receive t bits from Vendor 1
5: If t bits not received, abort
6: Send positions of measurements to Vendor 1
7: Receive positions of measurements from Vendor 1 and compare measurements
8: Send acknowledgment if comparison passes, else send negative acknowledgement

Protocol 10 Contract Creation Protocol

User

1: Assume it is known how many qubits exist on each available QPU for each QPU provider
2: Generate non-local circuits
3: Request gate and classical communication latency times of gates from each QPU provider
4: Generate a gate execution schedule using Algorithm 14
5: Send executions schedules for the respective QPUs to the respective vendor along with current system time
6: Await confirmation messages from all vendors
7: If any vendor responds negatively, broadcast abort
8: Gather all latest start times and broadcast start time as the minimum over all latest start times
9: Await acknowledgements from all vendors, broadcast abort if any do not arrive, otherwise broadcast start signal

Vendor

1: Await request from user for gate times and respond accordingly
2: Await gate execution schedule
3: Validate that instructions can execute within allotted time frame for the user, respond to user if negative
4: If there are instructions with classical communication to an IP address, perform a handshake with other IP, respond to user if negative
5: If there are instructions with entanglement, perform entanglement validation procedure in Protocol 9, respond to user if negative
6: With other IPs, perform clock synchronization, respond to user if negative
7: When all checks pass, respond positively to user with latest possible start time of execution adjusted for user system time difference
8: Await start time confirmation, send acknowledgement to user, and await for final acknowledgement from user

4. Software Frameworks for Quantum Networks

To develop novel applications and protocols for quantum networks, access to a quantum network is necessary to validate and benchmark. Generally, access to a quantum network is hard to come by as 1) Quantum network technology is very much still in development 2) The quantum network technology that exists is not at the quality that it is expected to be in for future networks and 3) Setting up quantum networks is a laborious undertaking. To overcome the lack of physical hardware, the next best approach is to simulate and emulate the networks, so that when the technology catches up, the applications can already be developed and studied to then be adapted later to the hardware, knowing that the applications will work.

In this section, we review three various simulation frameworks. The first, QuNetSim is a high-level quantum network simulator designed for developing and verifying quantum networks. Next, we introduce Interlin-q, a framework built on top of QuNetSim which can be used to perform simulations of distributed quantum computing. And finally, the third framework, QontainerNet, is a merger between QuNetSim and Mininet which can be used to perform simulations of quantum-enhanced communication networks.

4.1. An Overview of Quantum Network Simulators

Section based on the article: "QuNetSim: A Software Framework for Quantum Network"

A detailed list of quantum software libraries hosted at [165] contains approximately 100 different flavors of quantum simulation software. Most of these are directed at simulating quantum computation and circuitry on various hardware configurations with various levels of realism. With regards to quantum networking, as far as we know at this time, there are five publicly available quantum network simulators: SimulaQron [166], NetSquid [43], SQUANCH [167], QuISP [168], and SeQUeNCe [169]. Of these, all libraries are freely available. SQUANCH, QuISP, SeQUeNCe, and SimulaQron are open-source under the MIT license or 3-Clause BSD License. NetSquid uses a license restricting its use for educational and non-commercial research and development purposes. Indeed, some simulators simulate quantum key distribution [170, 171], but these are single-use case simulators and do not allow for arbitrary application layer protocols and we do not compare them to QuNetSim for that reason.

SimulaQron [166] is a simulator that can be used for developing quantum internet software. It simulates several quantum processors that are located at the end nodes of a quantum network and are connected by simulated quantum links. The main purpose of SimulaQron is to simulate the application layer of a network; tasks such as routing are left to the user to implement using their approach if needed. SimulaQron further offers the ability to run simulations across a distributed system, that is, simulations can

be set up to run on multiple computers. What we found slightly lacking in SimulaQron is a way to easily synchronize the parties regarding qubit arrival. A key difference in QuNetSim is that it adds a layer of synchronization. Built into QuNetSim is the approach of acknowledging when information arrives at the receiver. One can more naturally write protocols in a standard way, where one handles the information arriving or not before proceeding. SimulaQron also has hosts which have features like sending qubits, establishing EPR pairs, and sending classical information. To simplify the task of developing protocols on top of existing protocols, we try to include more built-in tasks such as sending teleportation qubits, establishing a GHZ state, and establishing a secret key using Quantum Key Distribution (QKD).

NetSquid [43] is a powerful event-driven quantum network simulator. It can simulate the physical properties of quantum devices like quantum gates and memory, noise, and loss of a quantum channel, and time-dependent quantum state decoherence. NetSquid can be used as a benchmarking tool that can be used to test the robustness of quantum network protocols against the physical and link layer effects of the network. NetSquid uses a modular approach for network configuration, allowing users to customize their simulations in many ways. A key strength of NetSquid is its ability to incorporate the time-dependent effects of quantum systems. With these features comes a layer of added complexity that falls to the user. To use NetSquid to its full extent, the user should have a fairly good understanding of the hardware structure of a quantum network. QuNetSim is designed to develop quantum protocols and test them for correctness over networks, and not for benchmarking against the physical properties of the network more easily. This greatly reduces the need to understand quantum networks at a deep level but does remove the ability to benchmark quantum networking protocols against any hardware specifications. One could use both NetSquid and QuNetSim together to develop their ideas as a first prototyping step with QuNetSim and then benchmark the protocols in NetSquid to see how it performs with the added physical models.

SQUANCH (Simulator for Quantum Networks and Channels) [167] achieves similar functionality as SimulaQron but allows for customizable physical layer properties and error models. It allows for creating simulations of distributed quantum information processing that can be parallelized for more efficient simulation. It is designed specifically for simulating quantum networks to test ideas in quantum transmission and networking protocols. SQUANCH can be used to simulate many qubits and can allow a user to add their error models, which we think allows for a more realistic quantum network simulator. SQUANCH also allows one to separate the quantum and classical networks of a complete network and as well as adding length-dependent noise to the channel. A key difference between SQUANCH and QuNetSim is that in SQUANCH, as far as we know, a node can run one set of instructions at a time and not more in parallel. This may not be so restrictive, but in multiparty protocols, it may become challenging to develop all of the behavior in one set of instructions. QuNetSim allows one to develop multiparty protocols one at a time and run them in parallel. Further, synchronization between parties is again potentially an issue with SQUANCH. QuNetSim gives each Host an addressable quantum memory such that given an ID, they can fetch a qubit and can manipulate it as desired. In SQUANCH one should initialize their qubits before the start of the simulation whereas with QuNetSim qubits are initialized at run

time. We think this adds more flexibility when writing protocols and allows for more natural logic in the code.

QuISP (Quantum Internet Simulation Package) [168] is also an event-driven simulation of quantum repeater networks. The goal of QuISP is to simulate a full Quantum Internet consisting of up to 100 networks of up to 100 nodes each and 100 qubits at each node. Its focus is on protocol design and studying emergent behaviors of complex, heterogeneous networks at a large scale while keeping the physical layer as realistic as possible. In comparison to QuNetSim, QuISP uses a different approach regarding how qubits are simulated. The quantum state vector of the qubits in the system is not represented. To simulate a large-scale network as QuISP does it is not possible to store the state information of many qubits mutually entangled, as the size of this vector grows exponentially quickly in the number of entangled qubits. Instead, QuISP stores the errors applied to the qubits which simplify the qubit data structure. In QuNetSim, to get a better sense of the network's effects, we do provide the qubit state vector, albeit with the trade-off that large-scale simulations that are possible with QuISP are not with QuNetSim. QuISP also follows "RuleSet" paradigm for programming the logic of the network [172]. QuNetSim uses standard Python programming to define the actions in the network, in line with the goal of simplicity.

SeQUeNCe [169] is a discrete-event quantum network simulator as are QuISP and NetSquid. It is customizable, so users can define the network topology, the hardware parameters of the nodes, and the actions the nodes take regarding storing qubits. SeQUeNCe has physical models built in and follows a modularized design with cross-module communication to allow for flexibility of the simulation. In this regard SeQUeNCe and NetSquid are the most similar in the collection. In comparison to QuNetSim, again the main difference is QuNetSim is not a discrete-event simulator and focuses on simplifying protocol development rather than benchmarking.

4.2. QuNetSim: A Software Framework for Quantum Networks

Section based on the article: "QuNetSim: A Software Framework for Quantum Network"

A quantum network is a network of physical devices that can transmit quantum information and distribute quantum entanglement amongst themselves. As developments are made towards realizing a standardized quantum internet, the first stages of which are likely to be available in the near future [64, 173], there is a stronger need to be able to efficiently develop and test quantum networking protocols and applications. Recently, there has been much effort into developing quantum simulation software for quantum computing [165], whereas simulation software for quantum networks has received considerably less attention and there are very few available tools that exist capable of addressing a need for emulation. QuNetSim is especially suited because it provides built-in tools for asynchronous tasks. A need has therefore arisen for advanced quantum network simulation and emulation tools. The initial release of QuNetSim fills this need by providing a lightweight, easy-to-use, open-source, quantum

network simulation framework that is in principle also able to interact with laboratory equipment.

As it has been done for the classical Internet with, for example, the NS-3 [174] and Mininet [175] platforms, work towards a similar open-source simulation platform with many contributors should be developed for quantum networking. Although there have been developments in quantum network simulators, as we discuss in Section 4.1, presently, we think there is a gap between network simulators that work on a low level and network simulators that are easy to use and can be used in a testing phase of protocol development for quantum networks.

The goal for QuNetSim is to provide a high-level framework that allows users to quickly develop quantum networking protocols without having to invest time in purely software-related tasks, like managing threading and synchronization, writing thread-safe logic, or repeatedly implementing basic protocols that can be used as building blocks for new protocols. QuNetSim further aims to provide an open-source simulation platform offering a high degree of freedom to attract contributors, enabling even more simulation possibilities. As a consequence of meeting these goals, the learning curve needed to begin developing protocols for quantum networks is flattened, since QuNetSim makes it easier to write them. QuNetSim allows users to create examples of quantum networking protocols that are along the lines of how protocols are developed as a first stage for research papers, which helps to develop protocols as well as educate students about quantum networks. In the examples we provide, we see how there is an almost one-to-one correspondence between how protocols are written in quantum network protocol research papers and how simulations are developed in QuNetSim. In future releases, we aim to subsequently professionalize the emulation capabilities of QuNetSim. Although the currently limited availability of off-the-shelf hardware components for quantum networks leaves the network engineer with many unfulfilled needs, QuNetSim offers, at least in principle, already today the possibility for using hardware in the loop. Future emulation goals implement this behavior.

4.2.1. Overview of QuNetSim

The current main purpose of QuNetSim, as the name suggests, is to simulate quantum networks. To this end, we aim to allow for the writing and testing of robust protocols for multi-hop quantum transmission with various network parameters and configurations. QuNetSim allows users to create network configurations of nodes connected via classical or quantum links and then program the behavior of each node in the network as they choose. To simulate the asynchrony of networks, QuNetSim runs in a multi-threaded environment using first-in-first-out queues for packets. The generally challenging programming problem of managing multiple threads is simplified as QuNetSim provides the methods to synchronize the nodes in the network even when they are all performing their actions independently and asynchronously. Further, QuNetSim comes with many built-in protocols such as teleportation, EPR generation, GHZ state distribution, and more, over arbitrary network topologies, that make it easier to develop more complex protocols, using the basic ones as a toolbox. It also provides an easy way of constructing a complex network topology such that one can design and

test routing algorithms for quantum networks.

QuNetSim uses a network layering model inspired by the OSI model [15]. It naturally incorporates control information together with any payload type but is open to modifications where control information is explicitly transmitted separately from the payload. In future quantum network implementations, likely, the exact OSI model layering model will not be used and it could be that new layers are introduced. For example, an already proposed network layering includes a "connectivity layer" between the link and physical layer [16]. We anticipate that the basic concept of layering of classical communication networks will be carried over to quantum networks, so that included will be the layers such as application, transport, and network. Quantum information carriers will be encoded into some form of packets which will then be routed through the network to the desired destination. In its initial form, QuNetSim implements the network layer and assumes link- and physical layers deliver error-free bit- and qubit transmission capabilities. While it is relatively straightforward to model link layer (in the sense of a logical but potentially not error-free qubit channel between two nodes) behavior in QuNetSim, the modeling of a physical layer (in the sense of e.g. a continuous-variable quantum system) is currently not considered as within the scope of QuNetSim. By design, the accurate modeling of lower layers is left to simulators that are better suited for that task.

In Fig. 4.1, we depict the high-level design structure of QuNetSim. At this depth, it resembles a virtual connection between two nodes in a classical network, where "virtual connection" means that node **A** has the perspective that it is directly connected to node **B** even though the information sent from **A** is routed through the network with potentially many relaying hops. In the figure, the two nodes are connected (virtually) by a classical channel, represented by the green lines, and a quantum channel, represented by the red lines. Both modes of communication are processed through the same layering mechanism as the network can route both kinds of information but makes decisions based on the payload of the packets. This allows users to use the same programming logic for sending classical messages as for sending quantum, leaving it to the lower layers to work out the differences.

4.2.2. Assumptions Made in QuNetSim

Although much research is directed at building a quantum internet, or more generally and abstractly a network of nodes with the ability to create, distribute, and store quantum states, currently such a network does not exist nor are the features of a quantum internet yet to be standardized. To build a simulation software framework as general as possible while attempting to keep it simple, we, therefore, make only a few assumptions that we think will most likely be met by future quantum networks.

QuNetSim assumes that both classical- and quantum information transmitted via future quantum networks will use signal processing that is modeled based on quantum mechanics. However, following the principles of network layering, not all information that is available at the lower layers can be accessed by the upper layers. Therefore, the class `quantum_connection` in QuNetSim is assumed to model an error-free logical qubit channel between network Hosts where the physical means of transmitting a

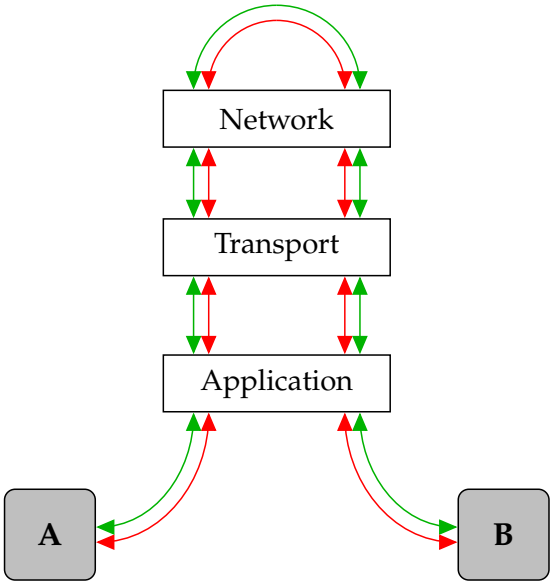


Figure 4.1.: A design depiction of QuNetSim. Here the green line represents a classical channel and the red a quantum channel. QuNetSim attempts to simulate the process of moving both classical and quantum packets through a set of network layers as does the classical Internet. Here Host **A** has a virtual connection to Host **B**, and so all of their communication is processed one layer at a time. QuNetSim does not explicitly incorporate features of the higher layers like the link layer or the physical layer.

qubit necessarily involve also sending – potentially classical – control information such as desired destination node, type of error correction applied, and synchronization information all of which might even rely on an exchange of information via a feedback loop from receiver to sender. In future networks, not every connection between network nodes will necessarily be able to transmit quantum information. Therefore, QuNetSim offers the additional class `classical_connection` which allows the transmission of classical data alone.

Another assumption we make is that quantum nodes will be able to detect that a qubit has arrived in their qubit storage. The practical methods (e.g. heralding) used to achieve this functionality are left open. QuNetSim aims to make it easy to synchronize between Hosts and therefore we allow acknowledgment of the arrival of qubits. Since QuNetSim simulates up to the network layer, we assume that the lower layers of the quantum network will be able to provide protocols to accomplish this. This is different from current quantum key distribution networks that simply measure the qubits as they arrive without storage. The measurement output acts as a herald but, also destroys the incoming photon's quantum state.

QuNetSim is a tool for protocol development in quantum communication networks modeled based on a structure similar to current classical communication networks. The state of the quantum internet is still very primitive and although we attempted to design QuNetSim in a way that safeguards against future physical implementations, it is still difficult to foresee how a quantum internet will be designed and implemented and

which features it will and will not have. We, therefore, do not claim that a simulation implemented in QuNetSim is guaranteed to work in a future quantum internet, but we hope that by using it one can more easily envision and experiment with possible quantum networks.

4.2.3. Limitations of QuNetSim

QuNetSim provides a high-level framework for developing quantum protocols. There are, however, some limitations in the current implementation. QuNetSim relies on existing qubit simulators (see section 4.2.4) which in some cases perform well and in some cases do not. This causes QuNetSim to periodically run more slowly than desired and, because we are using full qubit simulators, where the alternative is error tracking only as it is in QuISP [168], running large-scale simulations can consume much of the user's computer's resources. We have found that QuNetSim works well for smaller-scale simulations using ten to fifteen Hosts that are separated by a small number of network hops. QuNetSim tends to reach its limits when many entangled qubits are being generated across the network with many parallel operations. As this is an ongoing project, we will investigate performance improvements as a priority during future iterations. Moreover, QuNetSim does not aim at perfect physical realism, and so the physical properties of quantum networks are mostly neglected. One can although in principle recover physical realism by integrating QuNetSim with physical devices as described earlier.

4.2.4. Using QuNetSim

In this section, we introduce the key features of QuNetSim for implementing protocols. A full set of documentation is also available [176].

A foundational data structure used in QuNetSim is the Qubit. When a Qubit is created, it is created with a specified Host and gets assigned a unique ID. A qubit is generated by using `Qubit(host)`. Once a qubit is created by a Host, logic gates can be applied to it, it can be stored, or it can be transmitted to another Host. To send a qubit to another Host, one can send it directly or by using teleportation. The two Qubit methods that accomplish this are:

- `send_qubit`
- `send_teleport`

Hosts can easily establish entangled qubits with other Hosts in QuNetsim with two Host methods. Built-in, Hosts can generate EPR pairs with another Host or a GHZ or W-state with many Hosts. To do so, the following Host methods are in place:

- `send_epr`
- `send_ghz`
- `send_w`

Hosts can send classical messages in three ways; they can send an arbitrary string over a classical connection, a binary message via superdense coding, or classically broadcast messages through the network. These are accomplished with the following Host methods:

- `send_classical`
- `send_superdense`
- `send_broadcast`

Sending a message using superdense coding [58] requires that a quantum channel is also available between Hosts, as it requires a pre-established EPR pair.

For synchronization between communicating Hosts, it is sometimes beneficial to wait for acknowledgment from the receiving Host. "Waiting" in QuNetSim implies blocking a line of code. Waiting for an acknowledgment is possible for all sending methods which is done by setting the flag in the methods called `await_ack`. By setting the flag to false, the Host does not wait before executing the actions that follow and there is no thread blocking done. Moreover, it can be specified how long a Host should wait for acknowledgments by setting the `max_ack_wait` Host property. QuNetSim uses real-time for this, so this parameter is the number of seconds to wait. When `await_ack` is false, an acknowledgment is still sent, but there is no waiting. One can further specify that no acknowledgment should be sent by using the `no_ack` flag in sending methods.

Hosts can be programmed to retrieve an incoming classical or quantum message and also wait for it to arrive. When a classical or quantum message arrives, it is stored at the Host in its respective memory structure – there is a distinct memory for classical and quantum information. Hosts have the option to fetch the data from their memories so that actions can be performed on it. These methods are:

- `get_classical`
- `get_data_qubit`
- `get_epr`
- `get_ghz`
- `get_w`

Much like awaiting acknowledgments, Hosts can wait until a message or qubit arrives for a fixed amount of time before proceeding. For each "get" method, there is a parameter `wait=n` where `n` is a floating point number of seconds to wait. For example, `get_epr('Alice', wait=5)` will wait for five (real) seconds for an EPR to arrive from Alice.

In near-future quantum hardware, it is expected that quantum memories will be limited in their ability to store large numbers of qubits. QuNetSim supports limiting the number of qubits stored at a Host. The number of entangled qubits and data qubits can be limited separately or a limit for the combined number of qubits can be set. The Host methods for setting the limits are:

- `set_epr_memory_limit`
- `set_data_qubit_memory_limit`

These parameters will enforce that no more than the set number of qubits can be stored. When the limit is reached, the qubits are lost when they arrive.

To construct a network of Hosts, Hosts methods are provided for adding and removing connections. Connections in QuNetSim are uni-directional and can be either purely classical, purely quantum, or both classical and quantum. For example, if two Hosts are connected by only a classical connection, then qubits cannot be transmitted between the two Hosts. These Host methods are:

- `add_c_connection`

- `add_q_connection`
- `add_connection`

Connections can also be removed at run time with

- `remove_connection`
- `remove_c_connection`
- `remove_q_connection`.

Hosts can eavesdrop on communications. This is accomplished by firstly setting the following Host properties to `true`:

- `c_relay_sniffing`
- `q_relay_sniffing`

From there, a custom Python function can be run to manipulate the payload of the relaying packets. To do this, functions are set to the Host properties:

- `c_relay_sniffing_fn`
- `q_relay_sniffing_fn`

These features allow Hosts to intercept the qubits that pass through them en route and manipulate them by measuring them or performing a unitary operation on them before relaying them onward. With classical messages, the Hosts can manipulate the classical payloads, changing the messages for example.

Hosts are initialized in an "off" state and so to start Hosts one uses the `start` Host method. Once started, Hosts can run custom protocols using the `run_protocol` method taking any Python-written function as a parameter. We will see examples of this in the next section. A running protocol can be blocking or non-blocking. The Python thread object containing the running protocols is provided and can be handled by the user as desired.

Building the network topology is also a critical part of every simulation. QuNetSim uses a `Network` singleton object to abstract the classical and quantum networks. Once the network topology is established between the Hosts, Hosts are added to the network using the Network method `add_host`. The network builds a graph structure using the connections of the Host to be used for routing. As we will see in the next section, this involves adding Hosts to the network and then calling the Network method `start`.

Networks can be programmed to run custom routing functions for the classical and quantum messages separately. By default, the network uses shortest-path routing for packets in the network, but by setting the following Network properties:

- `quantum_routing_algo`
- `classical_routing_algo`

the Network will use any other routing algorithm. The output needs just be an ordered list which is the path to the receiver. The network can re-route at each hop as well by setting the `use_hop_by_hop` flag accordingly.

Overall, with these features, one can already develop a variety of protocols with varying network topologies. Further, by programming eavesdropping Hosts, one can easily test protocols against malicious third parties, programming various attacks, as is the norm in the development of QKD protocols.

Quantum Backends

QuNetSim relies on open-source qubit simulators that we use to simulate the physical qubits in the network. At the current stage, we are using three-qubit simulators that each have their benefits: CQC/SimulaQron [19], ProjectQ [177], EQSN [178], and recently QuTiP [179]. EQSN is the default backend and it is created by the TQSD group. Users are free to change the backend of QuNetSim to use different qubit simulators and we also explain how new backends can be easily added in our full documentation [176].

Other than SimulaQron/CQC, using these quantum simulators independent of QuNetSim removes the ability to easily mimic transporting of quantum information in a network setting. For example, one can indeed implement a simulation of quantum teleportation in ProjectQ. With only a quantum circuit alone, which ProjectQ uses for simulation input, one may not immediately know that the circuit represents a multi-party network protocol. In QuNetSim, one program multiple entities independently, clarifying the network aspects. With the circuit model, there is also no aspect of waiting and eavesdropping, both of which are available in QuNetSim. SimulaQron most closely resembles QuNetSim, but QuNetSim adds a lair of synchronization, such as waiting for arrivals and acknowledgment to the Hosts, on top of SimulaQron and adds additional network features as mentioned in the earlier section.

Each backend has advantages and disadvantages and we give a general benchmarking overview in Table 4.1. The advantage of using ProjectQ is that it is the fastest in terms of the run-time of the three implemented backends. We have found, however, that it tends to slow when there is a high volume of qubit entanglement and is less robust when protocols run for longer durations. ProjectQ is not designed to run in a multi-threaded environment and so we observed run-time errors when many measurements are made on qubits in multiple threads. We predict that there could be some additional thread synchronization done here to prevent these errors. Generally, these are not major issues and can be avoided by slowing the simulation slightly using the delaying mechanisms built-in to QuNetSim. EQSN and CQC both work well in terms of threading and processing speed, but are indeed slower in speed than ProjectQ. They are generally more reliable under many qubit operations and tend not to be affected as quickly when many entangled states are being generated. QuTiP currently shows the best performance. It is optimized in terms of representing a density matrix with as little data as possible as well as not using a single matrix to represent the entire quantum system. This allows for generating many qubits, even with bi-partite entanglement, efficiently. Overall, the user can use the same code for their network simulation and easily change the underlying quantum simulation backend to find the one that best suits their needs. In future iterations, we aim to integrate more qubit backends to allow for an even more diverse set of simulation possibilities.

4.2.5. Design Overview of QuNetSim

QuNetSim aims to allow for the development of simulations that contain enough realism that applications of quantum networks can be developed, tested, and debugged for a proof of principle step. With this in mind, we have designed the software such that

Backend	Teleportation	Superdense	GHZ State
ProjectQ	102 ± 30	82 ± 15	N/A
EQSN	283 ± 33	296 ± 72	2765 ± 245
CQC	301 ± 75	533 ± 70*	215 ± 17*
QuTiP	111 ± 28	92 ± 2	351 ± 40

Table 4.1.: Benchmark of the various backends with different networking tasks. Various protocols are run over networks with ten hops between the end nodes. Listed is the average duration in milliseconds to run the specified protocol 10 times repeating the procedure 30 times over to collect the samples. The GHZ state is distributed to 9 other Hosts from the first Host in the chain. The benchmarking is done using an average laptop with a 2.7 GHz Dual-Core Intel Core i5 processor with Python version 3.6.4. * represents tested with just 2 nodes to work with the benchmarking tool.

we remove the need for the large overhead of setting up new simulations and added built-in features that are potentially repeated across many simulations. Another design aspect we aim for is that a prior deep level of understanding of quantum networks and software development should not be required to use QuNetSim. To allow for as many users as possible to develop their applications, we keep the functionality at a high level such that protocols written with QuNetSim are as easy or easier to understand as the protocols written in scientific papers. QuNetSim allows users to merge various protocols which can be easily modified and simulated or chained together to run in parallel or sequential configurations.

The general architecture layout of QuNetSim is depicted in Fig. 4.2. Here there are three network nodes, which are Hosts in QuNetSim. Hosts **A** and **B** are connected via a communication link as are Hosts **B** and **C**. As in a classical network, Hosts run such that they sit idle awaiting any incoming packets and then act when packets arrive, or optionally they can be programmed to act during idle periods. Hosts in QuNetSim run applications asynchronously and transfer quantum and classical messages to other Hosts in the network. In the figure, Host **A** runs an application that sends a packet to Host **C**. Host **A** has no direct connection to Host **C**, and therefore its information must be routed through **B** to arrive at **C**. In a layered network architecture, since Host **A** is running on the application layer, it should not be concerned with how the information arrives at **C**, and the transport layer has to prepare metadata of the application's action and the network layer to route the information to **C** if a route exists.

The transport layer prepares the information sent from **A** for the network by encoding necessary information in a packet header such as the sender and receiver IDs, the protocol, and the packet sequence number. As the layering of quantum networks is not specified yet, the transport layer of QuNetSim has only a few responsibilities. For example, it can be used to ensure the availability of pre-established entanglement between network Hosts before executing protocols such as dense coding or teleportation. The transport layer processes incoming and outgoing packets as a layer between the network and the Hosts. Defined in the transport layer is a set of protocols so that packets are encoded and decoded properly. Once information is encoded into a packet,

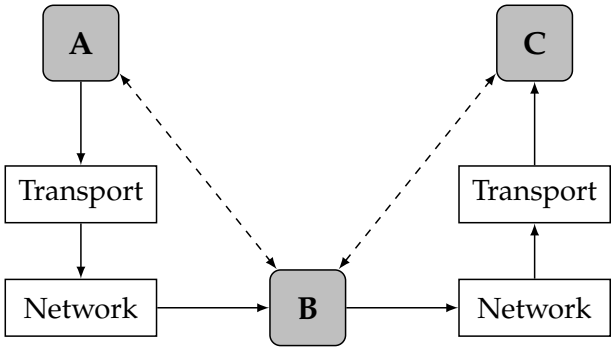


Figure 4.2.: An example of a communication process in QuNetSim with three Hosts. In this example, there are three Hosts, **A**, **B**, and **C**. Hosts **A** and **B** are connected via a 2-way channel (represented by the dashed line), as are Hosts **B** and **C**. When Host **A** executes an application that transmits information to Host **C**, since there is no direct connection, the information must first be routed through Host **B**. QuNetSim uses a layered approach like the Internet. First, application data is filtered through a transport preparation layer so that the information packet is prepared for the network. From there, the transport layer packet is put into the network. The network also encodes the packet with its header information and begins to route the packet through the network. The network packet is moved first to Host **B**, and Host **B** relays the data to Host **C** to complete the transmission.

the transport layer moves the packet to the network. In the current state of QuNetSim, the transport layer is not configurable, other than the ability to disable the check for EPR pairs for the built-in teleportation and superdense coding features. In the next major release of QuNetSim, we aim to allow users to define their packet structures and have a more configurable transport layer behavior.

Once the packet is added to the network the network layer routes it to the destination receiver. The path from **A** to **C** is through **B** and so a transport layer packet is encoded in a network packet and then moved through **B** to **C**. When Host **B** receives a packet from **A**, since it is not the intended receiver, it relays the network packet onward. Finally, when the network packet arrives at **C** the packet is unpacked in the network layer and again in the transport layer. The payload can then be processed according to the decoded information in the header. This separation of responsibility per layer is a fundamental element of the QuNetSim design. The network of QuNetSim behaves much like the network on the Internet with some key differences. In QuNetSim, the network is composed of two separate but parallel networks, one for quantum information and one for classical information. When quantum information is sent from a Host, the network routes it through the quantum links in the network as it does for classical information. Another responsibility of the network layer that differs from the classical setting is that the network layer is responsible for establishing an EPR pair between nodes that do not share a direct connection via an entanglement swapping routine. It can trigger a chain of Hosts to perform an entanglement swapping so that in the end, the sender and the intended receiver will share an EPR pair.

QuNetSim does not currently go above the network layer in terms of the simulation of quantum networks. As more features are developed, the code structure allows us to replace pieces that we currently omit, such as link layer functionality. We do, however, allow the user to integrate their qubit channel models, and in subsequent versions will incorporate this into the design such that these things will be easy to change.

We now discuss the software implementation of QuNetSim. The QuNetSim framework is developed in the Python programming language [180] as a software library. It uses Python's multi-threading library to simulate the asynchronous network behavior. Each Host has a processing queue and runs in a thread so that when a Host performs an action the actions run in a first-in-first-out ordering. These actions are then processed in the transport layer where the header information contains sender and receiver information, along with information on how to process the payload. The processed packet is put into the network for routing. The network acts like a Host where it has a packet queue that is being monitored for changes. Once a packet is put into the queue, it is analyzed and processed.

In the network packet can be a signal to generate entanglement between Hosts before executing a given task. Since the swap procedure—as mentioned above—consumes a large number of communication resources, and since this reflects on the run-time of QuNetSim, the default behavior of QuNetSim is to first check if it is possible to form a SWAP chain from sender to receiver. If it is possible to form an entanglement swap chain, then all corresponding Bell measurements and the transmission of classical messages associated with executing the entanglement swap protocol will not be performed explicitly. Instead, only the result of the protocol (deletion of the consumed EPR pairs and establishment of an EPR pair between sender and receiver) is directly provided to the backend. This way, the execution time of the already well-known procedure is not contradicting the design goal of efficient protocol testing. If no EPR pair is needed the payload is checked for the type of data it contains. A network in QuNetSim contains two directed graphs to represent disjoint networks for the respective data types. If the packet payload is classical data, it is routed over the classical network and if it is quantum data it is routed via the quantum network. By default, the routing algorithm is the shortest path, but it can be changed via parameter settings. We see an example of how this is done in Section A. Currently, in QuNetSim there are simple error models available that can be turned on as an optional feature. The error models are configured as network property and include packet loss and Pauli errors applied to Qubits en route. A more complex treatment of the link layer will be considered in future iterations of QuNetSim.

Finally, the packet arrives at the receiver Host after being routed in the network. The payload is then processed at the receiver side according to the header information. Once processed, the classical or quantum data is stored in either the classical memory of the Host or in one of the two quantum memories. There are two quantum memories, as this allows users to more easily distinguish between qubits that arrive as qubits encoded by the sender or qubits specifically generated using one of the built-in entanglement generation Host methods.

In summary, QuNetSim implements a layered model of component objects much like the OSI model [15]. The Host and network components are implemented using

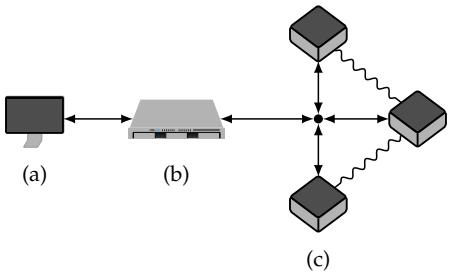


Figure 4.3.: The architecture used in Interlin-q: The client (a) constructs a circuit input designed for a monolithic quantum computer and sends it to the controller (b). The controller remaps the circuit for the pre-defined but arbitrarily distributed system and generates the execution schedule. Once complete, the respective control instructions are sent to the quantum computers in (c). The quantum computers execute the schedule and send the results back to the controller who processes the results to send back to the client.

threading and observing queues. The queues are monitored constantly and queue changes trigger an event. Extensive use of threading allows each task to wait without blocking the main program thread, which simulates the behavior of sending information and waiting for an acknowledgment or expecting information to arrive for some time from another Host.

4.3. Interlin-q: A Distributed Quantum Computing Simulator

Section based on the article: "Quantum Algorithms and Simulation for Parallel and Distributed Quantum Computing"

In this section, we introduce our novel distributed quantum algorithm simulation framework Interlin-q. Interlin-q is a Python framework allowing the simulation of networked quantum computers executing a quantum algorithm distributed over a user-specified topology. The goal of Interlin-q is not to perform high-performance computing, but rather to test and verify the necessary steps of distributing circuits and generating control instructions. The tool is meant for validating these tasks by executing them in this simulated environment and collecting various statistics regarding the number of resources. We begin by summarizing the architecture of Interlin-q and explaining its inner workings. In the next section review demonstrations.

4.3.1. Design Principles

The simulated architecture of Interlin-q consists of three types of network nodes: The client node, the controller node, and the computing node. In Fig. 4.3 is a depiction of the assumed architecture of Interlin-q. We base the design principles of Interlin-q on these node types, as a distributed quantum system will likely follow such an architecture. The responsibilities of each node type are as follows:

Client Node: The client is a user terminal node where the user of the system inputs the program information in a similar way as described in the previous section. A monolithic circuit is specified along with the merging function. This information is then passed forward to the controller node which continues the execution process. Once execution is complete, the controller node returns the results of the program after performing the merging function to the client node. This isolation of the client node removes the need for the user of the system to know the underlying architecture of the distributed quantum computer.

Controller Node: The controller node is the conductor which orchestrates the distributed system of quantum computers. It is aware of the distributed topology and the quantum processor architecture of each node in the network and therefore can define allocations for qubits for circuit execution. Once the program information from the client node is provided, the controller node can perform Alg. 8 to prepare for execution and awaits outputs from each program. Once outputs are received, the controller merges the results accordingly and responds to the client node.

Computing Nodes: The computing nodes execute quantum algorithms based on the instructions provided by the controller node. A computing node can prepare and perform logical operations on qubits, store qubits, and shared EPR pairs, and also perform any classical post-processing. Computing nodes further can communicate with other computing nodes in the network to share EPR pairs or transmit classical information. Computing nodes are networked via both a classical network for transmitting purely classical data and a quantum network for generating entanglement amongst themselves. Networked computing nodes also share a synchronized clock to maintain synchronization, important for two-qubit operations and joint measurements. Once execution is complete, the measurement results are sent back to the controller node.

4.3.2. Simulated Setup and Preprocessing

To implement these design principles, Interlin-q has Python classes to represent the controller and client nodes specified in the previous subsection where the client node is assumed to be the user of Interlin-q. Further, abstractions of circuits are developed for automatic circuit remapping. To create the simulated distributed computing environment, the user first initializes the computing nodes and a controller node. The network is initialized and configured using QuNetSim, thereby defining network nodes and topology. In its current state, Interlin-q will assume that the computing node forms a complete network and that each is connected to the controller node. Because of this, Interlin-q has a built-in function for generating the network where future work involves allowing for various network topologies.

To initialize a computing node using the respective class, one specifies the number of qubits and optionally the duration of the various quantum gates, or gate times, for the gates that the computing node supports. The latter is especially useful when connecting a network of quantum computers each realized using different qubit technologies, for example connecting one QPU based on superconducting with one using trapped ions, where the two technologies differ greatly in gate times. With known gate times, precise algorithm execution schedules can be generated [112]. Within Interlin-q is a simulated

synchronized clock. Each of the computing nodes shares a singleton clock object and function such that each tick of the clock, depending on the execution time of their instruction, perform a specific operation. The set of instructions is generated dependent on the gate times. The computing nodes and the nodes and their qubits are assigned unique IDs to be used for circuit creation.

A controller node is also initialized by the user and the collection of computing nodes is passed as an initialization parameter to the controller. To create a simulation, a user generates a monolithic circuit as a parameter to give to the controller where built-in Interlin-q is a feature to remap it to the distributed computing nodes. In the current state of development of Interlin-q, a custom circuit class is used to both enable the distribution of the circuit but also to simplify the user input process. The circuit model we used is composed of qubits and layers, the standard circuit model for quantum computing. To create a circuit in Interlin-q, a user, acting as the client node, specifies the gates for circuit qubit by qubit. Once the circuit object is created, it is passed to the controller and then automatically distributed based on the network of computing nodes. Built-in to the controller node is the conversion of the circuit model to a layered model, where a new layer is created for each $i \leq w$ where w is the longest sequence of gates in the circuit. With the layering generated, the controller performs an algorithm equivalent to [112, Alg. 3]. To summarize the referenced algorithm, Interlin-q processes the circuit layers one by one and determines if any of the gates are distributed across distinct QPUs. If found, the necessary logic to create an equivalent distributed circuit is filled. This is done repeatedly until all non-local control gates are generated, forming an equivalent distributed circuit.

With the now remapped circuit, the controller node generates an execution schedule, creating the instructions to distribute to send to each computing node. To generate the list of control operations, an equivalent algorithm to [112, Alg. 7] is implemented. For each gate in the remapped circuit, the algorithm maps it to a logical instruction, including any gate parameters, and also marks the control instruction with a timestamp. In practice, the networked QPUs would use a reference clock to execute instructions to maintain the needed synchrony for distributed instructions. This logic is simulated in Interlin-q using a custom shared clock class. The collection of control instructions is given an integer timestamp so that during execution the multi-threaded simulation performs according to the control instruction order. To complete the preprocessing stage, the controller completes the generation of the execution schedules according to Alg. 8 and transmits a broadcast message to the computing nodes.

4.3.3. Execution

Execution begins when the controller node broadcasts the instruction sets to the computation nodes. Because Interlin-q is built on QuNetSim, the simulated environment runs multi-threaded, with each node in the network running in its thread. The computation nodes, therefore, await control messages to perform their tasks. Once the specific scheduling message is received by the computing nodes, they begin carrying out the instructions chronologically depending on the timestamp of the instructions. An instruction is either completely local or can be non-local. A non-local instruction

is performed according to Fig. 3.16, where entanglement is generated between two computing nodes and used to transmit control information. The schedule will be such that at the same time instance, one node will wait for an EPR pair to arrive followed by a classical bit, performing their part of the cat-entangler while the other will send the EPR pair, measure their half and send the results onward. Once the instructions are all carried out by each computing node, the time moves forward by one unit. In reality, this clock will be independent of computing nodes, and gate times will know such that precise time schedules can be generated. When the set of instructions is completed, the computing nodes transmit their measurement outcomes back to the controller and receive a new set of instructions if there are more to receive, otherwise, the controller proceeds to merge the outputs and can output the results to the simulation.

4.3.4. Related Platforms

Related to this project are platforms that use batched circuit execution in a parallelized system. As far as we know, the only such example publicly available as part of the Qiskit Runtime Services offered by IBM is called "circuit-runner" [181]. The circuit-runner service takes as input a collection of un-optimized, pre-compiled quantum circuits and sends them to the IBM cloud service to be executed on their network of quantum computers and simulators. Once arrived, the circuits are optimized and compiled online, and executed on the selected hardware backend. Once execution is complete the output information for the circuits such as measurement results, duration, and more is sent back to the user. This sequence of steps for executing batch circuits is much like the steps for how one uses Interlin-q. Indeed, behind the scenes, IBM's circuit-runner service could be using distributed quantum computing when it becomes available and run much like the structure Interlin-q is built on. In terms of using a circuit-runner for running parallelized quantum circuits, one could make use of the paradigms introduced in Section 3.3.1. Where Interlin-q differs is that it opens the "black-box" into how the quantum computers are interacting behind the scenes. By adding a simulated quantum network, a user can investigate precisely how a networked quantum computer executes a distributed quantum program.

Interlin-q is built on top of the quantum network simulator QuNetSim [56] mainly due to its real-time simulation design. Indeed other quantum network simulators exist but are built more towards simulating the hardware properties of quantum networks rather than a focus on application development. Other than QuNetSim, another viable quantum network simulation platform is SimulaQron [166] which also runs in real-time, the key differences which are detailed in [56]. We focus on real-time simulators because of the key design principle of Interlin-q which is the shared clock. The shared clock controls the execution of each thread executing instructions, which is more aligned with a distributed quantum system, rather than knowing the execution schedule ahead of time as discrete-event simulators do, such as in NetSquid [182] and SeQUeNCe [183]. Indeed, using such discrete-event-based simulators for benchmarking would be a valuable extension of Interlin-q, but our current focus is on the verification of distributed quantum algorithms and estimating their resources.

4.4. ComNetsEmu with QuNetSim

Section based on the article: "Integrating Quantum Simulation for Quantum-Enhanced Classical Network Emulation"

To simulate the transmission protocol proposed in the previous section, we have integrated a quantum link layer and physical layer simulator with the classical network emulator ComNetsEmu [53]. From an implementation viewpoint, the quantum lower layers act as a sort of "man-in-the-middle" between the network nodes communicating. This software logical entity intercepts network packets en route and converts the incoming data to binary data as it would for a link layer. Using the binary format, we simulate the qubit data frame construction described in the previous section and transmit the qubits using QuNetSim [56]. With QuNetSim, we model a quantum point-to-point channel. The end nodes of this channel perform quantum tasks such as qubit encoding, transmission, storage, retrieval, and decoding. The two end-nodes run in a multi-threaded QuNetSim application running Protocol 1.

To integrate ComNetsEmu and the QuNetSim simulation, we use a bridge interface responsible for moving traffic generated in ComNetsEmu to the QuNetSim application and back again. With ComNetsEmu, the quantum simulation part of the program is "containerized" in a Docker environment. Docker uses "containers" to isolate code and all its dependencies in virtual computing environments [53]. ComNetsEmu simplifies the deployment of containers specifically to simulate communication networks. One can configure the IP addresses of these containers and monitor the incoming and outgoing traffic flow of the container. Using ComNetsEmu, the bridge interface is deployed in a container, which runs QuNetSim as an application and monitors the network traffic like a "man-in-the-middle". The bridge is configured such that it routes all network-layer traffic to a particular queue and it is configured to monitor the incoming traffic arriving in this queue. When packets arrive, they are disassembled into binary strings and processed in the QuNetSim application. The quantum channel running in the QuNetSim application converts the binary strings into data frames constructed as in Fig. 2.14, using Algorithms 1 and 2, and transmits them over the channel to the QuNetSim Host representing the receiver's link layer. Within the QuNetSim application, the destination receives the qubits and decodes the frames according to Algorithm 3 and reconstructs the binary string once the end of the data frame is detected. Once completed, the network-layer packet is reassembled and sent to the bridge to proceed with routing.

4.4.1. Quantum Communication Network Simulation

The area of hybrid classical-quantum networks is a novel domain, with many open challenges remaining. To the best of our knowledge, no tool exists yet with the ability to emulate a classical network with additional quantum technological features. However, in the domain of quantum network simulation platforms, there have been many very recent developments [56, 166, 168, 169, 182]. A detailed overview of these platforms can be found in [56] and [57, Chapter 6.5]. The available quantum network simulators

are developed for a variety of use cases and use various simulation methodologies as well. Here, we consider those which use real-time and not discrete-event simulation. The reason for this is that much of the challenge of programming the logic that reacts to events originating in another system—in this case ComNetsEmu—is handled with a real-time simulator. Real-time network simulations do not necessarily terminate when no events are pending in the simulation, and the CPU resources for keeping the program alive are minimal. Regarding the quantum network simulators that use real-time simulation, there are QuNetSim and SimulaQron [166]. SimulaQron is a quantum network simulator used for developing applications and runs in a multi-processed environment. QuNetSim runs in a multi-threaded environment in a single process and has many of the same features of SimulaQron, with the additional features that come from its design principle of mimicking the idle periods in a network. In contrast to SimulaQron, this brings the feature that nodes in the network are by default always running idly, waiting for incoming events to process. Thus, in QuNetSim, by default, one can program the network nodes to react—in real-time— to incoming events generated in ComNetsEmu, making this software a suitable choice for the integration. An additional feature of QuNetSim is the control over how qubits that are transmitted are accessed at the receiving node. In QuNetSim, the arriving qubits are automatically stored at the receiving node, and the receiving node can then act on the stored qubits while more qubits continue to arrive. As far as we can tell, such a feature would need to be developed as an additional layer in SimulaQron. Overall, this makes QuNetSim a more suitable choice for the type of integration into a network emulator, as it is described here.

4.4.2. Assumptions Made in the Integration

Using an entanglement-assisted classical-quantum network requires new protocols to make use of the additional abilities of the network. We assume: 1) At each interface between a communication link and a network node, qubits can be stored in and retrieved from a quantum memory; 2) In each interface, EPR pairs can be generated with a heralding success signal and transmitted to the corresponding interface at another node via an error-free quantum channel; 3) Each interface in the network can perform a controlled Pauli-X gate (CNOT), Bell measurements, and the Pauli gates on qubits; 4) Interfaces can generate and transmit qubits on demand; 5) Joint measurements at each interface can distinguish the four EPR pairs; 6) Each interface can forward classical messages to other interfaces in the same node. A possible way of achieving these functionalities in a practical demonstration is by using the connectivity and link layers proposed in [16]. The classical upper layers remain unchanged.

With these assumptions, the link layer can transmit a data frame of qubits such that the receiver can interpret whether to measure data or store EPR pairs. The link layer protocol on the receiver side can also reconstruct the classical information packet. The assumptions are tailored such that a) problems at the physical layer are abstracted away from the integration, b) the creation of complex quantum states via the network is limited such that simulations remain feasible c) the maximum flexibility is provided between network interfaces.

Thus the link layer resulting from our integration is capable of demonstrating the GEWI protocol. For this protocol to work, long qubit storage times are required. The recent progress in this domain shows store-and-retrieve rates of 90% for 10 µs for 225 individually accessible memory cells, with a proposal to extend the duration into the seconds range [184], which are in a range that can be useful to benefit from variations in classical traffic load.

This first implementation of a proof-of-concept of a quantum network simulator integrated into a classical emulator trades simplicity and flexibility for practicality: the true benefits of entanglement-assisted data transmission can be expected to arise in very specific transmission systems exploiting quantum effects right at the physical layer [5], and these applications do not necessitate a conversion into qubits as achieved via [16] which is used in our initial integration.

4.4.3. Simulation Setup and Configuration

The simulation is an integration of a quantum link layer and physical layer simulator offered by QuNetSim [56], with the classical network emulator ComNetsEmu [53]. The code for this work can be found at [185]. From an implementation viewpoint, the quantum lower layers act as a "man-in-the-middle" between the communicating network nodes. This software entity intercepts network packets en route and converts the incoming data to binary data, as it would for any link layer. Using the binary format, a qubit data frame is constructed and transmitted using QuNetSim, with which we model a quantum point-to-point channel. The end nodes of this channel perform quantum tasks, such as qubit encoding, transmission, storage, retrieval, and decoding. The two end nodes run in a multi-threaded QuNetSim application, which can run any communication protocol.

To integrate ComNetsEmu and the QuNetSim simulation, a bridge interface is used, responsible for moving traffic generated in ComNetsEmu to the QuNetSim application and back again. With ComNetsEmu, the quantum simulation part of the program is "containerized" in a Docker environment. Docker uses "containers" to isolate code and all its dependencies in virtual computing environments [53]. ComNetsEmu simplifies the deployment of containers specifically, to simulate communication networks. One can configure the IP addresses of these containers and monitor the incoming and outgoing traffic flow of the container. Using ComNetsEmu, the bridge interface is deployed in a container, which runs QuNetSim as an application and monitors the network traffic like a "man-in-the-middle".

The bridge is configured such that it routes all the network-layer traffic to a particular queue and it is configured to monitor the incoming traffic arriving in this queue. When packets arrive, they are disassembled into binary strings and processed in the QuNetSim application. The quantum channel running in the QuNetSim application converts the binary strings into data frames and transmits them over the channel to the QuNetSim Host, representing the receiver's link layer. Within the QuNetSim application, the destination receives the qubits and decodes the frames according to the particular algorithm, and reconstructs the binary string once the end of the data frame is detected. Once completed, the network-layer packet is reassembled and sent to the

bridge to proceed with routing.

4.4.4. Simulations of Bursty Network Traffic with Entanglement Assistance

As a proof of concept using the simulation framework for quantum-enhanced classical networks, we analyze the capacity of a single quantum link between two devices where one device communicates periodically with data bursts, separated by idle periods of fixed duration. Here we require no network routing and focus just on the single link. This type of data traffic profile is commonly found in Internet of Things (IoT) networks, specifically sensor networks[186]. To simulate this, we program the sender to perform periodic bursts of data packet transmissions to the receiver using superdense coding and then transmit EPR frames between data bursts, as described in the protocols in the previous section. The output statistics of the simulation provide a count of the occurrences of the various simulation events, such as the number of transmissions made over the fiber and the number of EPR pairs generated. We accumulate these statistics for various simulation parameters and show the results in Fig. 4.4.

First, we simulated the effects of varying the number of qubits an EPR frame is composed of, with a fixed number of 10 EPR frames sent between bursts. The number of data packets in a single burst is also a simulation variable, where each data packet is composed of 168 classical bits. The upper plot in Fig. 4.4 shows the trends of the average amount of data sent per transmission for varying both the length of the EPR frame as well as the number of packets in a burst. The trends show that—as one could expect—when fewer data packets are sent in a single burst, and an EPR frame is composed of more EPR pairs, the number of bits per transmission will more rapidly approach two. In the next simulation scenario, we simulated the transmission of data packets where a varying number of EPR frames were generated in the idle times also considering EPR frames of fixed length. The results are depicted in the lower plot of Fig. 4.4. As expected, when there is more time to generate EPR frames between data transmissions, the average capacity of the channel increases.

To reconfirm these results, by exploiting the periodicity in the setup, we can find analytic expressions matching our simulation as follows: Let B be the number of packets in a burst, E the EPR frames per burst, D the bits in the packet, and L the EPR pairs in an EPR frame. Then, the transmission time of one data burst is calculated to equal $EL + (DB - 2EL)$ whenever $EL < BD/2$ and the number of bits per transmission is $C(B, E, D, L) = BD/(DB - EL)$ when $EL < BD/2$, and $C(B, E, D, L) = 2$ otherwise. We add the analytical result to the plots with a dashed line and we see all the data points very closely matching the trends.

4.4.5. Conclusions and Outlook

In summary, we have demonstrated an integration of the classical network emulator ComNetsEmu with a quantum link layer, simulated using QuNetSim. As a proof-of-concept, we presented a classical-quantum link-layer protocol for transmitting classical information over a quantum network at an accelerated data rate, using pre-established quantum entanglement, where entanglement is generated during idle times in the network under a bursty network traffic model.

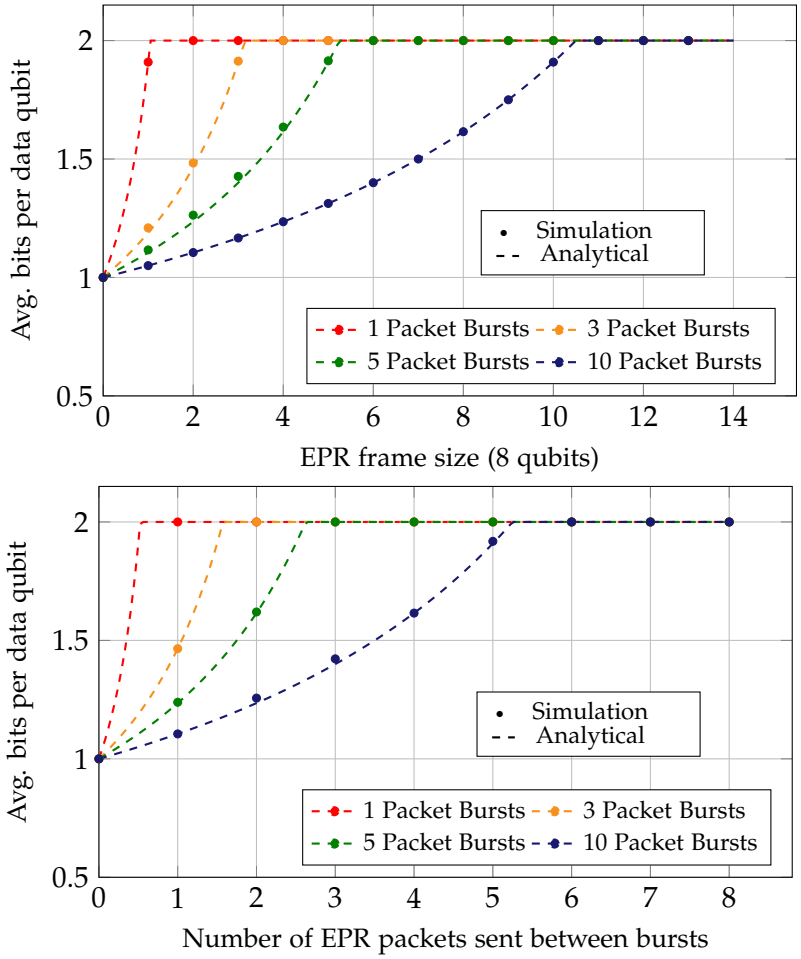


Figure 4.4.: Upper: Bursty traffic with varying EPR frame length with 10 EPR frames between bursts using superdense coding. Lower: Bursty traffic with varying numbers of EPR frames between 168-bit packet bursts with 160-qubit EPR frames using superdense coding.

Future work will consider other scenarios where data is transmitted in a more continuous stream. Using this network emulator, we plan to take the proposed encoding schemes of [187], which uses a "trade-off coding" approach, combining classical data with entanglement-assisted communication in a single transmission, achieving higher communication rates in some cases. In [188], there is another approach where only the receivers of the classical transmission share entanglement, which is used for message decoding. Using our simulation tool, we plan to test explicit protocols for each setting.

Because of the software architecture of QuNetSim, future work will also allow us to increase the level of realism in the quantum simulation, and to run such protocols on physical hardware without much change to the simulation code, as these synchronization tasks are abstracted away in QuNetSim. Overall, integrating quantum features into classical network emulators paves the way for classical network engineers to investigate their use cases in a familiar setting but with the additional features offered by quantum

networks, creating a tool for testing novel applications on quantum-enhanced networks.

5. Conclusion and Outlook

Quantum computing and communication are rapidly developing fields and, in this thesis, we have explored a relatively small portion of the potential applications. Research and development for classical computing and communication solutions are decades ahead of quantum networks, and so the fate of the future of quantum networks ultimately boils down to how big of an advantage the applications bring. The applications we have covered in this work have been focused on entanglement-assisted classical networks and their surrounding protocols and in networked quantum computers. Indeed we must approach this coming quantum revolution in computing and communication with both optimism and skepticism.

Using entanglement-assisted communication for general-purpose communication networks would lead to a large drop in communication rates for decades to come, as current fiber networks can transmit gigabytes of data per second. Indeed, the best rates achieved in quantum communication scenarios for quantum key distribution have so far been in the kilobytes per second domain, and distributing keys is far easier than arbitrary quantum message transmission. Moreover, moving from point-to-point to end-to-end communication would require a strong technological advancement in quantum repeater technology, assumed to be as difficult to build as a quantum computer itself. On the other side, for quantum computing, it is still not clear when the technology will be good enough or have enough qubits to solve problems at a large enough scale to be practical. Networking quantum computers for distributed quantum computing will bring yet further challenges in distributed control systems and algorithms for resource allocation, also requiring the ability to transmit high-quality entangled pairs.

The development timeline of revolutionary technologies tends to begin with a high level of skepticism and backtracking, but as history has shown, sometimes by pushing on with development despite the doubt, the uncertainty in the practicality of the technology begins to clear. Quantum networks have the potential to enable higher transmission rates for some known communication scenarios and quantum computers have the potential to perform certain computational tasks faster than any classical system. Building these technologies despite current skepticism will either prove the skeptics correct or lead to breakthrough technologies. In either case, a clear answer will form, where currently one is not known, a worthwhile pursuit in the scientific approach.

In this thesis, we investigated novel quantum communication and computing applications in depth to paint a clearer picture of how these technologies can be used in the future. We investigated various network layers for entanglement-enhanced quantum networks, thinking about entanglement distribution and generation strategies that can be applied to other scenarios requiring entanglement. For distributed quantum computing, we worked in two unique settings: cloud quantum computing, and distributed quantum computing. We explored the use of quantum clustering algorithms for an unsupervised learning problem related to energy grid clustering, using only cloud-based quantum computing. For distributed quantum computing, we designed generalized algorithms for distributing and scheduling quantum algorithms over a

network of quantum computers, with proposed architectures of the network and the control.

These applications have been shown to work in principle, but moving to an engineering setting, it is important to verify and test their correctness in many settings, especially as many protocols will need to be invented to put the applications to practice. We, therefore, developed various simulation environments that can be used to further explore these applications. QuNetSim is a software framework that allows for high-level quantum communication applications to be developed in a programmatic setting. Using QuNetSim, we have enabled further simulations, developing two more libraries built on top of QuNetSim, Interlin-q, and QontainerNet. These simulation platforms are an important initial step in the direction of building industry-level simulation frameworks and showing that such software is possible drives the field further.

Quantum technologies continue to develop and evolve. The level of interest in quantum science is growing ever faster, creating a new market in computing and communication technologies. The field is young and uncertain and so leveraging the interest level now to continue to push research and invention is an important step to ensuring a long lifetime for quantum science to come. Approaching the field of quantum technology with both skepticism and optimism is important, but few revolutionary technologies have justified their existence without struggle. With this work, we aim to shine an optimistic light on the field, showing the power of quantum technology, and planting a seed for future quantum scientists.

A. QuNetSim Example Simulations

In this section, we cover three examples using QuNetSim. The first example is an explanatory example that demonstrates establishing a network and sending qubits between Hosts in the network. The next example is of establishing quantum entanglement between two parties where the knowledge of which Hosts share the entangled pair is anonymous to all but the two Hosts. This is based off of the protocol in reference [189]. In the final example, we establish a network and define a custom routing mechanism. In this example, Hosts are constantly generating entanglement when possible, and super-dense encoded messages are then transmitted between parties using the route with the most established entanglement. This routing concept is further established in reference [190].

A.1. Sending Data Qubits

We demonstrate here the simple task of sending qubits that have been encoded with information, or as they are called in QuNetSim, “data" qubits. We send the qubits from Host **A** to Host **D** over the network in Fig. A.1.

```
# Network is a singleton
network = Network.get_instance()
# Start the network with the nodes defined above
network.start()

# Define the Hosts
host_A = Host('A')
# Define the Host's connections
host_A.add_connection('B')
# Start the Host
host_A.start()

host_B = Host('B')
host_B.add_connections(['A', 'E'])
host_B.start()

host_E = Host('E')
host_E.add_connections(['B', 'D'])
host_E.start()

host_D = Host('D')
host_D.add_connection('E')
host_D.start()

# Add the Hosts to the network to build the network
```

```
# graph
network.add_hosts([host_A, host_B, host_E, host_D])
```

Next, we want to generate the protocols for **A** and **D** to run. Protocols are the functionality of a Host. Protocols are very flexible with the only exception being that the protocol function must take the Host as the first parameter. Below is a sample protocol and code to launch the protocol for a Host. In this example, Host **A** is sending five data qubits to **D**.

```
def sender(host, receiver):
 """
 Sends 5 qubits to Host *receiver*.
 Args:
   host (Host): The Host object running the protocol
   receiver (str): The name of the receiver
 """
 for i in range(5):
   # The Host creates a qubit
   qubit = Qubit(host)
   # Perform a Hadamard operation on the qubit
   qubit.H()
   # The Host sends the qubit to the receiver
   # and awaits an ACK from the receiver that
   # the qubit arrived for some fixed amount of time.
   ack_arrived = host.send_qubit(receiver, qubit,
                                 await_ack=True)
   if ack_arrived:
       print('Qubit sent successfully.')
   else:
       print('Qubit did not transmit.')
```

A protocol for receiving qubits must also be written.

```
def receiver(host, sender):
 """
 Sends 5 qubits to Host *receiver*.
 Args:
   host (Host): The Host object running the protocol
   receiver (str): The name of the sender
 """
 for i in range(5):
   # The Host awaits a data qubit for 10 seconds maximum
   qubit = host.get_data_qubit(sender, wait=10)
   # If the qubit arrived, measure it
   if qubit is not None:
       m = qubit.measure()
       print("%s received qubit in state %d"
                       % (host.host_id, m))
```

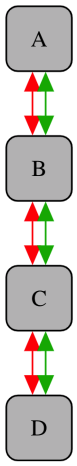


Figure A.1.: The network depiction for example A.

```
    else:
        print("Qubit did not arrive.")
```

To run the protocols, we have the following lines of code:

```
# A runs the sender protocol with D
host_A.run_protocol(sender, ('D',))
# D runs the receiver protocol A
host_D.run_protocol(receiver, ('A',))
```

In summary, with these code snippets, we can simulate the transmission of five qubits from **A** to **D** over the network in Fig. A.1. Of course, this simple example is intended to give a gentle introduction to how QuNetSim works. In the next examples, we develop more complex protocol simulations to see further the simplicity of using QuNetSim to write quantum network simulations.

A.2. GHZ-based Quantum Anonymous Distribution

In this example, we will demonstrate how to simulate an instance of GHZ-based quantum anonymous transmission [189]. The goal of the protocol is to hide the creation of an entangled pair between two parties. This protocol implementation demonstrates the simplicity of translating a protocol from a high-level mathematical syntax into a simulation using QuNetSim. The protocol involves establishing GHZ states amongst n parties as well as broadcasting measurement outcomes. These types of tasks would involve a relatively high level of software synchronization to program a simulation from scratch. Here we demonstrate that in QuNetSim, synchronization logic is programmatically kept at a high level.

As a first step, as always, we generate a network. Below is the code to generate such a network which is depicted in Fig. A.2.

```
network = Network.get_instance()
```

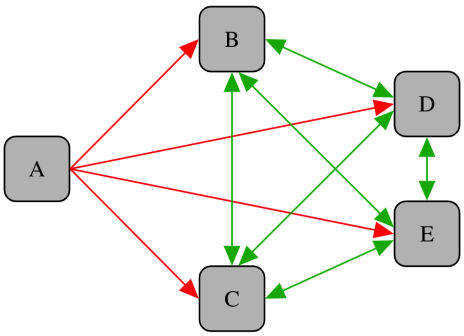


Figure A.2.: The network depiction for example B.

```
network.start()

host_A = Host('A')
host_A.add_connections(['B', 'C', 'D', 'E'])
host_A.start()

host_B = Host('B')
host_B.add_c_connections(['C', 'D', 'E'])
host_B.start()

host_C = Host('C')
host_C.add_c_connections(['B', 'D', 'E'])
host_C.start()

host_D = Host('D')
host_D.add_c_connections(['B', 'C', 'E'])
host_D.start()

host_E = Host('E')
host_E.add_c_connections(['B', 'C', 'D'])
host_E.start()

network.add_hosts([host_A, host_B, host_C,
                   host_D, host_E])
```

The next step is to write the behavior of the GHZ distributor, which in this example is Host **A**. QuNetSim provides a function for distributing GHZ states so the function `distribute` simply takes the distributing Host as the first parameter and the list of receiving nodes as the second. One notices that the flag `distribute` has been set to true in the `send_ghz` method call. This tells the sending Host that it should not keep part of the GHZ state, rather, it should generate a GHZ state amongst the list given and send it to the parties in the node list, keeping no part of the state for itself.

```
def distribute(host, nodes):
    """
    Args:
        host (Host): The Host running the protocol
        nodes (list): The list of nodes to distribute the
                      GHZ to
    """
    # distribute=True => don't keep part of the GHZ
    host.send_ghz(nodes, distribute=True)
```

The next type of behavior we would like to simulate is that of a node in the group that is not attempting to establish an EPR pair. In the anonymous entanglement protocol, the behavior of such a node is to simply receive a piece of a GHZ state, perform a Hadamard operation on the received qubit, measure it, and broadcast to the remaining participating parties the outcome of a measurement in the computational basis. Below we see how to accomplish this. In the `node` function, the first parameter is, as always, the Host that is performing the protocol. The second is the ID of the node distribution of the GHZ state, which in this example, is Host **A**. The Host fetches its GHZ state where, if it is not available at the time, they will wait ten seconds for it, accomplished by setting the `wait` parameter. If they have not received part of the GHZ state, then the protocol has failed, otherwise, they simply perform the Hadamard operation on the received qubit, measures it, and broadcast the message to the network. QuNetSim includes the task of broadcasting as a built-in function and therefore the task of sending classical messages to the whole network is simplified to one line of code.

```
def node(host, distributor):
    """
    Args:
        host (Host): The Host running the protocol
        distributor (str): The ID of the distributor
                           for GHZ states
    """
    q = host.get_ghz(distributor, wait=10)
    if q is None:
        print('failed')
        return
    q.H()
    m = q.measure()
    host.send_broadcast(str(m))
```

We implement next the behavior of the party in the protocol acting as one end of the EPR link that we label the *sender*. Here we take three parameters (other than the Host running the protocol), the ID of the distributor, the receiver that is the holder of the other half of the EPR pair, and an agreed-upon ID for the EPR pair that will be generated. In QuNetSim, qubits have IDs for easier synchronization between parties. For EPR pairs and GHZ states, qubits share an ID, that is, the collection of qubits would all have the same ID. This is done so that when parties share many EPR pairs, they can

easily synchronize their joint operations. The `sender` protocol is the following: first, they receive part of a GHZ state, select a random bit and then broadcast the message so that they appear as just any other node. They then manipulate their part of the GHZ state according to what the random bit was. If the bit was 1, then a Z gate is applied. The sending party can then add the qubit as an EPR pair shared with the receiver. This EPR pair can then be used as if the sender and receiver established an EPR directly.

```
def sender(host, distributor, receiver, epr_id):
    """
    Args:
        host (Host): The Host running the protocol
        distributor (str): The ID of the distributor
                           for GHZ states
        receiver (str): Who to teleport the qubit to after
                        EPR is established
        epr_id (str): The ID for the EPR pair established
                      ahead of time
    """
    q = host.get_ghz(distributor, wait=10)
    b = random.choice(['0', '1'])
    host.send_broadcast(b)
    if b == '1':
        q.Z()

    host.add_epr(receiver, q, q_id=epr_id)
    qubit_to_send = Qubit(host)
    host.send_teleport(r, qubit_to_send)
    host.empty_classical()
```

Finally, we establish the behavior of the *receiver*. The receiver here behaves as follows: First, to mask their behavior they randomly choose a bit and broadcast it to the network. Once complete, they await the remainder of the broadcast messages. In QuNetSim, classical messages are stored as a list in the `classical` field. Since there are three other parties, other than the receiver themself, they await the other three messages. Once they arrive, the receiver computes a global parity operation by taking the *XOR* of all received bits along with their own random choice. With this, the receiver can apply a controlled Z gate which establishes the EPR pair with the correct sender. They simply add the EPR pair and complete the protocol.

```
def receiver(host, distributor, sender, epr_id):
    q = host.get_ghz(distributor, wait=10)
    b = random.choice(['0', '1'])
    host.send_broadcast(b)

    messages = []
    # Await broadcast messages from all parties
    while len(messages) < 3:
```

```
        messages = host.classical

    parity = int(b)
    for m in messages:
        if m.sender != s:
            parity = parity ^ int(m.content)
    if parity == 1:
        q.Z()

    # Established secret EPR, add it
    host.add_epr(sender, q, q_id=epr_id)

    # Await a teleportation from the anonymous sender
    q = host.get_data_qubit(s, wait=10)
```

The last step of writing a QuNetSim simulation is to run the protocols for each desired Host. Below, we let Host **A** act as the GHZ state distributor, **B** and **C** are neutral parties running the `node` behavior, **D** acts as the sender and **E** acts as the receiver. The following code initiates the simulation.

```
epr_id = '12345'
host_A.run_protocol(distribute, (['B', 'C', 'D', 'E'],))
host_B.run_protocol(node, ('A',))
host_C.run_protocol(node, ('A',))
host_D.run_protocol(sender, ('A', 'E', epr_id))
host_E.run_protocol(receiver, ('A', 'D', epr_id))
```

A.3. Routing with Entanglement

In this example, we demonstrate how one can use QuNetSim to test a custom routing algorithm. We consider the network shown in Fig. A.3. The example here uses the approach of [190] for choosing the route and using dense coding for classical message transmission. For this example, we assume the network is using entanglement resources to transfer classical information via superdense coding from Host **A** to Host **B**. The sending and receiving parties must first establish an EPR pair to send messages via superdense coding. The sender performs a specific set of operations on its half of the EPR pair and then transmits it through the network. When the receiver received the qubit, it performs a specific set of operations such that it recovers two bits of classical information. What is important here is that **A** and **B** are separated by one hop. To share an EPR pair, an entanglement swap routing has to be made.

The strategy for this routing algorithm is to first build a graph of the entanglement shared amongst the Hosts in the network. The strategy, since superdense coding consumes entanglement pairs, will then be to route information through the path that contains the most entanglement. In this example, we show how this can be accomplished. As always, we first generate the network topology.

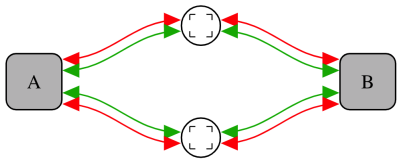


Figure A.3.: The network depiction for example C.

```
nodes = ['A', 'node_1', 'node_2', 'B']
network.use_hop_by_hop = False
network.use_ent_swap = True
network.set_delay = 0.1
network.start(nodes)

A = Host('A')
A.add_connections(['node_1', 'node_2'])
A.start()

node_1 = Host('node_1')
node_1.add_connections(['A', 'B'])
node_1.start()

node_2 = Host('node_2')
node_2.add_connections(['A', 'B'])
node_2.start()

B = Host('B')
B.add_connections(['node_1', 'node_2'])
B.start()

network.add_hosts([A, node_1, node_2, B])
```

```
def generate_entanglement(host):
 """
 Generate entanglement if the Host is idle.
 """
 while True:
   # Check if the Host is not processing
   if host.is_idle():
     for connection in host.quantum_connections:
        host.send_epr(connection)
```

```
def routing_algorithm(network_graph, source, target):
 """
 Entanglement based routing function.

 Args:
```

```
    network_graph (networkx.DiGraph): The directed graph
                                      representation of
                                      the network.
    source (str): The sender ID
    target (str): The receiver ID
Returns:
    (list): The route ordered by the steps in the route.
"""

# Generate entanglement network
entanglement_network = nx.DiGraph()
nodes = network_graph.nodes()
# A relatively large number
inf = 1000000
for node in nodes:
  host = network.get_host(node)
  for connection in host.quantum_connections:
    num_epr_pairs = len(host.get_epr_pairs(connection))
    if num_epr_pairs == 0:
        entanglement_network
          .add_edge(host.host_id,
                    connection,
                    weight=inf)
    else:
        entanglement_network
            .add_edge(host.host_id,
                      connection,
                      weight=1. / num_epr_pairs)

try:
  return nx.shortest_path(entanglement_network,
                          source,
                          target,
                          weight='weight')
except Exception as e:
  print('Error getting route.')
```

We can now begin to simulate the network using this configuration. In this simulation, the nodes in the middle are sending entanglement to the other nodes as often as they can, establishing the most EPR pairs while they are free to do so. We start them on this process as so:

```
node_1.run_protocol(generate_entanglement)
node_2.run_protocol(generate_entanglement)
```

Now we tell the network to use a different routing algorithm for the quantum information in the network:

```
network.quantum_routing_algo = routing_algorithm
```

Finally we trigger Host **A** to begin transmitting 100 messages via superdense coding to Host **B**.

```
choices = ['00', '11', '10', '01']
for _ in range(100):
    m = random.choice(choices)
    A.send_superdense('B', m, await_ack=True)
```

B. Interlin-q Example Simulations

B.1. Distributed Quantum Phase Estimation

In this example, we demonstrate a version distributed QPE using the circuit configuration as depicted in Fig. B.1. In this simulated architecture, we place the measurement qubits on one quantum computer (the upper portion of the circuit), and the qubit whose phase to estimate on another (the lower portion). In this case, the control unitary gates, since they are non-local, will need additional instructions added to perform them correctly. In this example, we see we just need to build the circuit as it is depicted, and Interlin-q will then perform the circuit remapping and carry out the execution of the instructions.

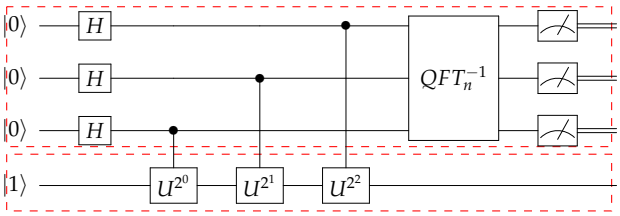


Figure B.1.: Circuit diagram for QPE with unitary operation U for this example.

To start the example, we import the necessary libraries and initialize the configuration:

```
import numpy as np
# QuNetSim Components
from qunetsim.components import Network
from qunetsim.backends import EQSNBackend
# Interlin-q Components
from interlinq import ControllerHost, Circuit, ComputingHost, Constants,
    Qubit

# Initializing network objects
network = Network.get_instance()
network.start()
controller_host = ControllerHost(
        host_id="controller",
        backend=EQSNBackend()
    )
# Create a network of distributed QPUs
computing_hosts, q_map = controller_host \
  .create_distributed_network(
     num_computing_hosts=2,
     num_qubits_per_host=3
  )
# Start the controller and create the network
```

```
controller_host.start()
network.add_host(controller_host)
network.add_hosts(computing_hosts)
```

We make use of QuNetSim's network object and add the controller and computing nodes to the network. The `create_distributed_network` ControllerHost method will generate a completely connected network topology and in this case specifically, with two ComputingHosts each with three qubits. The next step is to define the protocol logic for each of the network nodes:

```
def computing_host_protocol(host):
    """
    Protocol for the computing host
    """
    host.receive_schedule()
    host.send_results()

def controller_host_protocol(host, q_map, input_gate):
    """
    Protocol for the controller host
    """
    # Generate the circuit for QPE
    circuit = qpe_circuit(q_map, input_gate)
    host.generate_and_send_schedules(circuit)
    # Block until measurement results arrive
    host.receive_results()
    meas_results = host.results["QPU_1"]["val"]
    output = [0] * 3
    print(results)
    for qubit in meas_results.keys():
        output[int(qubit[-1])] = meas_results[qubit]
    decimal_value = 0
    output.reverse()
    for i, bit in enumerate(output):
        decimal_value += ((2 ** i) * bit)
    phase = decimal_value / 8
    print("The estimated phase is {0}".format(phase))
```

The actions of the computing host generally have the same structure which is to await instructions from the controller and then send the measurement results back to the controller. The controller on the other hand takes as input the network topology (assumed to be completely connected) and a unitary which to use for the phase estimation step. The controller uses the information to then generate the circuit, generate control instructions, and then send it to the computing nodes, awaiting the measurement results to perform the post-processing step. To generate the circuit:

```
def phase_gate(theta):
    return np.array([[1, 0], [0, np.exp(1j * theta)]])
```

```

def quantum_phase_estimation_circuit(q_map, client_input_gate):
    """
    Returns the monolithic circuit for the quantum phase estimation
    algorithm
    """
    phase_qubit = Qubit(computing_host_id='QPU_0', q_id=q_map['QPU_0'][0])

    phase_qubit.single(Operation.X)
    meas_qubits = []

    for q_id in q_map['QPU_1']:
        q = Qubit(computing_host_id='QPU_1', q_id=q_id)
        q.single(Operation.H)
        meas_qubits.append(q)
    for i, q in enumerate(meas_qubits):
        for _ in range(2 ** i):
            q.two_qubit(Operation.CUSTOM_CONTROLLED, phase_qubit,
    client_input_gate)
    # Inverse Fourier Transform
    meas_qubits.reverse()
    for i, q in enumerate(meas_qubits):
        for j, q2 in enumerate(meas_qubits[:i]):
            q2.two_qubit(Operation.CUSTOM_CONTROLLED, q, phase_gate(-np.
    pi * (2 ** j) / (2 ** i)))
        q.single(gate=Operation.H)
    # Measure the qubits
    for q in meas_qubits:
        q.measure()
    return Circuit(q_map, qubits=meas_qubits +
                                 [phase_qubit])
```

Finally, to begin execution and wait for results:

```
# For phase = 1/3
input_gate = np.array([
                        [1, 0],
                        [0, np.exp(1j * 2 * np.pi / 3)]
                    ])
t1 = controller_host.run_protocol(
    controller_host_protocol,
    (q_map, input_gate))
computing_hosts[0].run_protocol(computing_host_protocol)
computing_hosts[1].run_protocol(computing_host_protocol)
t1.join()
network.stop(True)
```

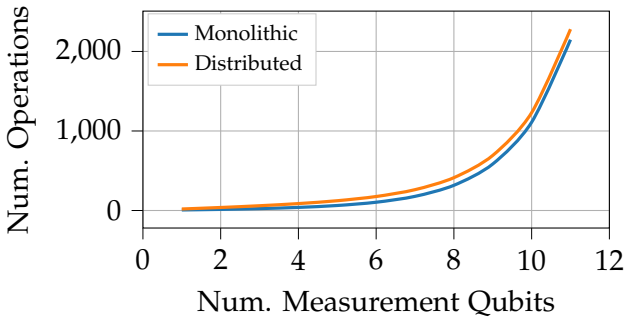


Figure B.2.: The number of operations required to perform quantum phase estimation on one qubit with varying levels of precision when using a distributed system of two QPUs.

Since the backend simulator we selected, EQSN, is a noiseless simulator, we need just use one shot to get an estimate of the phase. The output, in this case, is: "The estimated phase is 0.375" as expected for a 3-bit estimation. In this simulation, we can gather statistics about how much communication is involved in the network to execute the algorithm. For the example, we have used 3 measurement qubits, but by changing the parameter `num_qubits_per_host`, we can adjust the number of measurement qubits. In Fig. B.2, we plot the number of total operations needed, including the number of gates, entanglement generation, and classical communication between the nodes to execute the algorithm, comparing a monolithic version to a distributed version when the measurement qubits are separated as in Fig. B.1. In this case, many control operations can be performed via small amounts of classical communication and entanglement because the control information for each measurement qubit for non-local gates can be transferred using one EPR pair and two classical messages.

C. Additional Material

Algorithm 15 Does Not Fit

Input:

- $Q = [q_1, ..., q_n]$ the collection of QPUs in the distributed system, non-increasingly sorted by number of available qubits
- A the size of the Ansatz

Output: If the Ansatz can fit in the distributed QPU Q

DoesNotFit(Q, A)

```
if Q is empty then
  return true
for q_i ∈ Q do
  curAllocation ← [0, ..., 0]                    ▷ Allocate n element array of 0s
  possibleQPUs ← {q_1, ..., q_i} ⊆ Q
  if i == 1 then
    k ← QPUNumber(possibleQPUs[1])                ▷ The QPU index
    curAllocation[k] ← q_1 − 1
  else
    State k ← QPUNumber(possibleQPUs[1])          ▷ The QPU index
    curAllocation[k] ← q_1 − 3
    for q_j ∈ {q_2, ..., q_i} do
      k ← QPUNumber(possibleQPUs[j])              ▷ The QPU index
      curAllocation[k] ← q_j − 2
  if sum(curAllocation) ≥ A then
    return false                                  ▷ The Ansatz does fit
return true
```

Bibliography

[1] E. Gibney. "The quantum gold rush". In: *Nature* 574.7776 (2019), pp. 22–24.

[2] P. W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134.

[3] R. Renner. "Security of quantum key distribution". In: *International Journal of Quantum Information* 6.01 (2008), pp. 1–127.

[4] L. Salvail, M. Peev, E. Diamanti, R. Alléaume, N. Lütkenhaus, and T. Länger. "Security of trusted repeater quantum key distribution networks". In: *Journal of Computer Security* 18.1 (2010), pp. 61–87.

[5] S. Guha, Q. Zhuang, and B. A. Bash. "Infinite-fold enhancement in communications capacity using pre-shared entanglement". In: *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE. 2020, pp. 1835–1839.

[6] D. A. Lidar and T. A. Brun. *Quantum error correction*. Cambridge university press, 2013.

[7] E. Campbell, A. Khurana, and A. Montanaro. "Applying quantum algorithms to constraint satisfaction problems". In: *Quantum* 3 (2019), p. 167.

[8] E. Diamanti, H.-K. Lo, B. Qi, and Z. Yuan. "Practical challenges in quantum key distribution". In: *npj Quantum Information* 2.1 (2016), pp. 1–12.

[9] H. Shi, Z. Zhang, and Q. Zhuang. "Practical route to entanglement-enhanced communication over noisy bosonic channels". In: *2020 Conference on Lasers and Electro-Optics (CLEO)*. IEEE. 2020, pp. 1–2.

[10] J. S. Bell. "On the einstein podolsky rosen paradox". In: *Physics Physique Fizika* 1.3 (1964), p. 195.

[11] A. S. Holevo. "Bounds for the quantity of information transmitted by a quantum communication channel". In: *Problemy Peredachi Informatsii* 9.3 (1973), pp. 3–11.

[12] R. Van Meter, R. Satoh, N. Benchasattabuse, T. Matsuo, M. Hajdušek, T. Satoh, S. Nagayama, and S. Suzuki. "A Quantum Internet Architecture". In: *arXiv preprint arXiv:2112.07092* (2021).

[13] P. K. Tysowski, X. Ling, N. Lütkenhaus, and M. Mosca. "The engineering of a scalable multi-site communications system utilizing quantum key distribution (QKD)". In: *Quantum Science and Technology* 3.2 (2018), p. 024001.

[14] M. Mehic, M. Niemiec, S. Rass, J. Ma, M. Peev, A. Aguado, V. Martin, S. Schauer, A. Poppe, C. Pacher, et al. "Quantum key distribution: a networking perspective". In: *ACM Computing Surveys (CSUR)* 53.5 (2020), pp. 1–41.

[15] H. Zimmermann. "OSI reference model-the ISO model of architecture for open systems interconnection". In: *IEEE Transactions on communications* 28.4 (1980). doi: 10.1109/TCOM.1980.1094702, pp. 425–432. DOI: 10.1109/TCOM.1980.1094702.

[16] A. Pirker and W. Dür. "A quantum network stack and protocols for reliable entanglement-based networks". In: *New Journal of Physics* 21.3 (2019). doi: 10.1088/1367-2630/ab05f7, p. 033003. DOI: 10.1088/1367-2630/ab05f7.

[17] C. Jones, D. Kim, M. T. Rakher, P. G. Kwiat, and T. D. Ladd. "Design and analysis of communication protocols for quantum repeater networks". In: *New Journal of Physics* 18.8 (2016), p. 083015.

[18] S. Muralidharan, L. Li, J. Kim, N. Lütkenhaus, M. D. Lukin, and L. Jiang. "Optimal architectures for long distance quantum communication". In: *Scientific reports* 6 (2016), p. 20463.

[19] A. Dahlberg, M. Skrzypczyk, T. Coopmans, L. Wubben, F. Rozpędek, M. Pompili, A. Stolk, P. Pawełczak, R. Knegjens, J. de Oliveira Filho, et al. "A link layer protocol for quantum networks". In: *Proceedings of the ACM Special Interest Group on Data Communication*. SIGCOMM '19. doi: 10.1145/3341302.3342070. Beijing, China: Association for Computing Machinery, 2019, pp. 159–173. ISBN: 9781450359566. DOI: 10.1145/3341302.3342070.

[20] S. Bose, V. Vedral, and P. L. Knight. "Multiparticle generalization of entanglement swapping". In: *Physical Review A* 57.2 (1998), p. 822.

[21] C. H. Bennett, H. J. Bernstein, S. Popescu, and B. Schumacher. "Concentrating partial entanglement by local operations". In: *Physical Review A* 53.4 (1996), p. 2046.

[22] C. H. Bennett, G. Brassard, S. Popescu, B. Schumacher, J. A. Smolin, and W. K. Wootters. "Purification of noisy entanglement and faithful teleportation via noisy channels". In: *Physical review letters* 76.5 (1996), p. 722.

[23] W. Kozlowski, A. Dahlberg, and S. Wehner. "Designing a quantum network protocol". In: *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*. 2020, pp. 1–16.

[24] R. Van Meter and S. J. Devitt. "The Path to Scalable Distributed Quantum Computing". In: *Computer* 49.9 (2016), pp. 31–42. DOI: 10.1109/MC.2016.291.

[25] S. DiAdamo, M. Ghibaudi, and J. Cruise. "Distributed Quantum Computing and Network Control for Accelerated VQE". In: *IEEE Transactions on Quantum Engineering* 2 (2021), pp. 1–21. DOI: 10.1109/TQE.2021.3057908.

[26] Z. Zhang and Q. Zhuang. "Distributed quantum sensing". In: *Quantum Science and Technology* (2020).

[27] R. Jozsa, D. S. Abrams, J. P. Dowling, and C. P. Williams. "Quantum clock synchronization based on shared prior entanglement". In: *Physical Review Letters* 85.9 (2000), p. 2010.

[28] S. Pirandola, U. L. Andersen, L. Banchi, M. Berta, D. Bunandar, R. Colbeck, D. Englund, T. Gehring, C. Lupo, C. Ottaviani, et al. "Advances in quantum cryptography". In: *Advances in Optics and Photonics* 12.4 (2020), pp. 1012–1236.

[29] R. Bassoli, H. Boche, C. Deppe, R. Ferrara, F. H. Fitzek, G. Janssen, and S. Saeedinaeeni. *Quantum Communication Networks*. Vol. 23. Springer, 2021.

[30] J. Nötzel and S. DiAdamo. "Entanglement-enhanced communication networks". In: *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE. 2020, pp. 242–248.

[31] S. Daiss, S. Langenfeld, S. Welte, E. Distante, P. Thomas, L. Hartung, O. Morin, and G. Rempe. "A quantum-logic gate between distant quantum-network modules". In: *Science* 371.6529 (2021), pp. 614–617.

[32] R. Quan, Y. Zhai, M. Wang, F. Hou, S. Wang, X. Xiang, T. Liu, S. Zhang, and R. Dong. "Demonstration of quantum synchronization based on second-order quantum coherence of entangled photons". In: *Scientific reports* 6.1 (2016), pp. 1–8.

[33] Y.-A. Chen, Q. Zhang, T.-Y. Chen, W.-Q. Cai, S.-K. Liao, J. Zhang, K. Chen, J. Yin, J.-G. Ren, Z. Chen, et al. "An integrated space-to-ground quantum communication network over 4,600 kilometres". In: *Nature* 589.7841 (2021), pp. 214–219.

[34] S. Hao, H. Shi, W. Li, J. H. Shapiro, Q. Zhuang, and Z. Zhang. "Entanglement-Assisted Communication Surpassing the Ultimate Classical Capacity". In: *Phys. Rev. Lett.* 126 (25 2021), p. 250501. DOI: 10.1103/PhysRevLett.126.250501.

[35] M. Pompili, S. L. Hermans, S. Baier, H. K. Beukers, P. C. Humphreys, R. N. Schouten, R. F. Vermeulen, M. J. Tiggelman, L. dos Santos Martins, B. Dirkse, et al. "Realization of a multinode quantum network of remote solid-state qubits". In: *Science* 372.6539 (2021), pp. 259–264.

[36] C. H. Bennett, P. W. Shor, J. A. Smolin, and A. V. Thapliyal. "Entanglement-assisted capacity of a quantum channel and the reverse Shannon theorem". In: *IEEE Transactions on Information Theory* 48.10 (2002), pp. 2637–2655.

[37] P. Mandayam, K. Jagannathan, and A. Chatterjee. "The Classical Capacity of Additive Quantum Queue-Channels". In: *IEEE Journal on Selected Areas in Information Theory* 1.2 (2020), pp. 432–444. DOI: 10.1109/JSAIT.2020.3015055.

[38] J. Nötzel and S. DiAdamo. "Entanglement-Assisted Data Transmission as an Enabling Technology: A Link-Layer Perspective". In: *2020 IEEE International Symposium on Information Theory (ISIT)*. Vol. 1. 2. IEEE. IEEE, 2020, pp. 1955–1960. DOI: 10.1109/ISIT44484.2020.9174366.

[39] M. M. Wilde, P. Hayden, and S. Guha. "Quantum trade-off coding for bosonic communication". In: *Physical Review A* 86.6 (2012), p. 062306.

[40] I. B. Djordjevic. "On Entanglement Assisted Classical Optical Communications". In: *IEEE Access* 9 (2021), pp. 42604–42609.

[41] V. Semenenko, X. Hu, E. Figueroa, and V. Perebeinos. "Entanglement generation in a quantum network with finite quantum memory lifetime". In: *arXiv preprint arXiv:2110.01061* (2021).

[42] S. DiAdamo, J. Nötzel, S. Sekavčnik, R. Bassoli, R. Ferrara, C. Deppe, F. H. Fitzek, and H. Boche. "Integrating Quantum Simulation for Quantum-Enhanced Classical Network Emulation". In: *IEEE Communications Letters* (2021), pp. 1–5. DOI: 10.1109/LCOMM.2021.3115982.

[43] T. Coopmans, R. Knegjens, A. Dahlberg, D. Maier, L. Nijsten, J. de Oliveira Filho, M. Papendrecht, J. Rabbie, F. Rozpędek, M. Skrzypczyk, et al. "NetSquid, a NETwork Simulator for QUantum Information using Discrete events". In: *Communications Physics* 4.1 (2021), pp. 1–15. DOI: 10.1038/s42005-021-00647-8.

[44] J. R. Jackson. "Networks of waiting lines". In: *Operations research* 5.4 (1957), pp. 518–521.

[45] F. P. Kelly. "Networks of queues". In: *Advances in Applied Probability* (1976), pp. 416–432.

[46] H. Chen and D. D. Yao. *Fundamentals of queueing networks: Performance, asymptotics, and optimization*. Vol. 4. Springer, 2001.

[47] L. Clarke, I. Glendinning, and R. Hempel. "The MPI message passing interface standard". In: *Programming environments for massively parallel distributed systems*. Springer, 1994, pp. 213–218.

[48] S. Lloyd. "Least squares quantization in PCM". In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.

[49] S. Kar and B. Swenson. "Clustering with distributed data". In: *arXiv preprint arXiv:1901.00214* (2019).

[50] G. Oliva, R. Setola, and C. N. Hadjicostis. "Distributed k-means algorithm". In: *arXiv preprint arXiv:1312.4176* (2013).

[51] J. Qin, W. Fu, H. Gao, and W. X. Zheng. "Distributed *k*-means algorithm and fuzzy *c*-means algorithm for sensor networks based on multiagent consensus theory". In: *IEEE transactions on cybernetics* 47.3 (2016), pp. 772–783.

[52] D. M. Powers. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation". In: *arXiv preprint arXiv:2010.16061* (2020).

[53] F. H. P. Fitzek, F. Granelli, and P. Seeling, eds. *Computing in Communication Networks - From Theory to Practice*. 1st ed. Vol. 1. 1. Elsevier, Jan. 1, 2020. ISBN: 9780128209042. published.

[54] J. Nötzel and S. DiAdamo. "Entanglement-enhanced communication networks". In: *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE. 2020, pp. 242–248.

[55] J. Nötzel. "Entanglement-Enabled Communication". In: *IEEE Journal on Selected Areas in Information Theory* 1.2 (2020), pp. 401–415. DOI: 10.1109/JSAIT.2020.3017121.

[56] S. DiAdamo, J. Nötzel, B. Zanger, and M. M. Beşe. "QuNetSim: A Software Framework for Quantum Networks". In: *arXiv:2003.06397*. Vol. 2. 2020, pp. 1–12. DOI: 10.1109/TQE.2021.3092395. arXiv: 2003.06397 [quant-ph].

[57] R. Bassoli, H. Boche, C. Deppe, R. Ferrara, F. H. P. Fitzek, G. Janßen, and S. Saeedinaeen. *Quantum Communication Networks*. 1st ed. Springer, Jan. 2021. ISBN: 978-3-030-62938-0.

[58] C. H. Bennett and S. J. Wiesner. "Communication via one-and two-particle operators on Einstein-Podolsky-Rosen states". In: *Physical review letters* 69.20 (20 Oct. 1992). doi: 10.1103/PhysRevLett.69.2881, p. 2881. DOI: 10.1103/PhysRevLett.69.2881. URL: https://link.aps.org/doi/10.1103/PhysRevLett.69.2881.

[59] J. Nötzel and S. DiAdamo. "Entanglement-Assisted Data Transmission as an Enabling Technology: A Link-Layer Perspective". en. In: *2020 IEEE International Symposium on Information Theory*. Virtual: IEEE, June 2020.

[60] F. Leditzky, M. A. Alhejji, J. Levin, and G. Smith. "Playing games with multiple access channels". In: *Nature communications* 11.1 (2020), pp. 1–5.

[61] A. Holevo and A. Kuznetsova. "The information capacity of entanglement-assisted continuous variable measurement". In: *arXiv preprint arXiv:2004.05331* (2020).

[62] B. P. Williams, R. J. Sadlier, and T. S. Humble. "Superdense coding over optical fiber links with complete Bell-state measurements". In: *Physical review letters* 118.5 (2017), p. 050501.

[63] T. S. Humble, R. J. Sadlier, B. P. Williams, and R. C. Prout. "Software-defined quantum network switching". In: *Disruptive Technologies in Information Sciences*. Vol. 10652. International Society for Optics and Photonics. 2018, 106520B.

[64] S. Wehner, D. Elkouss, and R. Hanson. "Quantum internet: A vision for the road ahead". In: *Science* 362.6412 (2018). doi: 10.1126/science.aam9288. DOI: 10.1126/science.aam9288.

[65] J. H. Saltzer, D. P. Reed, and D. D. Clark. "End-to-end arguments in system design". In: *ACM Transactions on Computer Systems (TOCS)* 2.4 (1984), pp. 277–288.

[66] P. C. Humphreys, N. Kalb, J. P. Morits, R. N. Schouten, R. F. Vermeulen, D. J. Twitchen, M. Markham, and R. Hanson. "Deterministic delivery of remote entanglement on a quantum network". In: *Nature* 558.7709 (2018), pp. 268–273.

[67] A. Reiserer, S. Ritter, and G. Rempe. "Nondestructive detection of an optical photon". In: *Science* 342.6164 (2013), pp. 1349–1351.

[68] G. Nogues, A. Rauschenbeutel, S. Osnaghi, M. Brune, J. Raimond, and S. Haroche. "Seeing a single photon without destroying it". In: *Nature* 400.6741 (1999), pp. 239–242.

[69] S. R. Sathyamoorthy, T. M. Stace, and G. Johansson. "Detecting itinerant single microwave photons". In: *Comptes Rendus Physique* 17.7 (2016), pp. 756–765.

[70] C. H. Bennett, D. P. DiVincenzo, J. A. Smolin, and W. K. Wootters. "Mixed-state entanglement and quantum error correction". In: *Physical Review A* 54.5 (1996), p. 3824.

[71] D. Gavinsky. "On the role of shared entanglement". In: *arXiv preprint quant-ph/0604052* (2006).

[72] C. H. Bennett, P. W. Shor, J. A. Smolin, and A. V. Thapliyal. "Entanglement-assisted classical capacity of noisy quantum channels". In: *Physical Review Letters* 83.15 (1999), p. 3081.

[73] J. Nötzel and S. DiAdamo. "Entanglement-Enabled Communication for the Internet of Things". In: *2020 International Conference on Computer, Information and Telecommunication Systems (CITS)*. 2020, pp. 1–6. DOI: 10.1109/CITS49457.2020.9232550.

[74] C. Crépeau, D. Gottesman, and A. Smith. "Secure multi-party quantum computation". In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. 2002, pp. 643–652.

[75] A. K. Ekert. "Quantum cryptography based on Bell's theorem". In: *Physical review letters* 67.6 (1991), p. 661.

[76] I. Maric, R. D. Yates, and G. Kramer. "Capacity of interference channels with partial transmitter cooperation". In: *IEEE Transactions on Information Theory* 53.10 (2007), pp. 3536–3548.

[77] U. Pereg and Y. Steinberg. "The arbitrarily varying broadcast channel with causal side information at the encoder". In: *IEEE Transactions on Information Theory* 66.2 (2019), pp. 757–779.

[78] B. Chor and E. Kushilevitz. "A communication-privacy tradeoff for modular addition". In: *Information Processing Letters* 45.4 (1993), pp. 205–210.

[79] M. Hayashi and T. Koshiba. "Verifiable Quantum Secure Modulo Summation". In: *arXiv preprint arXiv:1910.05976* (2019).

[80] S. Popescu and D. Rohrlich. "Causality and nonlocality as axioms for quantum mechanics". In: *Causality and locality in modern physics*. Springer, 1998, pp. 383–389.

[81] C. G. Almudever, L. Lao, X. Fu, N. Khammassi, I. Ashraf, D. Iorga, S. Varsamopoulos, C. Eichler, A. Wallraff, L. Geck, et al. "The engineering challenges in quantum computing". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE. 2017, pp. 836–845.

[82] *Quantum Services*. URL: https://www.ibm.com/quantum-computing/services.

[83] M. Schuld and F. Petruccione. *Supervised learning with quantum computers*. Vol. 17. Springer, 2018.

[84] S. Lloyd, M. Mohseni, and P. Rebentrost. "Quantum algorithms for supervised and unsupervised machine learning". In: *arXiv preprint arXiv:1307.0411* (2013).

[85] H. Buhrman, R. Cleve, J. Watrous, and R. De Wolf. "Quantum fingerprinting". In: *Physical Review Letters* 87.16 (2001), p. 167902.

[86] L. K. Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 212–219.

[87] M. Plesch and Č. Brukner. "Quantum-state preparation with universal gate decompositions". In: *Physical Review A* 83.3 (2011), p. 032302.

[88] E. Tang. "Quantum Principal Component Analysis Only Achieves an Exponential Speedup Because of Its State Preparation Assumptions". In: *Physical Review Letters* 127.6 (2021), p. 060503.

[89] S. U. Khan, A. J. Awan, and G. Vall-Llosera. "K-means clustering on noisy intermediate scale quantum computers". In: *arXiv preprint arXiv:1909.12183* (2019).

[90] S. Johri, S. Debnath, A. Mocherla, A. Singh, A. Prakash, J. Kim, and I. Kerenidis. "Nearest centroid classification on a trapped ion quantum computer". In: *npj Quantum Information* 7.1 (Aug. 2021), p. 122. ISSN: 2056-6387. DOI: 10.1038/s41534-021-00456-5.

[91] V. Kumar, G. Bass, C. Tomlin, and J. Dulny. "Quantum annealing for combinatorial clustering". In: *Quantum Information Processing* 17.2 (2018), pp. 1–14.

[92] D. Arthur et al. "Balanced k-means clustering on an adiabatic quantum computer". In: *Quantum Information Processing* 20.9 (2021), pp. 1–30.

[93] K. Benlamine, Y. Bennani, A. Zaiou, M. Hibti, B. Matei, and N. Grozavu. "Distance estimation for quantum prototypes based clustering". In: *International Conference on Neural Information Processing*. Springer. 2019, pp. 561–572.

[94] K. Benlamine, Y. Bennani, N. Grozavu, and B. Matei. "Quantum Collaborative K-means". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–7.

[95] I. Kerenidis, J. Landman, A. Luongo, and A. Prakash. "q-means: A quantum algorithm for unsupervised machine learning". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/16026d60ff9b54410b3435b403afd226-Paper.pdf.

[96] C.-H. Nguyen, K.-W. Tseng, G. Maslennikov, H. Gan, and D. Matsukevich. "Experimental SWAP test of infinite dimensional quantum states". In: *arXiv preprint arXiv:2103.10219* (2021).

[97] *Initialize*. URL: https://qiskit.org/documentation/stubs/qiskit.extensions.Initialize.html.

[98] R. LaRose and B. Coyle. "Robust data encodings for quantum classifiers". In: *Physical Review A* 102.3 (2020), p. 032420.

[99] D. Kopczyk. "Quantum machine learning for data scientists". In: *arXiv preprint arXiv:1804.10068* (2018).

[100] A. Cross. "The IBM Q experience and QISKit open-source quantum computing software". In: *APS March Meeting Abstracts*. Vol. 2018. 2018, pp. L58–003.

[101] R. Parekh, A. Ricciardi, A. Darwish, and S. DiAdamo. "Quantum Algorithms and Simulation for Parallel and Distributed Quantum Computing". In: *arXiv preprint arXiv:2106.06841* (2021).

[102] R. L. Thorndike. "Who belongs in the family?" In: *Psychometrika* 18.4 (1953), pp. 267–276.

[103] L. Van der Maaten and G. Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[104] J. Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (2018), p. 79.

[105] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah. "On the qubit routing problem". In: *arXiv preprint arXiv:1902.08091* (2019).

[106] K. R. Brown, J. Kim, and C. Monroe. "Co-designing a scalable quantum computer with trapped atomic ions". In: *npj Quantum Information* 2.1 (2016), pp. 1–10.

[107] M. Zomorodi-Moghadam, M. Houshmand, and M. Houshmand. "Optimizing teleportation cost in distributed quantum circuits". In: *International Journal of Theoretical Physics* 57.3 (2018), pp. 848–861.

[108] R. Van Meter, K. Nemoto, and W. Munro. "Communication links for distributed quantum computation". In: *IEEE Transactions on Computers* 56.12 (2007), pp. 1643–1653. DOI: 10.1109/TC.2007.70775.

[109] S. Daiss, S. Langenfeld, S. Welte, E. Distante, P. Thomas, L. Hartung, O. Morin, and G. Rempe. "A quantum-logic gate between distant quantum-network modules". In: *Science* 371.6529 (2021), pp. 614–617. ISSN: 0036-8075. DOI: 10.1126/science.abe3150. eprint: https://science.sciencemag.org/content/371/6529/614.full.pdf. URL: https://science.sciencemag.org/content/371/6529/614.

[110] A. Serafini, S. Mancini, and S. Bose. "Distributed quantum computation via optical fibers". In: *Physical review letters* 96.1 (2006), p. 010503.

[111] A. Yimsiriwattana and S. J. Lomonaco Jr. "Distributed quantum computing: A distributed Shor algorithm". In: *Quantum Information and Computation II*. Vol. 5436. International Society for Optics and Photonics. 2004, pp. 360–372. DOI: 10.1117/12.546504.

[112] S. DiAdamo, M. Ghibaudi, and J. Cruise. "Distributed Quantum Computing and Network Control for Accelerated VQE". In: *IEEE Transactions on Quantum Engineering* 2 (2021), pp. 1–21. DOI: 10.1109/TQE.2021.3057908.

[113] D. Ferrari, A. S. Cacciapuoti, M. Amoretti, and M. Caleffi. "Compiler Design for Distributed Quantum Computing". In: *IEEE Transactions on Quantum Engineering* 2 (2021), pp. 1–20. DOI: 10.1109/TQE.2021.3053921.

[114] J. Eisert, K. Jacobs, P. Papadopoulos, and M. B. Plenio. "Optimal local implementation of nonlocal quantum gates". In: *Physical Review A* 62.5 (2000), p. 052317. DOI: 10.1103/PhysRevA.62.052317.

[115] O. Daei, K. Navi, and M. Zomorodi-Moghadam. "Optimized Quantum Circuit Partitioning". In: *International Journal of Theoretical Physics* 59.12 (2020), pp. 3804–3820.

[116] P. Andres-Martinez and C. Heunen. "Automated distribution of quantum circuits via hypergraph partitioning". In: *Physical Review A* 100.3 (2019), p. 032308.

[117] R. G Sundaram, H. Gupta, and C. R. Ramakrishnan. "Efficient Distribution of Quantum Circuits". In: *35th International Symposium on Distributed Computing (DISC 2021)*. Ed. by S. Gilbert. Vol. 209. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 41:1–41:20. ISBN: 978-3-95977-210-5. DOI: 10.4230/LIPIcs.DISC.2021.41. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14843.

[118] S. J. Devitt. "Performing quantum computing experiments in the cloud". In: *Physical Review A* 94.3 (2016), p. 032329.

[119] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli. "State-of-the-art in heterogeneous computing". In: *Scientific Programming* 18.1 (2010), pp. 1–33.

[120] S. Mittal and J. S. Vetter. "A survey of CPU-GPU heterogeneous computing techniques". In: *ACM Computing Surveys (CSUR)* 47.4 (2015), pp. 1–35.

[121] T. Häner, D. S. Steiger, T. Hoefler, and M. Troyer. "Distributed Quantum Computing with QMPI". In: *arXiv preprint arXiv:2105.01109* (2021).

[122] N. M. Neumann, R. van Houte, and T. Attema. "Imperfect distributed quantum phase estimation". In: *International Conference on Computational Science*. Springer. 2020, pp. 605–615.

[123] M. Ying and Y. Feng. "An algebraic language for distributed quantum computing". In: *IEEE Transactions on Computers* 58.6 (2009), pp. 728–743.

[124] W. Tang, T. Tomesh, M. Suchara, J. Larson, and M. Martonosi. "CutQC: using small quantum computers for large quantum circuit evaluations". In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 2021, pp. 473–486.

[125] M. J. Quinn. *Parallel computing theory and practice*. McGraw-Hill, Inc., 1994.

[126] G. M. Amdahl. "Validity of the single processor approach to achieving large scale computing capabilities". In: *Proceedings of the April 18-20, 1967, spring joint computer conference*. 1967, pp. 483–485.

[127] M. D. Hill and M. R. Marty. "Amdahl's law in the multicore era". In: *Computer* 41.7 (2008), pp. 33–38.

[128] R. Beals, S. Brierley, O. Gray, A. W. Harrow, S. Kutin, N. Linden, D. Shepherd, and M. Stather. "Efficient distributed quantum computing". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 469.2153 (2013), p. 20120686. DOI: 10.1098/rspa.2012.0686.

[129] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien. "A variational eigenvalue solver on a photonic quantum processor". In: *Nature communications* 5.1 (2014), pp. 1–7. DOI: 10.1038/ncomms5213.

[130] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles. *Variational Quantum Algorithms*. 2020. arXiv: 2012.09265 [quant-ph].

[131] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, S. Boixo, M. Broughton, B. B. Buckley, D. A. Buell, et al. "Hartree-Fock on a superconducting qubit quantum computer". In: *arXiv preprint arXiv:2004.04174* 369.6507 (2020), pp. 1084–1089.

[132] C. N. Self, K. E. Khosla, A. W. Smith, F. Sauvage, P. D. Haynes, J. Knolle, F. Mintert, and M. Kim. "Variational quantum algorithm with information sharing". In: *arXiv preprint arXiv:2103.16161* (2021).

[133] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp. "Quantum amplitude amplification and estimation". In: *Contemporary Mathematics* 305 (2002), pp. 53–74.

[134] A. Montanaro. "Quantum speedup of Monte Carlo methods". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 471.2181 (2015), p. 20150301.

[135] Y. Suzuki, S. Uno, R. Raymond, T. Tanaka, T. Onodera, and N. Yamamoto. "Amplitude estimation without phase estimation". In: *Quantum Information Processing* 19.2 (2020), pp. 1–17.

[136] T. Giurgica-Tiron, I. Kerenidis, F. Labib, A. Prakash, and W. Zeng. "Low depth algorithms for quantum amplitude estimation". In: *arXiv preprint arXiv:2012.03348* (2020).

[137] D. Grinko, J. Gacon, C. Zoufal, and S. Woerner. "Iterative quantum amplitude estimation". In: *npj Quantum Information* 7.1 (2021), pp. 1–6.

[138] D. Wang, O. Higgott, and S. Brierley. "Accelerated variational quantum eigensolver". In: *Physical review letters* 122.14 (2019), p. 140504. DOI: 10.1103/PhysRevLett.122.140504.

[139] S. U. Khan, A. J. Awan, and G. Vall-Llosera. "K-means clustering on noisy intermediate scale quantum computers". In: *arXiv preprint arXiv:1909.12183* (2019).

[140] I. F. Araujo, D. K. Park, F. Petruccione, and A. J. da Silva. "A divide-and-conquer algorithm for quantum state preparation". In: *Scientific Reports* 11.1 (2021), pp. 1–12.

[141] J. Romero, R. Babbush, J. R. McClean, C. Hempel, P. J. Love, and A. Aspuru-Guzik. "Strategies for quantum computing molecular energies using the unitary coupled cluster ansatz". In: *Quantum Science and Technology* 4.1 (2018), p. 014008.

[142] A. Yimsiriwattana and S. J. Lomonaco Jr. "Generalized GHZ states and distributed quantum computing". In: *arXiv preprint quant-ph/0402148* (2004).

[143] A. Y. Kitaev. "Quantum computations: algorithms and error correction". In: *Russian Mathematical Surveys* 52.6 (1997), p. 1191.

[144] M. Saeedi and M. Pedram. "Linear-depth quantum circuits for n-qubit Toffoli gates with no ancilla". In: *Physical Review A* 87.6 (2013), p. 062318.

[145] Y. He, M.-X. Luo, E. Zhang, H.-K. Wang, and X.-F. Wang. "Decompositions of n-qubit Toffoli gates with linear circuit complexity". In: *International Journal of Theoretical Physics* 56.7 (2017), pp. 2350–2361.

[146] D. Maslov. "Advantages of using relative-phase Toffoli gates with an application to multiple control Toffoli optimization". In: *Physical Review A* 93.2 (2016), p. 022311.

[147] R. Babbush, N. Wiebe, J. McClean, J. McClain, H. Neven, and G. K.-L. Chan. "Low-depth quantum simulation of materials". In: *Physical Review X* 8.1 (2018), p. 011044.

[148] K. Michielsen, M. Nocon, D. Willsch, F. Jin, T. Lippert, and H. De Raedt. "Benchmarking gate-based quantum computers". In: *Computer Physics Communications* 220 (2017), pp. 44–55.

[149] J. T. Seeley, M. J. Richard, and P. J. Love. "The Bravyi-Kitaev transformation for quantum computation of electronic structure". In: *The Journal of chemical physics* 137.22 (2012), p. 224109.

[150] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, M. S. Alam, S. Ahmed, J. M. Arrazola, C. Blank, A. Delgado, S. Jahangiri, et al. "Pennylane: Automatic differentiation of hybrid quantum-classical computations". In: *arXiv preprint arXiv:1811.04968* (2018).

[151] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu. "A Survey of Recent Results in Networked Control Systems". In: *Proceedings of the IEEE* 95.1 (2007), pp. 138–162. DOI: 10.1109/JPROC.2006.887288.

[152] T. C. Yang. "Networked control system: a brief survey". In: *IEE Proceedings-Control Theory and Applications* 153.4 (2006), pp. 403–412.

[153] M. S. Mahmoud and Y. Xia. *Networked control systems: cloud control and secure control*. Butterworth-Heinemann, 2019.

[154] K. E. Booth, M. Do, J. C. Beck, E. Rieffel, D. Venturelli, and J. Frank. "Comparing and integrating constraint programming and temporal planning for quantum circuit compilation". In: *arXiv preprint arXiv:1803.06775* (2018).

[155] A. Swales et al. "Open modbus/tcp specification". In: *Schneider Electric* 29 (1999).

[156] WhichNAS. *10Gbase-T vs. SFP+ - Which is the Best 10G Network Solution for Small Businesses*. June 2020. URL: https://whichnas.com/10gbase-t-vs-sfp-which-is-the-best-10g-network-solution-for-small-businesses/.

[157] K. Behrendt and K. Fodero. "Implementing MIRRORED BITS Technology Over Various Communications Media". In: *SEL Application Guide* 12 (2001).

[158] T. A. Youssef, M. M. Esfahani, and O. Mohammed. "Data-Centric Communication Framework for Multicast IEC 61850 Routable GOOSE Messages over the WAN in Modern Power Systems". In: *Applied Sciences* 10.3 (2020), p. 848.

[159] A. Apostolov. "R-GOOSE: what it is and its application in distribution automation". In: *CIRED-Open Access Proceedings Journal* 2017.1 (2017), pp. 1438–1441.

[160] M. Lipiński, T. Włostowski, J. Serrano, and P. Alvarez. "White rabbit: A PTP application for robust sub-nanosecond synchronization". In: *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE. 2011, pp. 25–30.

[161] T. Włostowski. "Precise time and frequency transfer in a White Rabbit network". PhD thesis. Instytut Radioelektroniki, 2011.

[162] S. Fukuda, Y. Fukuda, T. Hayakawa, E. Ichihara, M. Ishitsuka, Y. Itow, T. Kajita, J. Kameda, K. Kaneyuki, S. Kasuga, et al. "The super-kamiokande detector". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 501.2-3 (2003), pp. 418–462.

[163] C. R. Contaldi. "The OPERA neutrino velocity result and the synchronisation of clocks". In: *arXiv preprint arXiv:1109.6160* (2011).

[164] B. C. Sanders. "Review of entangled coherent states". In: *Journal of Physics A: Mathematical and Theoretical* 45.24 (2012), p. 244002.

[165] Q. O. S. Foundation. *List of Open Quantum Projects*. 2020. URL: https://qosf.org/project%5C_list/.

[166] A. Dahlberg and S. Wehner. "SimulaQron - a simulator for developing quantum internet software". In: *Quantum Science and Technology* 4.1 (2018). doi: 10.1088/2058-9565/aad56e, p. 015001. DOI: 10.1088/2058-9565/aad56e.

[167] B. Bartlett. "A distributed simulation framework for quantum networks and channels". In: *arXiv preprint arXiv:1808.07047* (2018).

[168] T. Matsuo, C. Durand, R. Satoh, R. Van Meter, S. Shigeya, S. Nagayama, T. Satoh, N. Tatetani, M. Nakai, S. Metwalli, T. Ladd, L. Aparicio, and P. Pathumsoot. *QuISP - Quantum Internet Simulation Package*. 2020. URL: https://aqua.sfc.wide.ad.jp/quisp%5C_website/.

[169] X. Wu, A. Kolar, J. Chung, D. Jin, T. Zhong, R. Kettimuthu, and M. Suchara. "SeQUeNCe: A Customizable Discrete-Event Simulator of Quantum Networks". In: *arXiv preprint arXiv:2009.12000* (2020).

[170] M. Mehic, O. Maurhart, S. Rass, and M. Voznak. "Implementation of quantum key distribution network simulation module in the network simulator NS-3". In: *Quantum Information Processing* 16.10 (2017). doi: 10.1007/s11128-017-1702-z, pp. 1–23. DOI: 10.1007/s11128-017-1702-z.

[171] R. Chatterjee, K. Joarder, S. Chatterjee, B. C. Sanders, and U. Sinha. "qkdSim, a Simulation Toolkit for Quantum Key Distribution Including Imperfections: Performance Analysis and Demonstration of the B92 Protocol Using Heralded Photons". In: *Phys. Rev. Applied* 14 (2 2020). doi: 10.1103/PhysRevApplied.14.024036, p. 024036. DOI: 10.1103/PhysRevApplied.14.024036. URL: https://link.aps.org/doi/10.1103/PhysRevApplied.14.024036.

[172] T. Matsuo. "Simulation of a Dynamic, RuleSet-based Quantum Network". In: *arXiv preprint arXiv:1908.10758* (2019).

[173] H. J. Kimble. "The quantum internet". In: *Nature* 453.7198 (2008). doi: 10.1038/nature07127, pp. 1023–1030. DOI: 10.1038/nature07127.

[174] G. F. Riley and T. R. Henderson. "The ns-3 network simulator". In: *Modeling and tools for network simulation*. Springer, 2010, pp. 15–34.

[175] R. R. Fontes, S. Afzal, S. H. Brito, M. A. Santos, and C. E. Rothenberg. "Mininet-WiFi: Emulating software-defined wireless networks". In: *2015 11th International Conference on Network and Service Management (CNSM)*. doi: 10.1109/CNSM.2015.7367387. IEEE. 2015, pp. 384–389. DOI: 10.1109/CNSM.2015.7367387.

[176] S. DiAdamo, J. Nötzel, B. Zanger, and M. Mert Beşe. *QuNetSim: A Software Framework for Quantum Networks*. 2020. URL: https://tqsd.github.io/QuNetSim.

[177] D. S. Steiger, T. Häner, and M. Troyer. "ProjectQ: an open source software framework for quantum computing". In: *Quantum* 2 (2018). doi: 10.22331/q-2018-01-31-49, p. 49. DOI: 10.22331/q-2018-01-31-49.

[178] B. Zanger and S. DiAdamo. *EQSN: Effective Quantum Simulator for Networks*. 2020. URL: https://github.com/tqsd/EQSN_python.

[179] J. R. Johansson, P. D. Nation, and F. Nori. "QuTiP: An open-source Python framework for the dynamics of open quantum systems". In: *Computer Physics Communications* 183.8 (2012). doi: 10.1016/j.cpc.2012.02.021, pp. 1760–1772. DOI: 10.1016/j.cpc.2012.02.021.

[180] G. Van Rossum et al. "Python Programming Language." In: *USENIX annual technical conference*. Vol. 41. 2007, p. 36.

[181] IBM. *Qiskit Runtime Services*. 2021. URL: https://quantum-computing.ibm.com/services?program=circuit-runner.

[182] T. Coopmans, R. Knegjens, A. Dahlberg, D. Maier, L. Nijsten, J. de Oliveira Filho, M. Papendrecht, J. Rabbie, F. Rozpędek, M. Skrzypczyk, et al. "NetSquid, a NETwork Simulator for QUantum Information using Discrete events". In: *Communications Physics* 4.1 (2021), pp. 1–15.

[183] X. Wu, A. Kolar, J. Chung, D. Jin, T. Zhong, R. Kettimuthu, and M. Suchara. "SeQUeNCe: a customizable discrete-event simulator of quantum networks". In: *Quantum Science and Technology* 6.4 (2021), p. 045027.

[184] Y. Pu, N. Jiang, W. Chang, H. Yang, C. Li, and L. Duan. "Experimental realization of a multiplexed quantum memory with 225 individually accessible memory cells". In: *Nature communications* 8.1 (2017), pp. 1–6.

[185] S. Sekavčnik, S. DiAdamo, J. Nötzel, and R. Bassoli. *QontainerNet*. 2021. URL: https://github.com/tqsd/QontainerNet.

[186] G. M. Dias, M. Nurchis, and B. Bellalta. "Adapting sampling interval of sensor networks using on-line reinforcement learning". In: *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. 2016, pp. 460–465. DOI: 10.1109/WF-IoT.2016.7845391.

[187] M.-H. Hsieh and M. M. Wilde. "Trading classical communication, quantum communication, and entanglement in quantum Shannon theory". In: *IEEE Transactions on Information Theory* 56.9 (2010), pp. 4705–4730.

[188] U. Pereg, C. Deppe, and H. Boche. "Quantum broadcast channels with cooperating decoders: An information-theoretic perspective on quantum repeaters". In: *Journal of Mathematical Physics* 62.6 (2021), p. 062204.

[189] M. Christandl and S. Wehner. "Quantum anonymous transmissions". In: *International Conference on the Theory and Application of Cryptology and Information Security*. doi: 10.1007/11593447_12. Springer. 2005, pp. 217–235. DOI: 10.1007/11593447_12.

[190] J. Nötzel and S. DiAdamo. "Entanglement-Enhanced Communication Networks". In: *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. doi: 10.1109/QCE49297.2020.00038. 2020, pp. 242–248. DOI: 10.1109/QCE49297.2020.00038.