

Master's Thesis
Academic Year 2025

Performance Analysis of QAOA on Distributed
Quantum Network Topologies Using SwitchQNet

Graduate School of Media and Governance,
Keio University

Samanvay Sharma

Abstract of Master's Thesis Academic Year 2025

Performance Analysis of QAOA on Distributed Quantum Network Topologies Using SwitchQNet

Category: Science / Engineering

Summary

This thesis evaluates the Quantum Approximate Optimization Algorithm, QAOA, across configurations of the Clos, Fat-Tree and Spine-Leaf network topologies used for quantum data centers (QDCs), leveraging the SwitchQNet compiler to benchmark entanglement routing efficiency and algorithm scalability. While existing quantum compilers like SwitchQNet provide foundational tools for distributed quantum computing (DQC), their performance remains underexplored for structured topologies central to modern QDC architectures, as well as for a broad range of quantum algorithms. By integrating QAOA into SwitchQNet's workflow and analyzing its execution under selected interconnects, this work quantifies trade-offs in entanglement latency, throughput, and solution quality. The results establish topology-specific guidelines for deploying QAOA in distributed settings, advancing practical compiler design for near-term quantum applications. Contributions include a comparative framework for network topology-aware QAOA benchmarking and improved codebase for ease-of-access and collaborative future co-development.

Keywords:

Quantum Computing, Quantum Networking, Distributed Quantum Computing, Quantum Data Centers, Quantum Network Compilation, Performance Benchmarking

Graduate School of Media and Governance, Keio University

Samanvay Sharma

Contents

1	Introduction	1
1.1.	Background	1
1.2.	Contributions	2
1.3.	Thesis Outline	4
2	Overview of Quantum Computing and Quantum Networks	5
2.1.	Introduction to quantum computing	5
2.1.1	Quantum mechanics	5
2.1.2	The ”qubit”	7
2.1.3	Dirac notation	8
2.1.4	Density Matrices and Unitary (Reversible) Evolution . . .	10
2.1.5	Bloch sphere representation	12
2.1.6	Quantum gates	12
2.1.7	Pauli Basis	13
2.1.8	Measurement and Probability in Quantum Systems	15
2.1.9	Quantum Circuit Model and Multi-Qubit Gates	15
2.1.10	Fidelity and Verification	16
2.1.11	Decoherence and Entangled States	16
2.1.12	Entangled states	17
2.1.13	Bell states	18
2.1.14	Clifford Operations	18
2.1.15	Non-Clifford Operations	19
2.1.16	Some Notable Protocols and Mechanisms	19
2.1.17	DiVincenzo’s Criteria	24
2.2.	Introduction to quantum communications	24

2.2.1	Classical and Quantum Networks: Fundamental Differences	24
2.2.2	Quantum Networks	25
2.2.3	Quantum Network Scales and Architectures	29
2.2.4	Classical Switched Network Topologies	30
2.2.5	Distributed Quantum Computing	33
2.2.6	Quantum Data Centers	36
2.2.7	Quantum Error Correction in DQC and QDCs	40
3	Motivations and Problem Statement	42
3.1.	The SwitchQNet compiler	42
3.2.	The Quantum Approximate Optimization Algorithm	50
3.3.	Problem Motivation	54
4	Methodology and Experimental Results	57
4.1.	Solution methodology	57
4.2.	Experiment Setup	60
4.2.1	Architecture Setup	60
4.2.2	Benchmark Programs	61
4.2.3	Metrics	62
4.2.4	Comparison Baseline	63
4.3.	Primary Experiment Results with QAOA	64
4.3.1	Interpretation of Performance Metrics	65
4.4.	Latency	68
4.5.	Compilation Switches	70
4.6.	Sensitivity analysis	70
4.7.	QEC Integration	79
5	Conclusion and Future Works	81
	Availability	82
	References	83
	Appendix	90
A.	Appendix: Quantum Gate Representations and Circuits	90
B.	Appendix: Mathematical Framework of Quantum Information	93

C. Additional Record of Results	94
---	----

List of Figures

2.1	Entanglement swapping	23
2.2	Generic entanglement distillation (purification)	23
2.3	Clos topology	34
2.4	Spine-leaf topology	34
2.5	Fat-tree topology	35
2.6	QDC with switch-linked QPUs	39
3.1	QDC abstraction for SwitchQNet	44
3.2	EPR pair abstraction in SwitchQNet	44
3.3	Preprocessing in SwitchQNet	45
3.4	Abstraction of post-split distillation	47
3.5	SwitchQNet Compilation flowchart	49
3.6	QAOA circuit	52
3.7	QAOA graphs	53
4.1	In-rack vs. Cross-rack EPR pair percentages	69
4.2	Latency percentages	69
4.3	Relative Buffer size vs. Latency	71
4.4	Look-ahead depth vs. Latency	71
4.5	Latency with #Comm. qubit/QPU	72
4.6	Cross-rack EPR latency (reconfig.)	73
4.7	In-rack EPR latency (reconfig.)	73
4.8	QAOA Cross-rack fidelity vs. EPR overhead%	75
4.9	Cross-rack fidelity vs. EPR overhead%	75
4.10	QAOA Distilled in-rack fidelity vs. EPR overhead%	76
4.11	Distilled in-rack fidelity vs. EPR overhead%	77

4.12 QAOA EPR pairs per distillation vs. Latency	77
4.13 EPR pairs per distillation vs. Latency	78
4.14 Overall Improvement Factor	78
4.15 Post-QEC Latency Improvement	80

List of Tables

4.1	Program and architecture settings	61
4.2	Compiler Performance Across Different Quantum Benchmarks and Architecture Settings.	67
4.3	QEC for 3-regular QAOA	80
5.1	Single-Qubit Pauli Gates	90
5.2	Single-Qubit Clifford and Rotation Gates	91
5.3	Multi-Qubit Gates with Compressed Matrices	92
5.4	QEC for random QAOA	94

Chapter 1

Introduction

1.1. Background

Quantum computing has emerged as a promising computational paradigm with the potential to address classes of problems that are intractable for classical computers, including combinatorial optimization, quantum simulation, and cryptographic analysis. In recent years, experimental quantum devices have demonstrated proof-of-concept implementations of several quantum algorithms. However, current hardware platforms remain fundamentally constrained by limited qubit counts, short coherence times, and high error rates. As a result, most existing systems fall into the category of Noisy Intermediate-Scale Quantum (NISQ) devices, where scalability and reliability remain significant challenges.

Quantum error correction (QEC) provides a theoretical pathway toward fault-tolerant quantum computation by encoding logical qubits into many physical qubits. While QEC can, in principle, suppress noise and enable arbitrarily long computations, its practical realization requires a substantial overhead in physical qubits and control resources. Current and near-term hardware platforms are therefore unlikely to support large-scale fault-tolerant computation within a single quantum processor, motivating alternative approaches to scaling quantum systems.

Distributed quantum computing addresses this challenge by interconnecting multiple quantum processing units (QPUs) through quantum and classical communication channels. Rather than relying on a single monolithic device, a dis-

tributed architecture enables quantum states and operations to be shared across spatially separated nodes using entanglement as a fundamental resource. Such architectures are particularly relevant in short-distance settings such as quantum data centers, where optical interconnects are able to support high-rate entanglement generation without the need for long-distance quantum repeaters. In these environments, performance is jointly determined by local quantum operations and the efficiency of communication and coordination across the network.

The distributed nature of these systems introduces challenges that do not arise in standalone quantum processors. Entanglement generation, consumption, and scheduling impose constraints on execution order, latency, and resource availability. Consequently, there is a growing need for network-aware quantum compilers that are able to explicitly account for communication costs, routing constraints, and resource contention when mapping quantum programs onto distributed hardware.

To study these challenges, recent work has proposed SwitchQNet, a network-aware distributed quantum compiler designed to schedule quantum communication and computation across data-center-style quantum architectures. SwitchQNet models collections of QPUs interconnected through multi-stage switching fabrics and explicitly incorporates entanglement generation, communication latency, and resource contention into the compilation process. By simulating the interaction between quantum programs and network architectures, SwitchQNet provides a framework for evaluating the performance of distributed quantum systems under practical constraints.

While SwitchQNet represents an important step toward realistic evaluation of distributed quantum computing, its existing studies primarily focus on a limited set of benchmark programs, such as arithmetic circuits and structured algorithmic workloads. As a result, the behavior of optimization-oriented algorithms under distributed communication constraints remains insufficiently explored. Addressing this gap forms the central motivation for the work presented in this thesis.

1.2. Contributions

The primary contributions of this work are as follows.

First, we design and integrate an implementation of the Quantum Approximate Optimization Algorithm (QAOA) into the SwitchQNet compilation framework. QAOA is a variational algorithm for combinatorial optimization whose structure is defined by a problem Hamiltonian derived from an input graph. Although QAOA is commonly studied on single, near-term quantum devices, its repeated use of two-qubit interactions makes its performance sensitive to qubit placement and communication constraints in distributed settings. Incorporating QAOA into SwitchQNet enables systematic evaluation of optimization workloads within a network-aware compilation pipeline.

Second, we perform a comprehensive experimental evaluation of distributed QAOA execution using SwitchQNet across multiple architectural parameters and network topologies. These experiments analyze the impact of rack scaling, QPU density, interconnect topology, communication latency, entanglement fidelity, and quantum error correction overhead. By comparing QAOA against existing benchmark programs supported by SwitchQNet, we highlight qualitative differences in how optimization workloads respond to distributed communication constraints.

Third, we conduct sensitivity analyses of compiler optimizations and communication mechanisms within SwitchQNet, including scheduling depth, look-ahead strategies, and entanglement management policies. The results provide insight into how algorithm structure and network characteristics jointly influence performance, and demonstrate that optimization algorithms such as QAOA exhibit distinct scaling behavior compared to arithmetic and circuit-oriented benchmarks.

Finally, this work improves the usability and reproducibility of the SwitchQNet framework by clarifying the execution flow of the compiler and documenting the relationship between its conceptual protocols and their implementation. These improvements lower the barrier for future extensions and enable more systematic exploration of distributed quantum algorithms.

To the best of our knowledge, this is the first study to evaluate a graph-structured quantum optimization algorithm within SwitchQNet across multiple data-center-style network topologies while accounting for network-aware scheduling and fault-tolerant communication effects.

1.3. Thesis Outline

Chapter 2 discusses all the preliminary information and topics that should be sufficient to understand the basis of the research and proposed solution and terminology, discussing the basics of quantum mechanics, the basic tools involved in quantum computation, quantum networking and distribution quantum computation. Chapter 3 takes a look at the quantum networking compiler SwitchQNet, discusses advances and limitations of current solutions, and proposes a solution using QAOA that takes into account these oversights and aims to approach them using the background knowledge established in previous chapters. Chapter 4 discusses the proposed methodology in detail and presents the results of the developed solutions, making inferences from our findings. Chapter 5 leads to conclusions drawn from the compilation of resources and the comparison of results.

Chapter 2

Overview of Quantum Computing and Quantum Networks

2.1. Introduction to quantum computing

2.1.1 Quantum mechanics

Quantum computing is founded on the principles of quantum mechanics, the physical theory that governs the behavior of matter and energy at microscopic scales. Unlike classical computing, which relies on deterministic binary states, quantum computing exploits physical phenomena that have no classical counterpart, enabling new models of information representation and processing. Quantum mechanics provides the mathematical framework for describing the evolution and measurement of quantum systems, forming the basis upon which concepts such as superposition, measurement, and entanglement are built. Understanding these foundational principles is essential for explaining how quantum information is encoded and manipulated in quantum computing systems.

Superposition

A quantum particle is able to exist in a linear combination of multiple states, defined as a “superposition”. This superposition exists until the particle is acted upon by any kind of stimuli external to its system or environment, effectively making an “observation” over this quantum mechanical system, after which it

assumes a definite state and loses its superposition property, i.e. collapses. Physically, this could be thought of as a particle exhibiting wave nature in a system that upon observation collapses into a discrete unit of classical information, i.e. exhibits particle nature.

Entanglement

This phenomenon also occurs due to quantum mechanical properties, and implies that two or more particles under quantum effects may be considered as “entangled”, which means that these particles have correlated states across any amount of distance. This information within the quantum particles may be in the form of spin states, electronic configuration, any combination of statistical properties exhibited by these particles would be the same and behave similarly.

Interference

Quantum interference is a fundamental phenomenon arising from the principle of superposition, where the probability amplitudes of a system’s possible states combine to determine the final probability distribution of an observable. Unlike classical waves, quantum interference occurs at the level of the wavefunction Ψ . For a system that evolves through multiple indistinguishable paths, the total probability amplitude is the sum of the amplitudes for each path, $\Psi_{\text{total}} = \psi_1 + \psi_2$, resulting in a probability $P = |\Psi_{\text{total}}|^2 = |\psi_1|^2 + |\psi_2|^2 + 2\text{Re}(\psi_1^* \psi_2)$. The term $2\text{Re}(\psi_1^* \psi_2)$ is the interference term, facilitating constructive interference to maximize the probability of optimal states and destructive interference to suppress incorrect ones. In distributed architectures, maintaining the phase coherence required for this phenomenon is critical, necessitating high-fidelity entanglement and efficient protocols to mitigate the decoherence (or loss of superposition or phase coherence) that would otherwise degrade performance.

Schrodinger’s Equation

Erwin Schrodinger laid out one of the most fundamental results in quantum mechanical theory which begins with a thought experiment of a cat kept alive in a box with an external mechanism to kill the cat if certain conditions are met. This

cat is both alive and dead at the same time as long as the box remains closed, i.e. the quantum system is in a superposition of its possible states as long as the system is unobserved. Upon taking a look at the box to see if the cat is dead or alive, i.e. upon making an observation within a quantum mechanical system, the mechanism to kill the cat either activates or doesn't activate with certain probability, and the cat is confirmed to be either dead or alive, i.e. the quantum system collapses and we obtain a classical result with certainty.

In this sense, Schrodinger laid out an equation to describe these dynamics of a quantum system and the pattern of changes in such a system with or without the effects of time (unitary evolution). The most basic form of this equation is:

$$H|\psi\rangle = \frac{ih}{2\pi i} \frac{\delta|\psi\rangle}{\delta t}, \quad (2.1)$$

The No-Cloning Theorem

A critical operational constraint in quantum computing and subsequently quantum networking is the **No-Cloning Theorem**, which forbids the creation of an identical copy of an arbitrary unknown quantum state. This theorem is a direct consequence of the linearity of quantum mechanics.

Proof: Assume a unitary U exists such that $U(|\psi\rangle \otimes |0\rangle) = |\psi\rangle \otimes |\psi\rangle$ for any $|\psi\rangle$. If we apply this to a superposition $|\alpha\rangle = a|0\rangle + b|1\rangle$, linearity requires:

$$U(|\alpha\rangle \otimes |0\rangle) = a(|0\rangle \otimes |0\rangle) + b(|1\rangle \otimes |1\rangle) \quad (2.2)$$

However, a true copy would be $|\alpha\rangle \otimes |\alpha\rangle = a^2|00\rangle + ab|01\rangle + ab|10\rangle + b^2|11\rangle$. These states are unequal ($ab \neq 0$), proving that general cloning is impossible. Consequently, quantum networks cannot use "store-and-forward" repeaters; they must instead rely on entanglement swapping.

where ψ is the quantum state vector, H or the Hamiltonian is the square matrix vector describing the solution space of the system.

2.1.2 The "qubit"

Similar to how bits represent the fundamental unit of classical computation and information, a quantum bit or "qubit" is the analogue of the classical bit for

quantum computation storing information in binary states represented as zeros and ones, up and down electron spins, on and off switches, vertical and horizontal polarized photons, or any similar variable state, and giving a single output upon measurement. However, what differentiates qubits from classical bits is their ability to simultaneously exist in a superposition or a linear combination of these two states as long as the system of these qubits is isolated and unobserved. Upon observation, this superposition is destroyed and the output renders to one of these two states. The advantage here is that a system of “n” qubits may be evaluated for 2^n combinations of states, thus allowing for computation of highly complex systems that classical computers may not be able to simulate in a reasonable amount of time.

2.1.3 Dirac notation

One of the ways to represent quantum information states, operations and measurements using linear algebra, popularly borrowed by physicists, is called the “Dirac” notation named after Paul Dirac who developed it in the 1930s which utilizes both the vector and matrix notations of quantum states and operators.

A ket vector could be represented as a column matrix of the format:

$$|\alpha_n\rangle = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{pmatrix} \quad (2.3)$$

A bra vector could be represented as a row matrix of the format:

$$\langle\alpha_n| = (\alpha_1 \ \alpha_2 \ \dots \ \alpha_m) \quad (2.4)$$

We also describe this as the “bra-ket” notation.

For example, when utilizing qubits for encoding information into quantum states, their Dirac notation is described as $|0\rangle$ and $|1\rangle$ where,

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ and } |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

We shall be following this standard notation for dealing with quantum mechanical states.

To represent any arbitrary quantum state, we define a **basis** as a set of linearly independent vectors that spans the entire Hilbert space which is the state space equipped with a vector product where all of the quantum state vectors live. In the computational basis $\{|0\rangle, |1\rangle\}$, linear independence ensures that $|0\rangle$ cannot be expressed as a scalar multiple of $|1\rangle$, allowing them to represent distinct binary states. When we apply quantum gates, we are essentially performing a change of basis or a rotation within this space. For instance, the Hadamard gate maps the computational basis to the "sign" or "diagonal" basis $\{|+\rangle, |-\rangle\}$, enabling the parallel state processing that characterizes quantum speedup.

The behavior of a quantum gate U is fundamentally defined by its **eigenvectors** and **eigenvalues**. If a gate acts on a state $|\psi\rangle$ such that $U|\psi\rangle = \lambda|\psi\rangle$, then $|\psi\rangle$ is an eigenvector and the complex scalar λ is its eigenvalue. In the context of quantum algorithms, such as Quantum Phase Estimation, the primary goal is to extract the phase information stored in these eigenvalues. For software developers, this is analogous to identifying the "pure functions" of a system—states that remain stable under a specific transformation, merely picking up a phase factor that encodes the result of a computation.

While the inner product reduces two vectors to a scalar, the **outer product** $|\phi\rangle\langle\psi|$ creates an operator. This is particularly useful for constructing **projection operators**, such as $P_0 = |0\rangle\langle 0|$, which are used to describe the state of a system after a measurement. By representing gates and measurements as sums of outer products, we are able to decompose complex algorithms into a series of discrete, verifiable linear transformations. This allows us to define the "action" of a quantum program by how it maps input basis states to output configurations.

Building on the vector space properties of the Dirac formalism, we extend these operations to describe how quantum information is structured and manipulated within algorithms. This is viewed as defining the "data types" (bases) and the "functions" (eigenvalues/eigenvectors) that govern the transformation of information. This notation is analogous to an abstraction layer that treats quantum states as elements of a complex vector space—specifically a Hilbert space—while hiding the underlying coordinate-specific details until a specific basis is chosen.

See Appendix B for a list of Dirac operations.

2.1.4 Density Matrices and Unitary (Reversible) Evolution

While the state vector formalism is sufficient for describing isolated quantum systems prepared in pure states, it becomes inadequate when dealing with realistic scenarios involving statistical uncertainty, subsystem descriptions, or interactions with an external environment. In such cases, the density matrix, also referred to as the density operator, provides the appropriate mathematical framework. This formalism is central to quantum information science and quantum networking, where decoherence, noise, and partial system access are unavoidable.

The Density Operator Formalism

A quantum system is said to be in a *pure state* if it is described by a single state vector $|\psi\rangle$ in a Hilbert space. In practice, however, one often encounters situations in which the system is prepared in one of several possible pure states $\{|\psi_i\rangle\}$, each occurring with a classical probability p_i . Such a statistical ensemble is described as a *mixed state*.

The density operator ρ provides a unified description of both pure and mixed states and is defined as

$$\rho \equiv \sum_i p_i |\psi_i\rangle \langle \psi_i|, \quad (2.5)$$

where $p_i \geq 0$ and $\sum_i p_i = 1$. Importantly, different ensembles may correspond to the same density operator, reflecting the fact that ρ captures all physically observable information about the quantum state.

For a density operator to represent a physically valid quantum state, it must satisfy the following properties [41]:

- **Hermiticity:** $\rho = \rho^\dagger$.
- **Trace normalization:** $\text{Tr}(\rho) = 1$.
- **Positive semi-definiteness:** $\langle \phi | \rho | \phi \rangle \geq 0$ for all vectors $|\phi\rangle$.

The positive semi-definite property ensures that all measurement probabilities derived from ρ are non-negative. A quantum state is pure if and only if

$\text{Tr}(\rho^2) = 1$. If $\text{Tr}(\rho^2) < 1$, the state is mixed. This criterion plays a crucial role in quantum communication and distributed quantum computing, where interaction with external systems or loss of coherence results in mixed-state behavior [32].

Unitary Evolution of Closed Quantum Systems

The time evolution of a closed quantum system is governed by unitary transformations. If a system is initially described by a density operator ρ , its evolution under a unitary operator U is given by

$$\rho' = U\rho U^\dagger, \quad (2.6)$$

where U satisfies $U^\dagger U = I$. Unitary evolution preserves the trace, Hermiticity, and eigenvalues of the density operator, and therefore preserves the purity of the quantum state.

In physical terms, unitary evolution corresponds to ideal quantum gates or coherent propagation through a lossless medium. More generally, unitary operators arise from the system Hamiltonian H via

$$U(t) = \exp\left(-\frac{i}{\hbar}Ht\right), \quad (2.7)$$

which describes reversible, deterministic dynamics within the system's Hilbert space. In the context of quantum computation, this evolution represents the execution of a quantum circuit in the absence of noise or measurement.

Open Systems, Decoherence, and Communication Context

In realistic quantum communication and distributed computing settings, quantum systems are rarely closed. Interaction with an environment leads to the gradual loss of coherence, a process known as decoherence. Within the density matrix formalism, decoherence manifests as the suppression of off-diagonal elements, which encode phase relationships between basis states.

As decoherence progresses, a quantum state may approach the maximally mixed state. For a single qubit, this state is given by

$$\rho = \frac{I}{2}, \quad (2.8)$$

which contains no extractable quantum information and corresponds to a uniform probability distribution over measurement outcomes. At this point, the qubit no longer exhibits quantum behavior and effectively behaves as a classical random variable.

This formalism provides the mathematical foundation for analyzing noise, entanglement degradation, and fidelity loss in quantum networks. It underpins later discussions of entanglement distribution, error correction, and resource scheduling in distributed quantum systems, where density matrices are essential for modeling non-ideal, open-system dynamics.

2.1.5 Bloch sphere representation

For a single-qubit system ($d = 2$), the density matrix may be decomposed using the Pauli basis $\{I, \sigma_x, \sigma_y, \sigma_z\}$. Any density matrix ρ could be written as:

$$\rho = \frac{I + \mathbf{r} \cdot \vec{\sigma}}{2} = \frac{1}{2} \begin{pmatrix} 1 + r_z & r_x - ir_y \\ r_x + ir_y & 1 - r_z \end{pmatrix} \quad (2.9)$$

where $\mathbf{r} = (r_x, r_y, r_z)$ is the **Bloch vector**. This provides a geometric mapping where:

- Points on the surface ($|\mathbf{r}| = 1$) represent pure states.
- Points in the interior ($|\mathbf{r}| < 1$) represent mixed states.

2.1.6 Quantum gates

Quantum computation may be realized through several computational paradigms, including the circuit model, measurement-based quantum computation, adiabatic quantum computation, and topological approaches. In this work, we focus on the circuit model of quantum computation, which represents algorithms as sequences of quantum gates acting on qubits. This model provides a natural and widely adopted framework for analyzing quantum algorithms and is particularly well suited for studying the interaction between computation and communication in distributed quantum systems.

In the circuit model, quantum gates are the physical realizations of unitary operators acting on quantum states. Unlike classical logic gates, which operate irreversibly on binary values, quantum gates must be reversible and are represented by unitary matrices. Any computation that is expressible as a unitary transformation on an n -qubit system, corresponding to a $2^n \times 2^n$ unitary matrix, could in principle be decomposed into a sequence of quantum gates and implemented as a quantum circuit.

Single-qubit gates act locally on individual qubits and are used to manipulate basis states, relative phases, and superposition. Examples include the Pauli gates, which perform basis flips and phase operations, and the Hadamard gate, which maps computational basis states into equal superpositions. These gates enable the preparation and control of quantum states and serve as the building blocks for more complex operations.

Multi-qubit gates introduce interactions between qubits within the same circuit. Among these, the controlled-NOT (CNOT or CX) gate plays a central role, as it enables the creation of entanglement and allows conditional operations between qubits. Entangling gates are essential for achieving quantum advantage, as they generate correlations that cannot be reproduced by classical systems. Together, single-qubit and multi-qubit gates form a universal gate set capable of implementing arbitrary quantum algorithms within the circuit model.

See Appendix 5.1 for a list of gates.

2.1.7 Pauli Basis

The Pauli basis provides a fundamental operator representation for single-qubit quantum states and transformations. It consists of the identity operator and the three Pauli matrices, which together form a complete basis for the space of 2×2 Hermitian operators. Any single-qubit density matrix or Hamiltonian may be expressed as a linear combination of these operators, making the Pauli basis central to the mathematical description of quantum computation and control.

The Pauli matrices are defined as

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad (2.10)$$

together with the identity operator I . These operators correspond to π -rotations of the qubit state about the x , y , and z axes of the Bloch sphere, respectively, and generate the $\mathfrak{su}(2)$ Lie algebra that underlies single-qubit unitary evolution.

In this representation, an arbitrary single-qubit density matrix ρ may be written as

$$\rho = \frac{1}{2} (I + \vec{r} \cdot \vec{\sigma}), \quad (2.11)$$

where $\vec{r} = (r_x, r_y, r_z)$ is the Bloch vector and $\vec{\sigma} = (\sigma_x, \sigma_y, \sigma_z)$. This decomposition provides a direct geometric interpretation of qubit states and highlights the role of Pauli operators as generators of rotations on the Bloch sphere.

The Pauli basis is also essential for describing quantum gates and time evolution. Single-qubit unitary operations may be expressed as rotations about axes defined by linear combinations of Pauli operators, allowing quantum gates to be interpreted as controlled geometric transformations of the qubit state. This formalism naturally leads to the definition of general rotation operators and their special cases, which are widely used in quantum circuits and variational algorithms such as QAOA.

The general rotation operator about a Cartesian axis $\hat{n} = (n_x, n_y, n_z)$ in the Bloch basis by an angle θ is defined as:

$$R_{\hat{n}}(\theta) = \exp \left(-i \frac{\theta}{2} \hat{n} \cdot \vec{\sigma} \right) = \cos \left(\frac{\theta}{2} \right) I - i \sin \left(\frac{\theta}{2} \right) (n_x \sigma_x + n_y \sigma_y + n_z \sigma_z) \quad (2.12)$$

The rotation matrices for these operators are:

- **Rotation about the x -axis:**

$$R_x(\theta) = e^{-i\theta\sigma_x/2} = \begin{pmatrix} \cos(\theta/2) & -i \sin(\theta/2) \\ -i \sin(\theta/2) & \cos(\theta/2) \end{pmatrix} \quad (2.13)$$

- **Rotation about the y -axis:**

$$R_y(\theta) = e^{-i\theta\sigma_y/2} = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix} \quad (2.14)$$

- **Rotation about the z -axis:**

$$R_z(\theta) = e^{-i\theta\sigma_z/2} = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix} \quad (2.15)$$

2.1.8 Measurement and Probability in Quantum Systems

Measurement plays a fundamental role in quantum mechanics, as it provides the interface between abstract quantum states and observable classical outcomes. Unlike classical systems, where measurement simply reveals a pre-existing value, the act of measurement in a quantum system is intrinsically probabilistic and alters the state being measured. As a result, the outcomes of quantum measurements are predicted only in terms of probabilities.

Given a quantum state described by a state vector $|\psi\rangle$, the probability of obtaining a particular measurement outcome is determined by the projection of the state onto the corresponding measurement basis. If $\{|m_i\rangle\}$ denotes an orthonormal set of measurement basis states, the probability of observing outcome i is given by the Born rule,

$$p(i) = |\langle m_i | \psi \rangle|^2. \quad (2.16)$$

After measurement, the quantum state collapses to the basis state associated with the observed outcome, reflecting the irreversible nature of the measurement process.

This probabilistic description extends naturally to mixed states represented by a density operator ρ . In this case, the probability of obtaining outcome i is

$$p(i) = \text{Tr}(\rho M_i), \quad (2.17)$$

where M_i is the measurement operator corresponding to outcome i . Measurement thus provides the mechanism by which quantum information is converted into classical information, enabling tasks such as state verification, algorithm output extraction, and feedback control.

In most scenarios, unless specified otherwise, measurement of a given quantum state is performed in the Pauli Z basis, also known as the **computational basis**. Additionally, unless stated explicitly, we shall restrict our discussion to projective measurements throughout this thesis.

2.1.9 Quantum Circuit Model and Multi-Qubit Gates

Computation is modeled as a sequence of unitary gates applied to qubit registers. While single-qubit gates (like the Hadamard H and Phase S) allow for

superposition, **multi-qubit gates** are required to create entanglement and execute conditional logic. The standard entangling gate is the **Controlled-NOT (CNOT)**, defined as:

$$\text{CNOT} = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (2.18)$$

A set of gates is considered **universal** if it is able to approximate any unitary operation to arbitrary precision. The CNOT gate, combined with single-qubit rotations, constitutes a universal set.

2.1.10 Fidelity and Verification

To benchmark the performance of quantum channels and gates, we utilize the metric of **Fidelity**. For a pure target state $|\psi\rangle$ and an experimental density matrix ρ , fidelity is defined as the overlap:

$$F(|\psi\rangle, \rho) = \langle\psi|\rho|\psi\rangle \quad (2.19)$$

In general, for two mixed states ρ and σ , the Uhlmann fidelity is $F(\rho, \sigma) = (\text{Tr}\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}})^2$. In quantum networking, fidelity thresholds are critical; for example, a fidelity $F > 0.5$ is strictly required to verify that a shared state contains genuine quantum entanglement rather than classical correlation. States falling below this threshold must be purified or discarded.

2.1.11 Decoherence and Entangled States

A central challenge in quantum information processing is maintaining quantum coherence in the presence of unavoidable interactions with the surrounding environment. In an ideal isolated system, a quantum state evolves according to unitary dynamics and preserves phase relationships between its components. In practice, however, quantum systems are open, and interactions with external degrees of freedom lead to **decoherence** where quantum superposition is progressively degraded and the system's behavior becomes effectively classical.

Decoherence could be understood as the loss of phase coherence in a quantum state when it becomes correlated with environmental states that are not accessible or controlled. For example, a pure state

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

may evolve into a mixed state when environmental interactions randomize the relative phase between the basis states. This loss of coherence directly impacts the reliability of quantum computation and communication, as it limits how long quantum information is stored and manipulated before errors dominate.

2.1.12 Entangled states

An entangled state involves non-classical correlations between two or more subsystems such that the state of each subsystem cannot be described independently of the others. A simple example is the two-qubit entangled state

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle),$$

which exhibits perfect correlations between measurement outcomes on the two qubits. While such states are a key resource for quantum communication and distributed quantum computation, environmental interactions acting on any one subsystem might degrade the shared correlations and reduce the usefulness of the entangled state.

Some key entangled states include Bell states (and further generalized multi-party GHZ states or W states) and graph states. Given that they are the basis for most of the quantum computation processes and protocols that are involved in communication such as entanglement and teleportation, they are also vital in providing the essential elements for distributed computation utilizing quantum effects.

In the context of quantum networks and distributed quantum computing, decoherence plays a critical role because entangled states must often be generated, stored, and consumed across spatially separated nodes, but they are particularly sensitive to decoherence. The finite lifetime of entanglement places fundamental constraints on communication latency, resource scheduling, and network architecture. As a result, understanding decoherence and its effect on entangled states is

essential for motivating the need for careful resource management, fault-tolerant techniques, and network-aware compilation strategies discussed in later chapters of this thesis.

2.1.13 Bell states

Named after John Bell, these are the simplest non-trivial example of maximally entangled two-qubit state meaning that the qubits in a Bell pair share complete correlation regardless of physical distance between them. There are four different kinds of Bell states, expressed over a two-qubit Hilbert space with computational bases 00, 01, 10 and 11, and represented as:

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \quad (2.20)$$

$$|\Phi^-\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}} \quad (2.21)$$

$$|\Psi^+\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}} \quad (2.22)$$

$$|\Psi^-\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} \quad (2.23)$$

These Bell states are also used to form their own orthonormal two-qubit computational basis known as the Bell basis that is used in many applications making use of teleportation and entanglement such as in quantum cryptography.

The state $|\Phi^+\rangle$ is the canonical "e-bit" (entangled bit). It serves as a resource that may be consumed to perform quantum teleportation, allowing the transfer of an arbitrary qubit state $|\psi\rangle$ from one node to another using only classical communication and shared entanglement.

2.1.14 Clifford Operations

The Clifford group [30], denoted by \mathcal{C}_n , represents a fundamental class of quantum operations defined as the normalizer of the n -qubit Pauli group \mathcal{P}_n . Formally, an n -qubit unitary U is an element of the Clifford group if it maps

Pauli operators to Pauli operators under conjugation, such that $UPU^\dagger \in \mathcal{P}_n$ for all $P \in \mathcal{P}_n$ [41]. This is called the "Normalizer property" of the Pauli group.

The group is generated by the set of gates $\{H, S, \text{CNOT}\}$, where H is the Hadamard gate, S is the phase gate, and CNOT is the controlled-NOT gate. The computational significance of Clifford operations is established by the Gottesman-Knill theorem, which states that any quantum circuit consisting entirely of Clifford gates, Pauli-basis measurements, and Clifford group operations conditioned on classical bits which may be a result of earlier measurements, may be perfectly simulated on a classical computer in polynomial time [29]. Consequently, Clifford operations alone are insufficient for achieving a quantum speedup, despite their ability to generate highly entangled states.

2.1.15 Non-Clifford Operations

To overcome the limitations imposed by the Gottesman-Knill theorem and achieve universal quantum computation, the inclusion of non-Clifford operations is required. A non-Clifford operation is any unitary transformation U that does not satisfy the normalizer property of the Pauli group. The most common addition to the Clifford set is the T gate (or $\pi/8$ gate), which, when combined with the Clifford group, allows for the dense approximation of any unitary operation in the special unitary group $SU(2^n)$.

From a fault-tolerant architectural perspective, the distinction between these two classes is significant. While Clifford gates are often implemented transversally and with relatively low overhead in various quantum error-correcting codes, non-Clifford gates typically require sophisticated techniques such as magic state distillation. [4] [8] This makes non-Clifford operations the primary metric for quantifying the logical resource cost and execution latency of a quantum algorithm in a fault-tolerant regime.

2.1.16 Some Notable Protocols and Mechanisms

This subsection introduces several foundational protocols and mechanisms that enable distributed quantum operations across physically separated nodes. These mechanisms form the basis for quantum communication, remote computation,

and entanglement management in networked quantum systems, and they recur throughout later discussions of distributed architectures and performance trade-offs.

Entanglement Between Two Nodes

Entanglement between two remote quantum nodes is a fundamental resource for quantum communication and distributed quantum information processing. The creation of entanglement typically relies on photonic channels that mediate interactions between stationary qubits located at different nodes. While the physical realization depends on the underlying hardware platform, the logical objective is to establish a shared entangled state, often a Bell pair, between the two endpoints .

A common approach to entanglement generation involves the emission of photons from stationary qubits, which are then interfered and measured to herald the successful creation of entanglement. As a "candidate system" that is able to perform the trifecta of Bell state measurement operations (H gate, CNOT gate, measurement in Pauli Z Basis), stationary qubits represent quantum memories in repeater systems that are manipulated via optical interaction (laser pulses) to emit these photons. While classical memories store classical bits and information, quantum memories store initialized states ($|0\rangle$ & $|1\rangle$), superposition states, as well as entangled states, and the photons emitted from these memories are required to be collected in optical fibers. Depending on where the Bell state measurement is performed, entanglement generation protocols is categorized into different link configurations [32].

In **memory–memory (MM) links**, also called "**sender-receiver** architecture", both endpoints store qubits in quantum memories, and successful entanglement is heralded only after measurement is performed in one of the two devices and the outcomes are communicated classically. MM links are well suited for scenarios where quantum states must be stored for extended durations, but they are sensitive to memory decoherence since the photon generated by one of the two quantum memories must travel the entire length of the optical fiber to meet the other node holding the Bell state measurement (BSM) device.

In **memory–interfere–memory (MIM) links**, also called "**meet-in-the-**

middle architecture”, an intermediate photonic interface or measurement device (such as Bell state analyser (BSA), also called BSM) assists in entanglement generation by capturing and coupling entangled photons generated by quantum memories from two incoming sides and transferring entanglement between memories upon successful measurement. This configuration allows flexibility in network design and could reduce the burden on individual nodes by offloading measurement operations to shared infrastructure. The crucial point is that the arriving photons must be deemed as indistinguishable (exactly same arrival times and spectral properties), and any delays result in lowered measurement success probability.

In **memory–source–memory (MSM) links**, also called ”**mid-point-source** architecture” or MPM links [58], quantum memories at end node repeaters are connected by an entangled photon pair generation source in the middle (Spontaneous Parametric Down-Conversion (SPDC) [36] [33] device) where entangled photons are generated in the middle and sent to interfere locally with photons of quantum memories to establish links upon successful measurement. MSM links enable scalable connectivity by improving resilience to photon losses, at the cost of increased coordination and resource contention. Such link abstractions might be useful for modeling entanglement distribution in densely connected environments.

Teleportation (TP) Protocol

Quantum teleportation provides a mechanism for transferring an unknown quantum state between distant nodes using shared entanglement and classical communication, without physically transmitting the quantum system itself [58]. In the teleportation protocol, a sender performs a joint measurement on the local quantum state and one half of an entangled pair, then communicates the measurement outcomes to the receiver, who applies a corresponding correction operation.

Teleportation is used in two closely related contexts. In **teledata**, the quantum state itself is transferred from one node to another, allowing computation to continue at a remote location. In **telegate** protocols, teleportation is combined with local operations such that a logical quantum gate is effectively applied across nodes. Telegates are particularly important for implementing non-local two-qubit operations in distributed quantum circuits.

Both teledata and telegate protocols consume entanglement as a resource and

incur classical communication latency. As a result, their performance is tightly coupled to entanglement availability, fidelity, and scheduling, making teleportation a central primitive in distributed quantum systems.

Entanglement Swapping

Entanglement swapping is a protocol that enables the creation of entanglement between two nodes that have never interacted directly [58]. Consider three nodes arranged linearly, where the intermediate node shares entangled pairs with each endpoint. By performing a Bell state measurement on its two local qubits, the intermediate node projects the distant endpoints into an entangled state.

This process is also a type of Bell state measurement and effectively extends link-level entanglement across multiple network segments to form the basis of multi-hop quantum communication. Entanglement swapping does not require direct quantum transmission between the endpoints, but it relies on successful local operations and classical communication of measurement outcomes. Repeated application of entanglement swapping enables scalable entanglement distribution across larger networks, though imperfections accumulate and reduce fidelity with each hop.

Entanglement Distillation

Entanglement distillation, also referred to as entanglement purification, is a protocol used to improve the fidelity of shared entangled states by consuming multiple lower-fidelity pairs to probabilistically produce fewer high-fidelity pairs [32]. Distillation protocols typically involve local operations and classical communication between nodes and are essential for mitigating noise introduced during entanglement generation and swapping.

While distillation increases the quality of entanglement, it introduces additional resource overhead and latency. The trade-off between fidelity improvement and resource consumption plays a critical role in the design of quantum communication protocols and distributed computation strategies. In practice, distillation is often applied selectively, depending on application requirements and network conditions, to balance performance and resource efficiency.

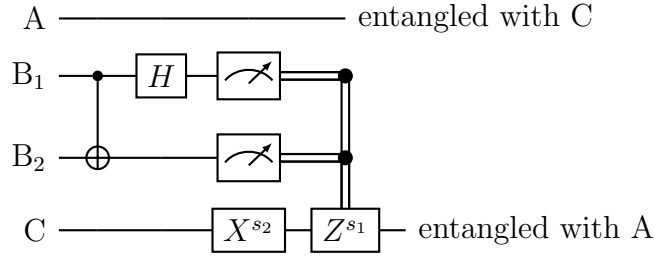


Figure 2.1: Entanglement swapping. If (A, B_1) and (B_2, C) are initially Bell pairs, a Bell-state measurement on (B_1, B_2) projects (A, C) into a Bell state up to Pauli corrections determined by classical outcomes (s_1, s_2) .

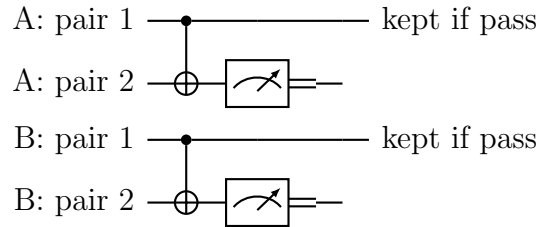


Figure 2.2: Generic entanglement distillation (purification) step. Two noisy Bell pairs are processed using local bilateral operations and measurements on the second pair. If classical outcomes satisfy an acceptance rule, the first pair is retained with improved fidelity; otherwise it is discarded.

2.1.17 DiVincenzo’s Criteria

The “Divincenzo criteria” is a set of conditions considered to be necessary to achieve quantum computation and communication. The general idea was given by David DiVincenzo, which was later converted into a criteria proposed by Richard Hughes for a quantum computing roadmap commissioned by ARDA [21]. Formalizing ideas generated from research and talks over two decades into the paper published in 2000 [23] [22], DiVincenzo describes his requirement criteria to implement quantum computation. Richard Hughes later referred to these as “promise criteria” instead in the QIST roadmaps proposed in 2001, where it was suggested that while achieving all of these criteria would get us closer to potentially forming a promising quantum computation system, however, it does not ensure that we will be able to implement it perfectly and that the making of such a system is also a highly complex task by itself.

These five (and two additional) criteria for quantum computation (also requirements for good quantum memories [32]) are:

1. Scalable physical system and well-defined qubits;
 2. Being able to initialize qubit state to a simple fiducial state such as $|000\dots\rangle$;
 3. Decoherence times much longer than gate operation times;
 4. A “universal” quantum gates set;
 5. Being able to perform measurement on qubits;
- and the “desirerata” for quantum communication are:
6. Being able to convert between stationary and flying qubits;
 7. Being able to transmit flying qubits to assigned locations.

2.2. Introduction to quantum communications

2.2.1 Classical and Quantum Networks: Fundamental Differences

Unlike classical communication networks, which operate by receiving, copying, and forwarding information packets, quantum networks are constrained by the fundamental laws of quantum mechanics. In particular, the no-cloning theorem prohibits the copying of unknown quantum states, and quantum measurements

irreversibly disturb the system being observed. As a result, quantum networks cannot transmit quantum information through amplification or store-and-forward mechanisms. Instead, their primary function is to distribute entanglement between distant nodes, which is later consumed to perform quantum teleportation using only local operations and classical communication.

A key distinction between classical and quantum communication lies in their respective noise models. In classical optical networks, signal attenuation could be compensated by amplification. In contrast, quantum signals cannot be amplified without destroying the encoded quantum information. For a photonic qubit transmitted through an optical fiber of length L , the probability of successful transmission typically scales as

$$P_{\text{trans}}(L) = e^{-\alpha L},$$

where α is the fiber attenuation coefficient. This exponential decay makes direct long-distance quantum communication impractical and motivates the use of entanglement-based protocols and intermediate nodes.

2.2.2 Quantum Networks

Entanglement Distribution and Teleportation

Quantum communication differs fundamentally from classical communication in that quantum information cannot, in general, be transmitted by copying and forwarding data packets. Instead, quantum networks rely on the distribution of entanglement between nodes, followed by quantum teleportation to transfer quantum states using only local operations and classical communication. Teleportation consumes shared entanglement and requires the transmission of classical measurement outcomes, ensuring that causality and relativistic constraints are preserved. As a result, the primary task of a quantum network is not the direct transmission of qubits, but rather the reliable and timely creation of entangled states between distant nodes.

Quantum networks are typically described in terms of *stationary* and *flying* qubits [32]. Stationary qubits are physical qubits that remain localized at network nodes and serves as quantum memories or computational resources, such

as trapped ions, neutral atoms, or solid-state spin systems. Flying qubits, most commonly implemented using photons, act as carriers of quantum information between nodes due to their low interaction with the environment and suitability for transmission through optical fiber or free space. In typical architectures, stationary qubits are entangled with flying qubits, which are then transmitted to other nodes or intermediate devices to establish entanglement across the network.

Bell State Measurements and Entanglement Swapping

A key hardware primitive in entanglement-based quantum networking is the *Bell State Measurement* (BSM), performed by a *Bell State Analyzer* (BSA). A BSM performs a joint measurement on two qubits in the Bell basis, projecting them onto one of the four maximally entangled Bell states. In optical implementations, BSAs are typically realized using linear optical elements such as beam splitters, phase shifters, and single-photon detectors, and are inherently probabilistic due to the limitations of linear optics. Despite this probabilistic nature, BSAs play a crucial role in enabling *entanglement swapping* to extend entanglement across multiple network hops.

From a networking perspective, the Quantum Internet is expected to evolve as a heterogeneous, multi-domain system rather than a single uniform network [61]. To address this complexity, architectural frameworks have been proposed that adapt principles from classical internetworking to the quantum domain. In particular, the Quantum Recursive Network Architecture (QRNA) [61] introduces recursive abstractions for naming, routing, and connection management to enable scalable quantum internetworking across administrative and technological boundaries. Complementing these architectural efforts, the Internet Engineering Task Force (IETF) has established the Quantum Internet Research Group (QIRG) to study architectural principles, use cases, and protocol requirements for quantum networks. QIRG provides a forum for developing a common vocabulary and conceptual framework, facilitating long-term interoperability between independently developed quantum networking technologies.

Entanglement swapping enables the extension of entanglement over multiple shorter links. If nodes A and B share an entangled pair, and nodes B and C share another, a Bell State Measurement performed at node B projects the qubits

at A and C into an entangled state, even though they have never interacted directly. Repeating this process across multiple segments allows entanglement to be established over distances that would otherwise be inaccessible due to photon loss.

Consider three nodes A , B , and C . Suppose nodes A and B share an entangled Bell state:

$$|\Phi^+\rangle_{AB} = \frac{1}{\sqrt{2}} (|00\rangle_{AB} + |11\rangle_{AB}) \quad (2.24)$$

and nodes B and C share another entangled Bell state:

$$|\Phi^+\rangle_{BC} = \frac{1}{\sqrt{2}} (|00\rangle_{BC} + |11\rangle_{BC}) \quad (2.25)$$

The joint state of the three nodes is then given by:

$$|\Psi\rangle_{ABC} = |\Phi^+\rangle_{AB} \otimes |\Phi^+\rangle_{BC} \quad (2.26)$$

If node B performs a Bell State Measurement (BSM) on its two local qubits, the joint state may be rewritten as:

$$|\Psi\rangle_{ABC} = \frac{1}{2} \sum_{i=0}^3 |\Phi_i\rangle_{AC} \otimes |\Phi_i\rangle_B \quad (2.27)$$

where $\{|\Phi_i\rangle\}$ denotes the four Bell states. As a result of the measurement outcome at node B , the qubits at nodes A and C are projected into a corresponding Bell state $|\Phi_i\rangle_{AC}$, up to a known Pauli correction determined by the classical measurement result.

This process establishes entanglement between nodes A and C without requiring a direct quantum channel between them, forming the fundamental building block of quantum repeaters and multi-hop quantum networks.

Quantum Repeaters

One of the primary technical challenges in realizing a large-scale Quantum Internet is the distribution of high-fidelity entanglement over long distances. Direct transmission of photonic qubits suffers from exponential attenuation in optical fiber and free-space channels, leading to prohibitive loss rates over distances beyond tens or hundreds of kilometers. Classical optical amplifiers cannot be used

to overcome this limitation, as they would violate the no-cloning theorem by effectively copying quantum states. Quantum repeaters have been proposed as a solution to this problem, enabling scalable long-distance entanglement distribution without direct transmission of quantum states over the entire path.

A quantum repeater network divides a long communication path into shorter segments, each of which establishes entanglement locally between neighboring nodes. These local entangled links are then connected using entanglement swapping operations performed at intermediate nodes. In entanglement swapping, a node that shares entangled pairs with two neighboring nodes performs a Bell State Measurement on its local qubits, resulting in the creation of an entangled state between the distant endpoints. By recursively applying this process, entanglement could be extended over arbitrarily long distances in principle. But imperfections in entanglement generation, storage, and swapping lead to a progressive degradation of entanglement fidelity as distance increases. To mitigate this effect, quantum repeater protocols incorporate *entanglement purification or distillation* where multiple low-fidelity entangled pairs are consumed to probabilistically produce a smaller number of higher-fidelity pairs [58] [32]. Purification protocols require two-way classical communication and introduce additional latency, but they are essential for maintaining usable entanglement in long-distance networks.

Quantum repeater architectures are often categorized into multiple generations based on how they handle loss and operation errors [32] [12]. First-generation repeaters rely on heralded entanglement generation and purification to combat both loss and operational errors. Second-generation designs reduce reliance on purification by improving memory coherence and local gate fidelity. 2G repeaters use quantum error correction and heralded entanglement generation to combat operational errors. Third-generation repeaters also incorporate quantum error correction to actively protect quantum information during transmission. 3G repeaters use QEC to combat both photon loss and operational errors. All three generations incorporate quantum memories. While higher-generation repeaters promise improved rates and scalability, they also require significantly more complex hardware and control systems.

Beyond long-distance communication, quantum repeater concepts are also rel-

evant in more localized settings, such as modular quantum computers and quantum data centers. Recent work on optical interconnects for modular quantum systems demonstrates that entanglement swapping via shared Bell State Analyzers may be used to efficiently interconnect multiple quantum computing modules within a confined environment [48]. In such architectures, repeater-like functionality enables scalable interconnection without the extreme distance-related losses characteristic of wide-area networks, highlighting the close conceptual relationship between quantum repeaters and short-range quantum interconnects.

Although our work will not utilize quantum repeaters directly, understanding repeater-based quantum networking is essential for contextualizing alternative approaches to scalable quantum computation. We shall compare repeater-based long-distance quantum networks with quantum data center architectures, motivating the choice of tightly coupled, short-distance interconnects for the performance analysis conducted in this study.

2.2.3 Quantum Network Scales and Architectures

Quantum Local-Area Networks and Quantum Data Centers

It is important to distinguish between different classes of quantum networks. Quantum networks broadly encompass any system that distributes entanglement between quantum nodes. Quantum local-area networks (qLANs), which operate over short distances and typically do not require repeaters, are well suited for modular quantum computing and quantum data centers. Metropolitan-scale and wide-area quantum networks, on the other hand, require quantum repeaters to overcome loss and maintain fidelity.

The Quantum Internet represents a further abstraction layer, concerned not with a single network, but with the internetworking of multiple heterogeneous quantum networks. Its primary focus is interoperability, routing, and coordination across administrative and technological boundaries rather than solely scaling quantum computation.

The Quantum Internet

The **Quantum Internet** is envisioned as a networked infrastructure that interconnects remote quantum devices using quantum channels in conjunction with classical communication links, enabling tasks that are fundamentally infeasible in classical networks alone [58] [57] [7]. Unlike the classical Internet, which is designed to reliably transmit and copy classical bits, a Quantum Internet must operate under the constraints imposed by quantum mechanics, including the no-cloning theorem, measurement-induced disturbance, and decoherence. These constraints fundamentally alter the design space of networking protocols and require new abstractions for communication, routing, and resource management.

A central motivation for the Quantum Internet is the need to scale quantum information processing beyond the limits of monolithic quantum devices. Current and near-term quantum hardware platforms are restricted in qubit count, coherence time, and physical footprint, making it difficult to realize large-scale fault-tolerant quantum computers within a single machine [7]. Distributed quantum computing addresses this limitation by interconnecting multiple quantum processing units (QPUs) such that quantum states and operations may be shared across spatially separated nodes. In this paradigm, a network of smaller quantum devices could collectively behave as a larger virtual quantum computer, with entanglement serving as the key enabling resource [58].

2.2.4 Classical Switched Network Topologies

Blocking Properties in Switched Network Design

Switched networks are widely used to interconnect large numbers of endpoints in data centers and high-performance systems. Rather than relying on a single monolithic switch, these networks are constructed from multiple stages of smaller switches to improve scalability. A fundamental property that differentiates such designs is whether the network could support simultaneous communication demands without internal contention. This property is commonly described using the notions of blocking, non-blocking, and rearrangeable non-blocking networks [44].

A **blocking network** is one in which a new connection request between two

endpoints may be denied even when the source and destination ports are free. This occurs because internal switching resources are already occupied by existing connections, and the network lacks sufficient alternative paths to route the new request. Blocking behavior arises when internal bandwidth is limited or path diversity is insufficient. While blocking networks could be cost-effective and perform well under average workloads, they provide no guarantee that all admissible traffic patterns may be supported simultaneously.

A **strictly non-blocking network** guarantees that any new connection between free input and output ports shall always be established without disturbing existing connections. In such a network, internal contention never prevents the addition of a valid connection. Achieving this property requires substantial internal bandwidth and a large number of interconnections, which makes strictly non-blocking designs difficult to scale economically. As a result, they are rarely implemented at large system sizes.

A **rearrangeable non-blocking network** occupies an intermediate position between these two extremes. In this class of networks, any set of input-to-output connections may be realized, but doing so may require rearranging existing connections. The network is therefore non-blocking in principle, but not necessarily non-blocking with respect to ongoing traffic. Multi-stage switching fabrics such as Clos networks fall into this category. They trade the ability to preserve all existing connections for a scalable structure that could support arbitrary traffic patterns through appropriate path selection and, when necessary, reconfiguration.

Blocking properties are essential for interpreting the behavior of modern data center topologies. Blocking designs favor simplicity and cost, strictly non-blocking designs favor predictability, and rearrangeable non-blocking designs provide a practical balance that enables scalability while retaining high utilization. These concepts form the classical foundation for later discussions of multi-stage network topologies and their applicability to distributed computing systems.

Clos networks architecture

In a time when the cost difference between commodity components that were general-purpose and off-the-shelf easy-access hardware, and specialized or customized hardware that were relatively scarce and harder to obtain, already had a

strong influence over the way communication networks were built, Charles Clos in 1953 was motivated by the telephone switch trends of the era to create a network topology for non-blocking telephone switches delivering higher bandwidth across multiple end-devices through the interconnection of a layer of smaller commodity switches [15] [51].

A three-stage Clos network consists of an ingress stage, a middle stage, and an egress stage of switches. By providing a sufficient number of switches in the middle stage, i.e. if the network is not oversubscribed [1], it is possible to create a "non-blocking" fabric, where 'theoretically', a connection path could always be established between any idle input port and any idle output port, regardless of other traffic in the network.

Fat-trees

The original idea of fat-tree network was first devised by Charles Leiserson in 1985 [38] for the purpose of interconnecting processors in parallel computers, where the idea was to address the bottleneck inherent in traditional tree structures where the single link to the root becomes congested. In a fat-tree, the branches of the tree become "fatter" and possess higher bandwidth while moving up through the hierarchy of the network toward the root in an effort to make sure that the bandwidth at any level of the tree is sufficient to handle the aggregated traffic from all the branches below it. For the purpose of modern data centers, fat-trees are constructed as multi-stage switching fabrics based on the Clos network. [55] [1]

A standard implementation of the fat-tree topology in modern data centers uses three layers of switches:

1. Edge/Access Layer: The lowest layer, equivalent to the access layer, where servers connect. These are typically ToR (top-of-rack) switches.
2. Aggregation Layer: The middle layer, which interconnects groups of edge switches.
3. Core Layer: The top layer, which provides global connectivity between all aggregation switches.

The main goal of this design philosophy is making sure that the total bandwidth of the "uplinks" from any switch (or group of switches) is equal to the total

bandwidth of the "downlinks" to the servers supported by said switch(es). Upon achieving this 1:1 "oversubscription" ratio [1], the network provides full bisection bandwidth and is considered non-blocking.

The two-tier spine-leaf architecture

In modern networking [55] [51], the terms "fat-tree" and "spine-leaf" are used interchangeably. The two-tier spine-leaf architecture is simply a **"folded" three-stage Clos network** or a **"two-layer" fat tree**. By taking a three-stage Clos network (with ingress, middle, and egress stages) and "folding" it in the middle such that the ingress and egress stages merge into a single layer, we create a visual realization of this folded model with the "leaf switches" performing the function of both the ingress and egress stages by connecting to the end nodes, acting as Top-of-Rack (ToR) switches, and the "spine switches" performing the function of the folded middle stage by providing an interconnected, unified, network fabric. This folded architecture is representative of the two-tier spine-leaf design which inherits all the desirable properties of the Clos network, including the potential for non-blocking performance as well as multiple parallel paths.

The spine-leaf architecture was constructed to overcome the limitations of the traditional three-tier (core, aggregation, access) model where it was ill-suited for high volume internal and server-to-server traffic prevalent in virtualized and cloud environments. With a spine-leaf design, if one or more spine-switches in the middle fail for some reason, it will not disrupt the flow of traffic across the network, providing advantages such as the mitigation of risk over multiple devices instead of the two end nodes of a connection.

2.2.5 Distributed Quantum Computing

Distributed Quantum Computing (DQC) studies how multiple quantum processing units (QPUs) could be interconnected and orchestrated to execute computations that exceed the scale or connectivity limits of a single device [3, 59]. A primary motivation for DQC is that scaling monolithic quantum processors is constrained not only by fundamental noise and decoherence, but also by practical engineering limitations such as wiring density, control electronics, and increasing

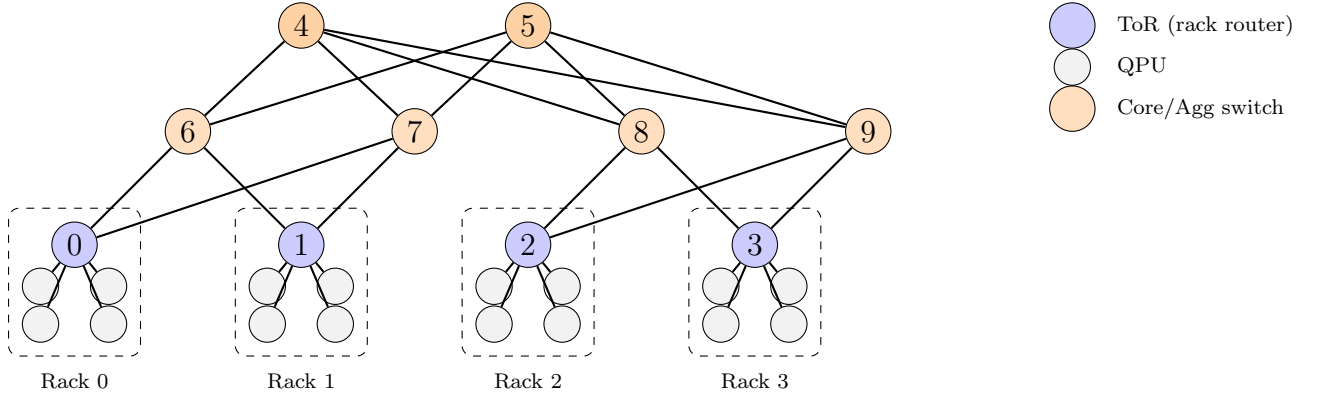


Figure 2.3: Clos topology for $n = 4$ (from `gen_clos_conn`). Each rack contains one ToR switch and four QPUs.

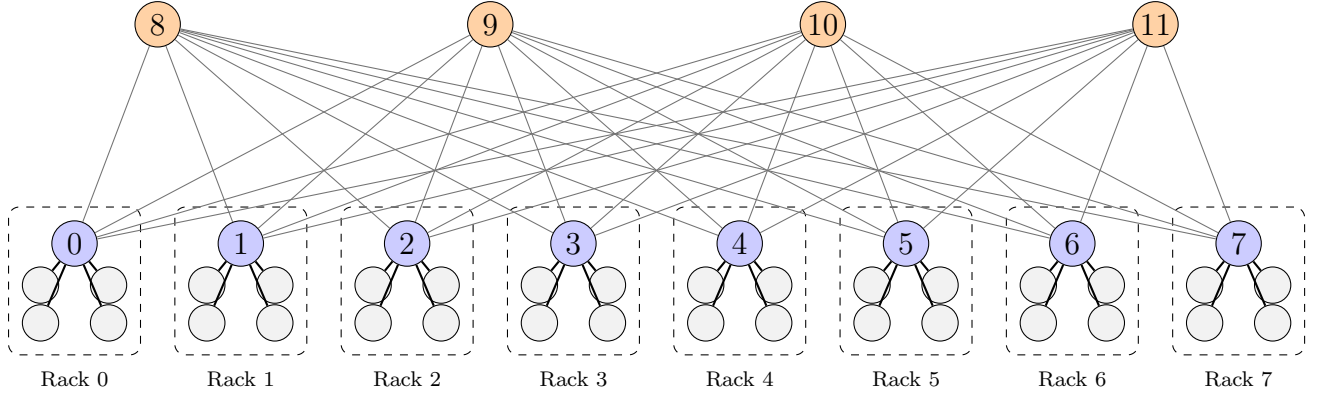


Figure 2.4: Spine-leaf topology (L2 fat-tree) from `gen_L2_fat_tree`: 4 spines (8–11) fully connected to 8 rack ToRs (0–7). Each rack contains one ToR and four QPUs.

crosstalk as systems grow [9]. As a result, there is substantial interest in modular and networked approaches in which several smaller processors are linked using quantum and classical communication so that they could collectively behave as a larger computational resource.

The core operational distinction between DQC and conventional parallel computing is that quantum information cannot generally be duplicated and broadcast across nodes, and therefore distributed execution must rely on entanglement-assisted primitives rather than copying intermediate state [13]. Cirac *et al.* analyzed distributed quantum computation over noisy channels and introduced a cost model that explicitly accounts for precomputation and communication re-

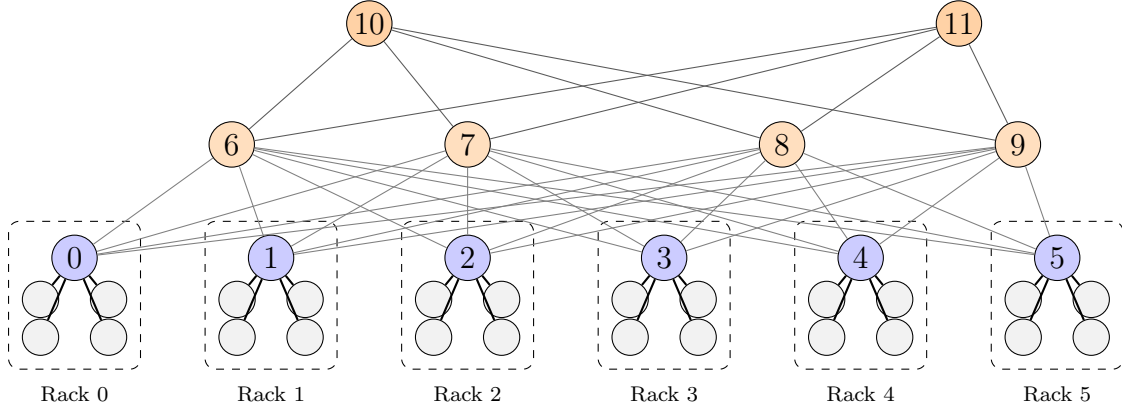


Figure 2.5: Fat-tree topology (L3 fat-tree) from `gen.L3_fat_tree`: 2 upper routers (10–11) fully connected to 4 mid routers (6–9), which are fully connected to 6 rack ToRs (0–5). Each rack contains one ToR and four QPUs.

sources needed to establish entanglement among processors, including the role of purification under noise. This early framing remains relevant because modern DQC systems still face the same underlying constraints. Non-local correlations are valuable for computation, but establishing and maintaining those correlations across nodes could dominate runtime and resource cost.

A central requirement in DQC is the ability to implement entangling operations between qubits residing on different QPUs, since distributed execution typically partitions a circuit such that some two-qubit gates cross QPU boundaries [9] [46]. In QDC-oriented DQC discussions, Campbell *et al.* compare teleportation-based communication (TP-comm) and cat-state-based communication (cat-comm), and analyze how error mechanisms and latency costs influence end-to-end circuit fidelity for these remote gates. Additional noise from entanglement distribution and time overhead is introduced, and counting only the number of remote gates or the number of inter-QPU entangled links may be insufficient to predict output fidelity due to error propagation effects. From a systems perspective, algorithm execution time might be dominated by the latency and contention associated with generating the entanglement needed by remote operations. This concern motivates work such as DQC-QR which explicitly treats the problem of distributing and routing quantum circuits to minimize execution time under network resource and decoherence constraints and considers multiple proto-

cols for remote gates, including telegate-style teleportation and cat-entanglement methods [54]. Together, these results position remote-gate protocol selection and scheduling as first-order design choices for practical DQC systems.

Layered models for DQC systems

Recent surveys organize DQC as a full-stack system with layers spanning hardware, networking, and software, in order to clarify dependencies between physical capabilities and compiler-level decisions. One representative layered model partitions the system into a physical layer that enables interconnection between QPUs, a network layer that coordinates multi-QPU connectivity and entanglement distribution, a development layer comprising partitioning, compilation, optimization, and qubit mapping, and an application layer containing distributed algorithms [3]. This perspective emphasizes that distributed compilation is tightly coupled to the underlying interconnect because the achievable entanglement rate, the availability of communication qubits, and link-level reliability directly shape which circuit partitions are practical and how frequently remote operations are scheduled.

Van Meter and Devitt similarly motivate architectural investigation by noting that scalability concerns have driven proposals for distributed-memory multicomputer architectures, and that experimental progress on building blocks has increased urgency for understanding how algorithms interact with architecture [59]. In these views, DQC is not a purely networking problem and not purely a compilation problem, but a co-design problem in which hardware connectivity, control latency, and fault tolerance considerations must be matched to software orchestration and algorithmic structure.

2.2.6 Quantum Data Centers

Quantum Data Centers (QDCs) are infrastructures that interconnect multiple QPUs using photonic networking components, proposed to enable scalable distributed quantum computing in a controlled environment [51] [9]. QDCs are a response to the gap between current QPU sizes and the scale required for large quantum computations, addressing challenges in qubit transfer rate and fidelity, network latency, and the probabilistic nature of entanglement generation. Distribut-

ing computation across multiple smaller QPUs would be able to reduce certain intra-processor error mechanisms such as crosstalk associated with scaling within a single control system, while taking on the challenge of new error sources from inter-QPU entanglement distribution and communication latency. A defining feature of QDC operation is that distributed circuit execution relies on entanglement-assisted primitives for non-local operations, which introduces quantum-specific performance constraints absent in classical data centers. Stochastic entanglement generation under optical loss, coherence-limited retry windows, and contention for Bell-state measurement resources have been observed as key factors that interact nontrivially with topology and scheduling policy, which motivate evaluating QDC designs not only by path length and path diversity, but also by where and how shared quantum resources such as BSM modules are provisioned across the network fabric [46].

Why architectural classification matters

As DQC proposals move from conceptual models to scalable systems, architectural classification becomes useful for comparing designs and for understanding where performance bottlenecks originate [3, 59]. In particular, the structure of the interconnect determines whether entanglement is created primarily by direct connectivity between endpoints, or whether intermediate nodes participate in entanglement routing and swapping which changes both fidelity scaling and resource contention patterns. These distinctions become especially important in QDC settings, where co-location enables short physical links but does not remove the need for careful orchestration of probabilistic entanglement generation and shared measurement resources [51] [46].

Network architectures in QDCs

Multiple QDC network topologies have been proposed by adapting classical data center ideas to quantum networking constraints, where some architectures leverage a dynamic circuit-switched quantum network with shared resources such as Bell-state measurement devices and entanglement sources, aiming to enable on-demand connectivity while reducing reliance on expensive quantum hardware at every endpoint [51]. This distinction illustrates how entanglement routing differs

when QPUs act as intermediate repeaters in server-centric designs versus when optical switching fabrics route photons and swapping is performed at BSM-capable switches in switch-centric designs [46]. Within this design space, QDC topologies are categorized into two broad classes, namely switch-centric and server-centric architectures, following classical data center networking paradigms.

Server-centric architectures

In server-centric architectures, the network does not necessarily provide direct optical connectivity between every QPU pair. Instead, QPUs may be connected in a modular graph where intermediate QPUs participate in establishing end-to-end entanglement by acting as entanglement routers and performing entanglement swapping [46]. This design choice places additional operational responsibility on QPUs and introduces multi-hop effects in which latency and noise accumulate as entanglement is routed through intermediate nodes. In QDC benchmarking contexts, server-centric designs are often represented by architectures such as **BCube**, in which intermediate QPUs contribute to forming repeater-like chains for entanglement distribution, making performance sensitive to coherence windows and scheduling policy.

Switch-centric architectures

In switch-centric architectures, QPUs act primarily as endpoints, and the switching fabric provides optical paths that connect communicating QPUs through reconfigurable optical switches [51]. In this setting, swapping and measurement operations may be consolidated into BSM-capable switching nodes and shared photonic resources, and the network aims to provide high connectivity without requiring QPUs themselves to serve as repeaters. Switch-centric QDC topologies inspired by classical data center designs such as Clos and Fat-trees are representative examples of structured fabrics intended to scale while supporting on-demand entanglement distribution (See Figure 2.6). Pouryousef *et al.* benchmark representative architectures including **QFly**, **Clos**, and **Fat-tree**, with server-centric architectures like **BCube**, and analyze how topology and shared BSM resource provisioning affect distributed circuit execution latency and contention [46].

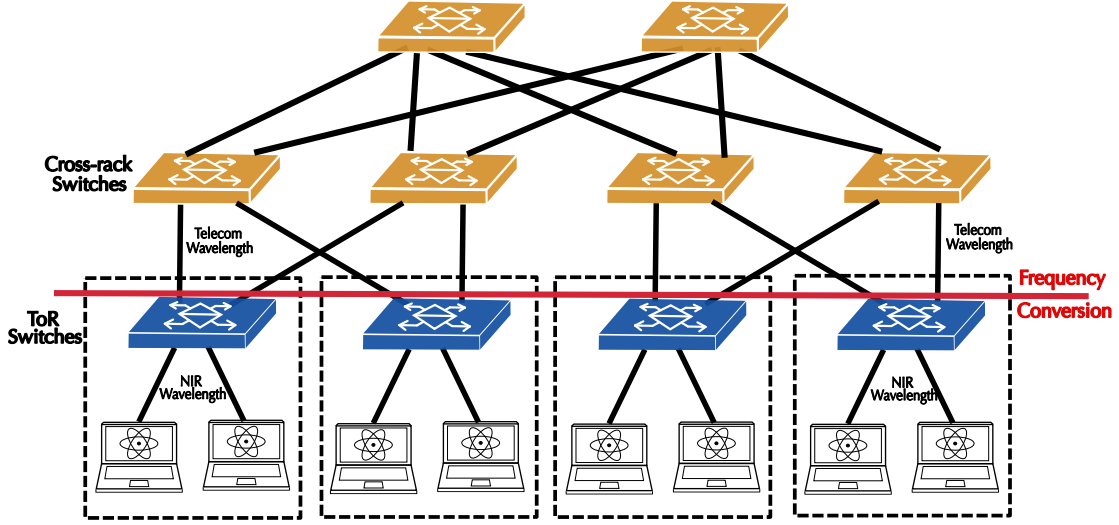


Figure 2.6: A quantum data center (QDC) with switch-linked Quantum Processing Units (QPUs).

Switched networks for scalable QDCs

A recurring theme in QDC research is that a switched-network approach offers a viable path to scalability because it separates the roles of computation and interconnect, and enables network-wide resource pooling for entanglement generation and measurement. Shapourian *et al.* propose circuit-switched quantum networks that use optical switches and shared BSM devices to enable on-demand connectivity across the fabric while reducing the need for fully provisioned quantum networking hardware at every QPU-to-QPU link [51]. Pouryousef *et al.* show that performance are tightly coupled to how BSM resources are provisioned, and that architectural properties such as path length, path diversity, and the number of switching elements interact with physical-layer parameters such as optical insertion loss and reconfiguration delay [46].

Classical switched-network Clos and Fat-tree are designed to provide scalable, high-bisection-bandwidth interconnects via multistage switching, and these same structural properties translate into opportunities for parallel entanglement generation and reduced contention when adapted to quantum settings. However, the cost model differs from classical networking because entanglement generation is stochastic and consumes quantum resources, therefore benefits of path diversity

and resource pooling must be evaluated together with coherence constraints and shared measurement bottlenecks. Improvements in individual hardware parameters do not translate uniformly into system-level gains, and that topology-aware benchmarking is necessary because topology, scheduling, and physical-layer limitations interact in nontrivial ways [46]. These observations motivate focusing on switched fabrics as an architectural axis for systems such as SwitchQNet [68], where the interconnect structure and orchestration policy are central to performance analysis in distributed workloads.

2.2.7 Quantum Error Correction in DQC and QDCs

Due to physical constraints of noise and errors, many physical qubits are required to realize a logical qubit. **Quantum error correction (QEC)** addresses the fragility of quantum information by encoding logical qubits into entangled states of multiple physical qubits such that errors may be detected and corrected without directly measuring the encoded quantum information. The fragility of coherent quantum states is a central obstacle to building large-scale quantum computers, and that QEC provides active techniques to mitigate this problem and enable fault-tolerant computation [20]. From an architectural standpoint, experimental efforts have demonstrated building blocks for scalable designs and that fault tolerance has motivated a strong focus on practical QEC implementation paths with effort coalescing around topological coding models [59]. Fowler *et al.* present surface codes as a prominent topological-code approach aimed at practical large-scale quantum computation, framing them as a key candidate for fault-tolerant systems [28].

In DQC and QDC settings, QEC is relevant because distributed execution introduces additional error sources and timing overheads associated with non-local operations and entanglement distribution. Campbell *et al.* discuss how inter-QPU entanglement distribution and latency-driven decoherence contribute noise that is absent in purely local computation, implying that scalable distributed systems must account for how such errors propagate through distributed circuits [9]. Additionally, Pouryousef *et al.* further highlight that coherence-limited entanglement retry windows and contention for shared BSM resources shape end-to-end execution performance, tying network behavior directly to qubit lifetimes and, conse-

quently, to the motivation for fault-tolerant protection mechanisms [46]. QEC is thus an integral component for future discussions of how distributed architectures might evolve from near-term, error-constrained operation toward fault-tolerant regimes, and how network structure and orchestration could influence the feasibility and overheads of integrating protection mechanisms.

Chapter 3

Motivations and Problem Statement

3.1. The SwitchQNet compiler

SwitchQNet [68] is a compiler framework intended to study distributed quantum computing in a quantum data center setting, where multiple QPUs are arranged across racks and connected through a switch network. The framework is motivated by the observation that inter-QPU communication is substantially slower and more error prone than local computation, and that cross rack communication and switch reconfiguration dominates end-to-end execution latency even when cross-rack EPR pairs constitute a minority of the total entanglement demand. SwitchQNet addresses this by performing co-optimization across the program and network layer, using program structure to guide when EPR pairs should be generated and stored, and using network state to determine which EPR requests could be served without causing resource conflicts. EPR pair preparation may be decoupled from the moment of use because EPR pairs do not carry program data, and that entanglement swapping constructs an end-to-end EPR pair from smaller segments when bandwidth and buffer conditions permit. These two features provide an optimization routine that is not available in purely on-demand entanglement generation, but they also introduce the risks of buffer congestion and deadlock that must be managed at compile time.

Terminology: SwitchQNet uses a small set of concepts that appear repeat-

edly in both the paper and the code. A QPU is a compute node that holds computation qubits and a limited number of communication qubits. Racks are groups of QPUs that share rack level resources. EPR pairs are entangled Bell pairs used to realize non-local operations between QPUs. In-rack EPR pairs refer to entanglement generated within the same rack, while cross-rack EPR pairs refer to entanglement generated across racks and mediated by the switch fabric. The scheduler represents the compiler component that produces a time ordered schedule of entanglement generation events while enforcing resource constraints. In the code, these constraints include limited communication qubits per QPU, limited cache or buffer capacity, and limited concurrent fabric connections and BSM resources that are modeled on network nodes and edges. SwitchQNet also distinguishes between teleportation based and cat state based communication protocols at the level of EPR requests and uses them to motivate how different remote gate patterns translate into entanglement demand. Each EPR request node carries a flag indicating whether it corresponds to a teleport-style EPR pair or a cat-style EPR pair.

Architectural model and network abstraction: SwitchQNet targets a quantum data center architecture in which QPUs are connected by a switch network. Such data center scale settings differ from long range repeater networks, and that the dominant bottlenecks arise from cross rack entanglement generation and switch reconfiguration overhead. The code represents the physical interconnect as a graph. Nodes include routers and QPUs, and edges represent available optical channels with a finite number of parallel connections. The scheduler treats these node and edge resources explicitly through per resource occupancy state. A useful model for documentation (Figure 3.1) is that the topology generator builds a router fabric, after which QPU nodes are attached to the rack routers. The scheduler then operates on the resulting graph that includes both routers and QPUs. This is consistent with the way the scheduler identifies QPU nodes by selecting graph nodes that are marked as not being routers.

Communication Protocols: The paper motivates inter-QPU computation by describing how remote gates may be realized using shared entanglement, and distinguishes between teleportation cat-state protocols as representative mechanisms for implementing non local operations. In SwitchQNet’s compiler model,

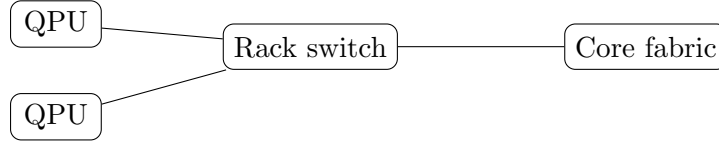


Figure 3.1: Conceptual quantum data center abstraction used by SwitchQNet. QPUs connect to rack level switching, while cross rack connectivity is mediated by a higher level fabric.

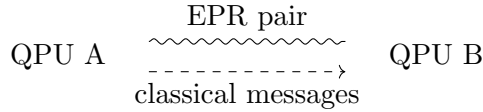


Figure 3.2: EPR Pair Abstraction used in SwitchQNet. Inter QPU interactions are modeled through EPR consumption plus classical coordination.

the details of gate level operations are abstracted into the requirement that an EPR pair must be available between two QPUs to complete the corresponding remote interaction. This abstraction is essential because the compiler’s objective is to schedule EPR generation and consumption under network constraints, rather than to simulate the full quantum circuit state evolution. Figure 3.2 captures the key idea that a non-local interaction consumes entanglement and triggers classical side information exchange, without committing to a specific hardware gate sequence.

Preprocessing: SwitchQNet’s framework design begins by converting a program’s communication demands into a representation suitable for scheduling. The paper describes a preprocessing step that takes a list of required EPR pairs produced by an upstream compilation pass and transforms it into a directed acyclic graph by adding dependencies between EPR requests whose QPUs overlap. It also cautions that these dependencies are an approximation because true schedulability depends on communication qubit multiplicity and network bandwidth contention, but it argues that the DAG still provides useful guidance for look ahead scheduling. The code follows the same conceptual structure. A routing or communication demand list is first constructed at the program layer, and then a DAG is built where each node stores the endpoint communication pair and a protocol tag. The dependency edges are introduced so that if a QPU appears in consecutive demand entries, the later request depends on the earlier one. The code comments

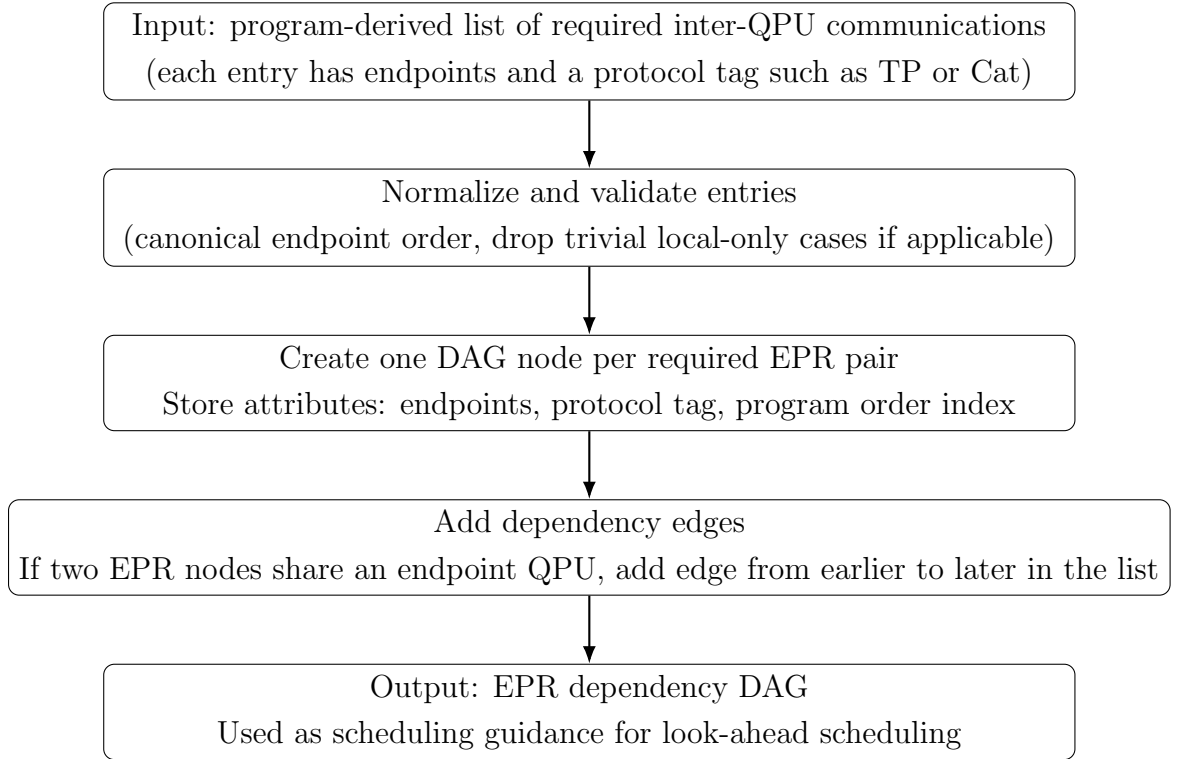


Figure 3.3: Flowchart of the preprocessing stage. A program-level EPR request list is converted into a dependency DAG that guides scheduling.

note that this is not fully accurate and that qubit level dependencies would be more precise, which is consistent with the paper’s characterization of the DAG as guidance rather than ground truth.

Scheduling model and constraints: SwitchQNet schedules entanglement generation over discrete time, which the paper refers to as time slices. At each time slice, the scheduler decides which EPR requests may be served based on a combination of hard resource constraints and soft buffer management constraints. The paper states three hard conditions involving communication qubit availability at both endpoints, availability of BSM resources at rack switches, and availability of an optical channel between endpoints, and it introduces a soft condition that biases scheduling toward front layer requests to mitigate buffer congestion. The code instantiates the scheduler with explicit parameters controlling communication qubit count, cache capacity, reserve capacity, and time costs associated with

in-rack EPR pair generation, switch reconfiguration, and cross-rack EPR pair generation. It returns a schedule along with counts of cross-rack and in-rack pairs, distillation related in-rack pairs, cache wait times, and retry overhead. This aligns with the evaluation metrics introduced in the paper’s experiment setup, which emphasizes latency, EPR overhead, buffer wait time, and retry overhead.

Collective In-rack generation and look-ahead scheduling: The first major optimization is the collective generation of in-rack EPR pairs, motivated by the fact that switch reconfiguration cost dominates the incremental cost of generating several in-rack EPR pairs once a channel has been configured. The compiler uses look ahead into near future program demands to identify repeated in rack needs between the same QPUs and schedules them consecutively while the channel is available. This behavior is controlled by a parameter that enables joining or combining in rack generations, and by a look ahead depth parameter that controls how many DAG layers are examined when selecting schedulable requests. The scheduler’s algorithmic structure follows the two-round approach at each time slice, where it first attempts to schedule regular EPR pairs and then attempts splitting if bandwidth remains.

Cross-rack EPR pair splitting and Entanglement Swapping: The second major optimization is the parallelization of cross-rack EPR generation by splitting congested cross-rack requests into alternative cross rack requests plus additional in-rack requests. The motivation is that if a particular source or destination QPU is busy, the compiler routes cross-rack entanglement through another QPU in the same rack and later complete the original endpoint pair using an additional in-rack EPR and an entanglement swapping operation on QPUs. The scheduler contains an explicit split step that is performed after attempting standard scheduling, and it maintains additional bookkeeping to track whether a scheduled pair corresponds to an original request or one produced by splitting. The schedule record includes both the origin pair and the mapped pair, allowing later analysis to distinguish split induced operations.

Post-split distillation and fidelity overhead management: The paper [68] emphasizes that splitting improves latency by introducing additional in-rack EPR pairs, but that this increases fidelity overhead because EPR generation is more error prone than local gates. To mitigate this, SwitchQNet prepares multiple

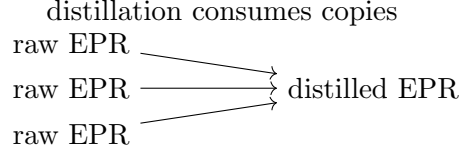


Figure 3.4: Abstraction of post-split distillation. Multiple lower fidelity in-rack EPR pairs are consumed to produce a higher fidelity pair that reduces fidelity overhead introduced by splitting.

copies of post split in-rack EPR pairs and applies entanglement distillation. The paper also discusses how the number of EPR pairs consumed in distillation affects both the resulting fidelity and the latency impact, noting that sacrificed pairs are in rack and collectively generated, which limits the latency penalty. In the code, the distillation related overhead is exposed through parameters that control how many in-rack pairs are used to form a distilled pair during split handling, and the scheduler returns a count of distilled in rack pairs attributable to splitting, enabling the EPR overhead metric as part of the evaluation.

Deadlock, buffer congestion, and retry based recovery: Flexible scheduling introduces failure modes that do not arise in strict on-demand generation. Deadlock may occur when multiple splits reserve buffer resources in a cyclic dependency, and buffer congestion may occur in scenarios where teleportation changes buffer availability asymmetrically across endpoints. To ensure progress, the framework includes an automatic retry mechanism that restores a previous state and retries scheduling with a more conservative strategy, ultimately falling back to a strict on demand strategy in the worst case. The code implements this concept by maintaining snapshots of scheduler state and restoring from a prior snapshot when scheduling cannot complete, while tracking retry counts and the total number of time steps explored. The scheduler outputs retry statistics that correspond directly to the retry overhead metric in the paper results.

Full compilation process and codebase execution flow: For documentation and reproducibility, it is useful to describe SwitchQNet’s execution as a deterministic pipeline that transforms program level communication demands into a network constrained schedule and a set of metrics. The codebase supports running comprehensive experiments where a configuration specifies a topology type, a program type, and a scheduler variant. The primary entry point for experiments

constructs a network graph, creates a rack to QPU mapping, generates a routing or communication demand list for the chosen benchmark program, converts this demand list into a DAG of EPR requests, and then runs a scheduler that produces a schedule and summary statistics. The conversion from a routing result list into an EPR dependency DAG follows a consistent format. Each demand entry is interpreted as an interaction between two QPUs, labeled as either teleport (TP) or cat type, and stored as a node attribute in the DAG. Dependencies are introduced to prevent consecutive operations involving the same QPU from being scheduled as if they were independent. Once the DAG is constructed, the scheduler initializes its internal representation of QPU resources by scanning the network graph for nodes that represent QPUs and assigning each such node communication qubit and cache parameters. During scheduling, the implementation maintains an event list representing in-flight EPR generation operations. At each time step, it attempts to map schedulable DAG nodes into concrete network operations, applying look-ahead if enabled, applying joining of in-rack operations if enabled, and applying split operations if enabled. The scheduler records each completed event into a schedule log that includes whether the event corresponds to TP or cat type, whether it is original or produced by splitting, the origin and mapped endpoint pairs, and the start and end times. The final outputs include total schedule duration in standard time units, counts of cross-rack and in-rack pairs, the distillation related in-rack count attributable to splitting, accumulated cache wait times for cross-rack and in-rack EPRs, and retry statistics. The pseudocode in Figure 3.5 captures this execution flow at a level suitable for thesis exposition and future documentation.

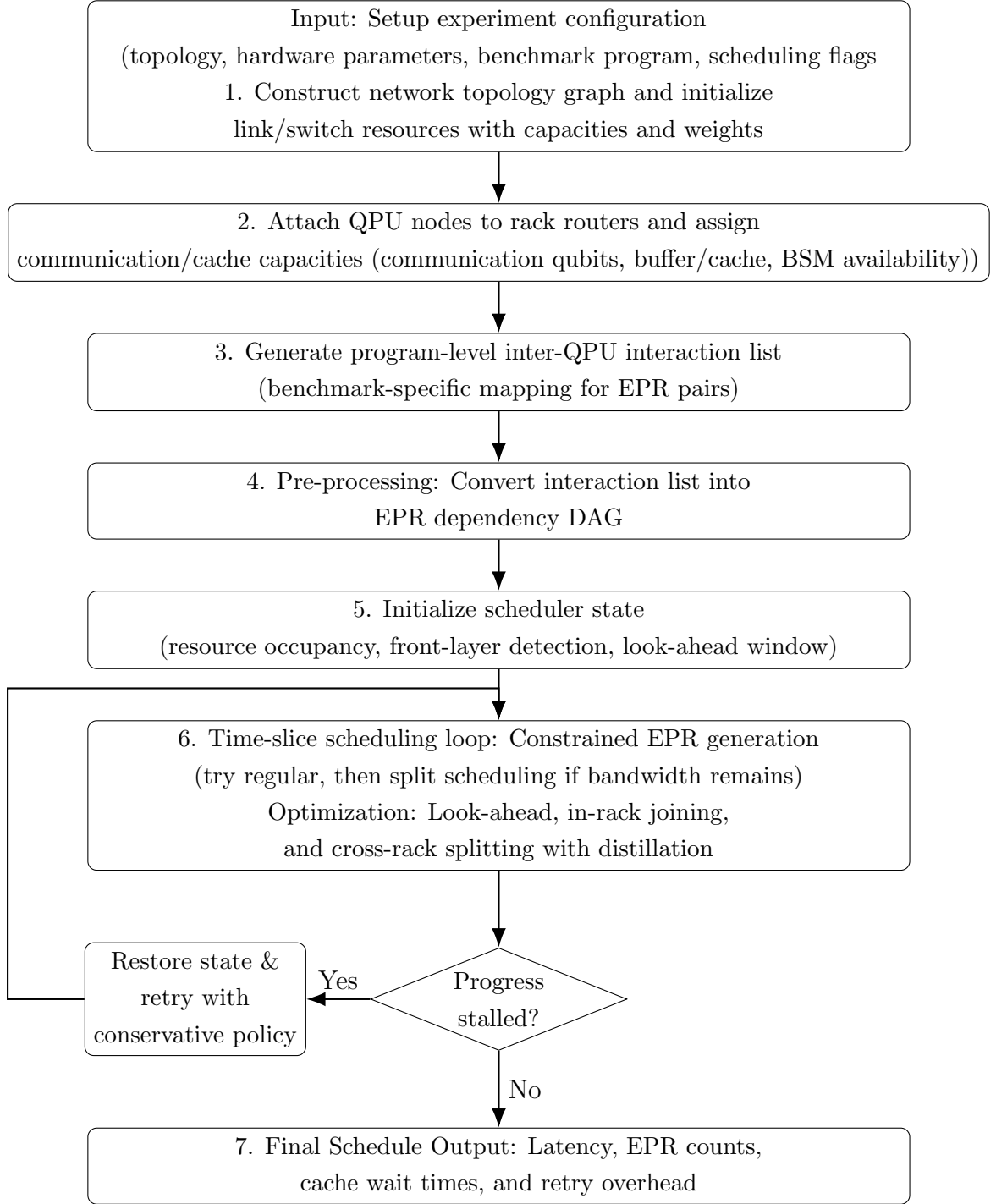


Figure 3.5: SwitchQNet end-to-end compilation and scheduling flow. The pipeline transforms a high-level benchmark into a time-sliced schedule through dependency analysis, optimized EPR routing, and a fail-safe recovery mechanism.

3.2. The Quantum Approximate Optimization Algorithm

First proposed by Farhi, Goldstone, and Gutmann in 2014 [25] and inspired by quantum annealing, the Quantum Approximate Optimization Algorithm, abbreviated as QAOA, is a quantum gate model-based algorithm for solving combinatorial optimization problems by using the variational principle to find parameters that maximize or minimize the defined objective function.

Classical problems that are able to be formulated into a binary optimization problem are capable of being solved using QAOA. The goal in such an instance would be to take a classical objective function $C(x)$ with “ x ” being a vector of “ n ” binary variables, and either minimize or maximize this function to reach the desired solution configuration. When using QAOA, this cost function $C(x)$ is mapped to a cost Hamiltonian H_c on a quantum system with “ n ” qubits, such that the ground state is corresponding to the most optimal solution of the mapped classical problem,

$$\begin{aligned} \text{minimize } C(x) \quad \text{subject to } x \in \{0, 1\}^n \\ H_C = \sum_{i,j} C_{ij} |i\rangle \langle j| \end{aligned} \quad (3.1)$$

where C_{ij} would represent the coefficients of the cost function, and $|i\rangle$ and $|j\rangle$ are the basis states of the Hilbert space.

A common way to select H_c for problems such as the Max-Cut problem or the Graph Partitioning problem is the following:

$$H_C = \sum_{\{i,j\} \in E} w_{ij} \left(\frac{1 - Z_i Z_j}{2} \right) \quad (3.2)$$

where $w_{i,j}$ is the weight associated with each edge within $\{i,j\}$ in the graph, and Z_i & Z_j are the Pauli-Z operators acting on qubits i & j respectively.

QAOA has a relatively simple structure requiring fewer qubits and shallower circuit depth compared to other quantum algorithms for similar combinatorial optimization problems, therefore making it robust and a highly appealing candidate for many types of such problems with a large solution subspace.

Regarding the design of the algorithm, we first start off with all qubits initially in a state such as $|0\rangle$, entangled with each other as an initial state which is the superposition of all possible states in the system, which for simplicity may be a set of Hadamard gates H over all the qubits in that circuit.

Next, the two main components of this method are introduced. The first one is the phase operator “gamma”, which is based on the cost hamiltonian H_c of the encoded problem and has the role of moving the quantum state and the initialized parameters of the problem to optimized and improved solution configurations. The second component is the mixer operator “beta” based on the mixer Hamiltonian H_m which is responsible to apply rotations to each qubit to make sure that the system does not stay stuck in the same state and allows it to explore solutions from different configurations.

From here, the entangled system is rotated by the phase operator based on the constraints of H_c and then mixed to check another configuration by the mixer operator. Depending on the type of problem, this process of phase rotating and mixing may be repeated over multiple alternating layers “p” which is defined as our depth hyperparameter. When all brought together, the QAOA may be expressed via the Trotterization process [56] as follows:

$$|\psi(\beta, \gamma)\rangle = e^{-i\gamma_p H_C} e^{-i\beta_p H_M} \dots e^{-i\gamma_1 H_C} e^{-i\beta_1 H_M} |\psi_{H_M}\rangle \quad (3.3)$$

Once the state has been prepared, we make a measurement in the computational basis and obtain the optimized bitstring “x” as output. We represent the expectation value of the cost function as follows:

$$F(\beta, \gamma) = \langle \psi(\beta, \gamma) | H_C | \psi(\beta, \gamma) \rangle \quad (3.4)$$

Upon sufficient repetitions of the state preparation process, the system should converge toward its absolute (global) maxima or minima. If p tends to infinity, it has been shown that QAOA will always give the most optimal result (adiabatic evolution). For a small value of p , we could still perform QAOA to give a fairly good approximation on NISQ devices. We eventually use this expectation value F as the objective function for our selected classical optimizer.

QAOA instances are naturally parameterized by a problem graph, since for many combinatorial optimization problems the cost Hamiltonian may be written

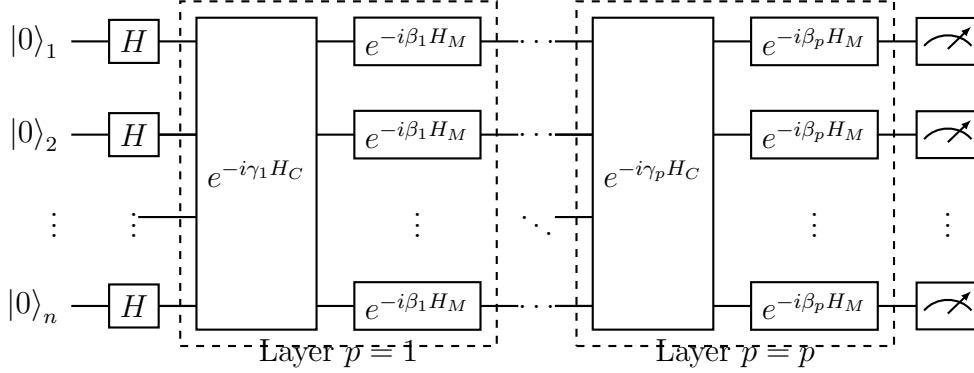


Figure 3.6: A QAOA circuit. The phase operator H_C acts across the register to encode problem constraints, while the mixer H_M facilitates state transitions. Dashed boxes denote the alternating cost and mixer layers repeated p times.

as a sum of local terms associated with edges (and sometimes vertices) of a graph. In the MaxCut formulation, for example, each edge $(i, j) \in E$ contributes a two-qubit interaction term (often expressed using $Z_i Z_j$), so the entangling structure of the QAOA cost unitary directly mirrors the adjacency structure of the input graph. This graph-induced locality is one reason QAOA is frequently studied as a representative near-term workload, because changes in graph degree and edge density translate directly into changes in the number and pattern of two-qubit interactions that must be implemented. Farhi, Goldstone, and Gutmann introduced QAOA and analyzed its behavior on MaxCut instances on regular graphs, making explicit how the graph family influences approximation quality and circuit structure.

Different graph families are commonly used to probe QAOA performance and systems-level behavior. k -regular graphs provide a controlled setting where every vertex has the same degree, keeping per-qubit interaction count uniform and making scaling trends easier to interpret, while still generating nontrivial entanglement patterns. Random graphs, particularly Erdős–Rényi graphs $G(n, p)$, are also widely used as generic benchmarks because they interpolate smoothly between sparse and dense instances as p varies, thereby changing both the number of cost terms and the connectivity-induced correlation structure.

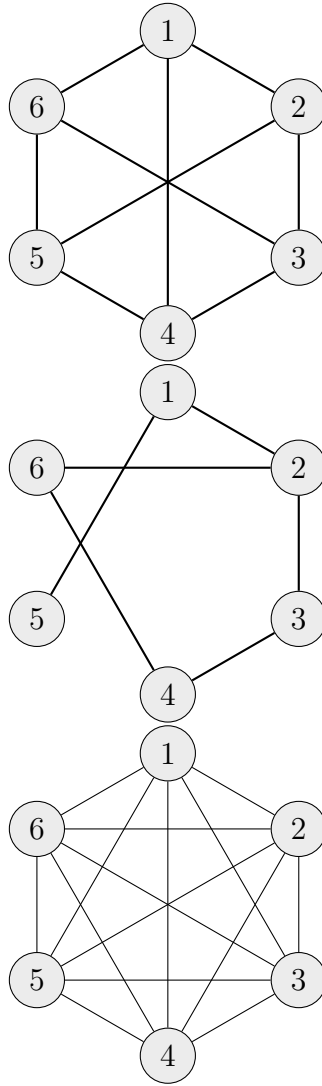


Figure 3.7: Example QAOA interaction graphs for four qubits: (a) a 3-regular graph, which for four nodes coincides with the complete graph K_4 , (b) a random graph with irregular connectivity, and (c) an all-to-all (complete) graph. Nodes represent qubits and edges denote two-qubit cost Hamiltonian interactions.

3.3. Problem Motivation

Quantum computing promises computational speedups for classes of problems that are intractable on classical hardware, particularly in optimization, simulation, and cryptography. However, today’s quantum processors remain constrained by limited device scale and non-negligible error rates, placing most platforms in the Noisy Intermediate-Scale Quantum (NISQ) regime where circuit depth and algorithmic fidelity are strongly limited by noise [47]. While there are effective strategies such as quantum multiprogramming [42] [43] as viable options for improving quantum hardware utilization using software approaches, challenges such as crosstalk and decoherence still exist and there is a need for a radical large-scale solution. Achieving fault-tolerant quantum computation (FTQC) is widely expected to require quantum error correction (QEC), which introduces substantial overhead by encoding each logical qubit into many physical qubits. Consequently, scaling toward FTQC is increasingly viewed as an architectural challenge: even if high-quality qubits exist, building a monolithic machine with the required number of physical qubits, connectivity, and control may be impractical in the near term. Additionally, one of the concerns could be the large monolithic systems being equivalent to a singular point-of-failure in scenarios where the system encounters technical disruptions or goes offline.

A compelling alternative is Distributed Quantum Computing (DQC), which interconnects multiple smaller quantum processing units (QPUs) into a larger system via entanglement and classical coordination. In principle, DQC trades monolithic integration for modular growth, enabling larger effective problem sizes by pooling resources across networked modules. This direction is strengthened by continued progress in networked quantum systems, including experimental demonstrations of distributed computation across photonic links between separated modules. [7] In parallel, data-center networking research has shown that optical switching may enable scalable, reconfigurable, high-bandwidth fabrics in classical infrastructure, an idea increasingly discussed as an enabling substrate for quantum data-center architectures that require frequent reconfiguration and low-contention connectivity. [26] [44] These trends motivate a need for systematic evaluation tools that jointly consider quantum programs, entanglement generation, and network topology before large-scale quantum data centers exist as

deployable hardware.

In this thesis, we build on SwitchQNet as a compiler framework introduced to optimize distributed quantum computing for quantum data centers with switch networks by scheduling communication across program and network layers and explained via resource-aware latency and overhead analyses. While SwitchQNet provides a structured way to study network-aware scheduling, its benchmarking suite could be broadened to better represent emerging algorithmic workloads, especially optimization algorithms that represent solutions to important applications. SwitchQNet possesses the defined layout of software compilation and compilation switches over a description of distributed networking hardware, and the potential for an extensive benchmarking algorithm suite, but it still needs the right selections and the data to justify the benefits to hardware. Additionally, as with many research prototypes, extending SwitchQNet to support new workloads benefits from clarifying assumptions and documenting topology and routing parameters in a way that reduces friction for replication and further development. We focus on the Quantum Approximate Optimization Algorithm (QAOA) as our choice of a prominent quantum algorithm for combinatorial optimization. Although QAOA is a heuristic algorithm, it is also an important algorithm introduced to approach complex combinatorial problems with shallow-depth instances [25]; this makes it particularly interesting as a diagnostic workload for understanding how distributed architectures and network topologies affect execution cost. This brings us to the motivation for integrating QAOA in a DQC setting, to evaluate inter-QPU entanglement and enabling controlled routing experiments that isolate how parameters such as topology, routing look-ahead, and entanglement management impact latency and resource overhead. Generating effective metrics for QAOA system performance shall prove to be effective in system co-design, not only benefiting software-based optimization but also hardware implementations when engineers design both near-term testbeds and scaled-up large distributed networks of the future, providing useful data to improve the cost-efficiency of valuable time and resources.

Problem statement: Despite growing interest in distributed and modular quantum architectures, there remains limited tooling to evaluate how a widely used optimization workload such as QAOA behaves under network constraints

across multiple topologies, using a compiler that schedules entanglement resources. This thesis addresses that gap by extending SwitchQNet with QAOA as a benchmark program, and using the extended framework to quantify performance trends across network topologies (Clos, fat-tree-style) under consistent routing/scheduling assumptions. We expect (i) numerical characterization of QAOA’s sensitivity to network structure and (ii) a more extensible benchmarking pathway for future distributed quantum algorithm studies within SwitchQNet.

Chapter 4

Methodology and Experimental Results

4.1. Solution methodology

In order to integrate the QAOA program into the existing SwitchQNet hand-routing framework, we propose an inter-QPU routing schedule for the QAOA algorithm in order to evaluate inter-QPU communication consistently alongside existing benchmarking programs.

First, we write a function that identifies the required connections between QPUs in the identified network setup in order to execute the two-qubit CX gates defined by the problem's graph edges. When the module that runs the network routing to return metrics for the test (`run_hand_routing_one.py`) calls all functions from the scheduling module, the connection graph module, and the hand routing module, the conditional parameter checks for the type of connection graph, provides a QPU mapping, over the connection graph, and checks which benchmarking algorithm is selected. When QAOA is selected, the function takes in an integer for the number of racks in the quantum hardware defined by `rack_num`, an integer for the number of QPUs per rack defined by `qpu_per_rack`, an integer for the number of qubits per QPU defined by `qbit_per_qpu`, and a list of tuples where each tuple (u, v) represents an edge in the problem graph corresponding to a $CX(u,v)$ - $Rz(v)$ - $CX(u,v)$ gate optimization operation. This function, defined by `gen_qaoa_routing_result`, then returns a list of required connections where each

connection is formatted as [source_qpu, target_qpu, 'CX']. The list is repeated for p rounds upon the application of an optional input argument 'p' which takes in an integer and represents the number of QAOA layers, however for the sake of simplicity we have set p to default as 1.

The flow of the function is as follows:

- Calculate the total number of QPUs and qubits in the system.
- Build the routing plan for a single QAOA layer at a time. This list will hold the required connections for one application of the Problem Unitary U_C . The Mixer Unitary only uses single-qubit gates and thus requires no inter-QPU routing.
- Iterate over every edge in the problem graph.
- Check if qubits u and v are within the total number of available qubits.
- Determine the QPU for each qubit in the edge using integer division.
- Core routing logic: Only record a connection if the two qubits are on different QPUs. The edge (u, v) corresponds to a $CX(u, v) - Rz(v) - CX(u, v)$ sequence. The Rz gate is local and requires no routing. Both CX gates require the same remote connection between qpu_u and qpu_v . Then schedule the connection for the first CX gate, and then for the second CX gate.
- Finally, assemble the full routing plan by repeating the layer p times (1 time in our case).
- Use deepcopy to ensure each layer is a distinct object in the list.

Next, we write the graph functions for mapping the QAOA algorithm.

For the k -Regular graph, we create a function that generates the edge list where each node has a fixed degree. This uses the "pairing model", taking in the total number of qubits or nodes of our graph as an integer "num_qubits", and the desired degree for each node as an integer. For our case, we shall keep it to the value 3 for a 3-regular graph. This returns a list of tuples representing the graph edges, which we send to our QAOA routing result function described previously.

We have also written the graph functions for random-QAOA and all-to-all QAOA which may be selected as needed.

The steps are:

- For a regular graph, the product of num_qubits and degree must be even.
- Next, we create a list of 'stubs' for each node across the range of qubits and the degree.
- We use a set function to automatically handle multi-edges.
- We pair up stubs with the edge variables to create edges for the routing result function that calls this graph.
- We write an if-else condition avoid self-loops (a node connecting to itself). Whenever the two edges compared are not equal, we add the edges in a canonical order (min, max) to prevent duplicates like (1,0) and (0,1).
- We finish by returning a list of k-regular graph edges.

After the circuit graph and routing functions are ready, we perform the routing run of our QAOA algorithm. The main flow involves creation of the hardware configuration and the assignment of application variables. Then, we apply the routing logic for our application graph (QAOA) as described above. The program checks for in-rack connections, which are high-fidelity, and cross-rack connections, which are slower and low-fidelity. All of the inherent SwitchQNet tests are run and the results are generated as raw data outputs.

Next, we make several updates to the utility file 'Utils.py' where we modify the file data class, the function to compute dictionaries, and the function to extract data, to include results generated from QAOA tests as an additional tuple entry in each list.

We then move the raw data generated from the scheduler tests as .txt files containing metric values in JSON to be placed collectively in a single location for figure and table generation using the dataset. These files are then called in the figure and table generation Jupyter Notebooks using Python.

Following this, we update all of the figure generation notebooks to extract data for their corresponding tests from the relevant datapaths, adding QAOA as

a benchmark, writing the plotting code for all benchmark results’ figure generation and then saving the graphs as .jpeg, .pdf and .svg for later use. Additionally, we also add a codeblock pulling results just from the QAOA results and then saving them separately for a clearer focus of the results from just QAOA instead of a comparative graph plot.

4.2. Experiment Setup

4.2.1 Architecture Setup

We evaluate SwitchQNet using a detailed event-driven simulation of a modular quantum computing architecture. The Clos topology is used throughout the evaluation, consistent with the architectural assumptions adopted in prior SwitchQNet experiments. Unless otherwise stated, the system consists of four racks, each containing four quantum processing units (QPUs), resulting in a total of sixteen QPUs. Each QPU hosts thirty physical qubits, of which two qubits are reserved for communication, while the remaining qubits are allocated to computation and local caching. Communication qubits are shared between inter-QPU entanglement generation and cache reservation, reflecting hardware constraints.

Each rack is equipped with a rack-level router supporting four Bell-state measurement (BSM) units, enabling concurrent entanglement swapping operations. Cross-rack connections are modeled with higher routing weights than in-rack links to capture the increased latency and resource cost associated with long-distance entanglement generation.

The look-ahead depth is fixed at 10ms, while the latencies of in-rack & cross-rack EPR pair generation, and switch reconfiguration are set to 0.1ms, 10ms and 1ms respectively, according to SwitchQNet’s listed numbers when estimating rates and fidelity of in-rack and cross-rack EPR pair generation after considering certain hardware parameters. The settings of experiments are listed in Table 4.1. The program number is given by the total number of data qubits involved in the compilation program, for example, Program-480 represents a collection of 4 racks with 4 QPUs per rack and 30 data qubits per QPU, which total up to $4 \times 4 \times 30 = 480$. For the primary experiments, unless stated otherwise, we shall

also use Program-480 to maintain consistency among results for benchmarking and comparison.

Table 4.1: Program and architecture settings

Benchmark	#rack	#QPUs / rack	#Data Qubits / QPU	Buffer Size / QPU	Comm. Qubits / QPU
program-480	4	4	30	10	2
program-608	4	4	38	12	2
program-720	4	4	45	15	2
program-360	4	3	30	10	2
program-480	4	4	30	10	2
program-600	4	5	30	10	2
program-720*	4	6	30	10	2
program-240	4	3	20	7	2
program-540	9	3	20	7	2
program-960	16	3	20	7	2
spine-leaf-720	6	4	30	10	2
fat-tree-960	8	4	30	10	2

We shall work with the same latency bottleneck as in the source experiment, with high latency for cross-rack EPR pair generation at about 10ms, compared to in-rack EPR pair generation at about 0.1ms, due to converters facilitating the complex working and interaction between the classical core switches and the quantum core switches, and the added 1ms latency caused every time a switch is reconfigured whenever a photon loss is suppressed.

The QAOA graph used for these results is the 3-regular graph implementation as part of the hand routing module in the codebase, and all analysis will consider results from the 3-regular QAOA graph unless stated otherwise.

All time units are in milliseconds (ms) unless stated otherwise. Scheduling times wherever mentioned are in seconds (s).

The program is implemented in Python and evaluated on a MacBook Pro with 2.3 GHz 8-Core Intel Core i9 CPU and 32 GB memory. Operating System used is macOS 15.5.

4.2.2 Benchmark Programs

The pre-existing set of benchmark programs is kept for cross evaluation. The first one is the Multi-Control Target (MCT) gate [40] for representing impor-

tant non-Clifford operations that are essential for universality and which establish the efficiency of multi-qubit gate decomposition in quantum computing experiments. Quantum Fourier Transform (QFT) algorithm [17] and Ripple-Carry Adder (RCA) program are included because of their integral importance to Shor’s algorithm [41], as well as in other computing applications such as cryptography and signal processing [53]. The SwitchQNet compiler repeats the RCA circuit added for 100 iterations to adapt it to a sum calculation and increase its complexity. The compiler also includes the Grover’s Algorithm [31] as one of the benchmark programs to represent a quadratic quantum complexity simplification and improvement for unstructured list searches. Shor’s and Grover’s algorithms represent two of the most significant applications demonstrating an advantage of quantum computers.

As discussed in the previous section, we add QAOA as our choice of benchmark quantum program to evaluate routing behavior under non-trivial communication patterns. QAOA introduces structured two-qubit interactions determined by a problem graph, making it a representative workload for studying inter-QPU communication overhead in distributed quantum systems. Within QAOA, only the problem unitary requires two-qubit operations across logical qubits, while the mixer unitary consists solely of single-qubit gates and does not incur inter-QPU communication. As a result, the qubits are connected as a circuit in the framework laid out by the QAOA routing model. The program is thus integrated into the existing SwitchQNet benchmark framework using the same circuit abstraction and routing interface as the previously studied algorithms. This design allows QAOA to be evaluated alongside existing benchmarks without modifying the underlying routing or scheduling infrastructure, ensuring methodological consistency across experiments.

4.2.3 Metrics

To evaluate the compilation performance of the introduced benchmarking programs in SwitchQNet, we shall also consider the following metrics:

The first is the overall communication latency of the benchmarking programs, termed as ‘latency’ for simplicity from here, normalized by switch reconfiguration latency. These are all considered under inter-QPU communication while the intra-

QPU computation time is neglected being much quicker than inter-QPU times.

The second metric is the fidelity overhead, specifically the EPR overhead which is the number of additional distilled in-rack EPR pairs required arising due to communication splits, represented in terms of percentages of the original in-rack EPR pair numbers and evaluated in a weighted manner. These weights are defined as follows when performing compilation:

Given the 95% fidelity of in-rack, >96.5% fidelity of distilled in-rack, and 85% fidelity of cross-rack EPR pairs, they are each counted by weights 0.33, 0.23, and 1 respectively.

There is also an additional fidelity metric which is the average wait time of EPR pairs waiting in the buffer normalized by switch reconfiguration latency as well, but since the impacts of wait time are more dependent upon the QPU hardware in use, specifically on the coherence of the qubits involved in computation, not something that would be considered in network compilation here, we consider these separately and consider the EPR overhead percentage as the main contributor to fidelity effects here.

The third metric will be the retry overhead which shows the overhead for compilation time added by retry counts. Defined as:

$$\text{Retry Overhead} = \frac{t_{\text{attempts}}}{t_{\text{compilation}}}, \quad (4.1)$$

where t_{attempts} is the total attempted time step number in the compilation and $t_{\text{compilation}}$ is the actual number of time steps in the compilation results.

Based on these metrics, we shall be able to observe and assess the workload impacts of running QAOA on our choice of network setups.

4.2.4 Comparison Baseline

We use the same baseline as integrated by SwitchQNet for all benchmark tests, which is based on the QuComm compiler framework for general DQC communication [63], same as our preprocessing step in SwitchQNet. The EPR pair list is acquired by each of the benchmarking algorithms with pair number minimized through a buffer-aware compilation step equivalent to SwitchQNet, followed by the buffer-assisted on-demand approach of SwitchQNet.

4.3. Primary Experiment Results with QAOA

This section presents the primary experimental results obtained using the extended SwitchQNet framework with QAOA integrated as an additional benchmark program. The goal of these experiments is to evaluate how network-aware compilation and scheduling affect the performance of a structured optimization algorithm when key architectural parameters are varied. The results are summarized in Table 4.2, which reports latency, entanglement resource usage, and waiting overheads for both the baseline compiler and SwitchQNet across multiple scaling dimensions.

Across all experiments, SwitchQNet consistently outperforms the baseline compiler in terms of end-to-end latency for QAOA, although the magnitude of improvement is smaller than that observed for communication-intensive benchmarks such as Grover, QFT, and RCA. This behavior is expected given QAOA’s comparatively sparse and graph-dependent interaction pattern. When increasing the number of QPUs per rack, QAOA exhibits improvement factors ranging from approximately $1.9\times$ to $2.2\times$, indicating that SwitchQNet’s look-ahead scheduling and collective in-rack EPR generation are effective at hiding switch reconfiguration latency even for optimization workloads. Notably, the improvement factor does not increase monotonically with system size, reflecting the fact that QAOA’s communication demand grows with problem structure rather than purely with hardware scale.

When increasing the number of qubits per QPU, QAOA maintains stable improvement factors slightly above $2\times$. This suggests that increasing local computational capacity without proportionally increasing inter-QPU communication does not significantly change the relative benefit of network-aware scheduling. In this regime, QAOA remains primarily limited by cross-rack entanglement availability rather than local gate execution, and SwitchQNet continues to reduce idle waiting time by overlapping EPR generation with computation.

Scaling the number of racks highlights a different trend. As the system expands from the baseline configuration to larger rack counts, QAOA’s improvement factor remains modest, typically between $1.4\times$ and $2.0\times$. While absolute latency increases with additional racks due to higher cross-rack communication cost, SwitchQNet still achieves measurable gains over the baseline. The results indi-

cate that QAOA benefits from network-aware scheduling even in more distributed settings, but that its limited communication density constrains the achievable latency reduction compared to benchmarks with heavier cross-rack interaction.

The topology comparison further reinforces this observation. Under fat-tree and spine-leaf topologies, QAOA exhibits improvement factors close to those seen in Clos-based configurations. This suggests that QAOA is relatively insensitive to the specific topology choice compared to other benchmarks, provided that sufficient path diversity exists. As a result, QAOA serves as a useful contrast workload that exposes how algorithm structure influences the effectiveness of network-level optimizations.

Overall, these results demonstrate that while QAOA does not achieve the dramatic improvement factors observed for more communication-heavy programs, it consistently benefits from SwitchQNet’s scheduling strategies across all evaluated dimensions. This confirms QAOA’s suitability as both a realistic application workload and a diagnostic benchmark for studying the interaction between distributed quantum algorithms and network architectures.

4.3.1 Interpretation of Performance Metrics

Table 4.2 reports a comprehensive set of metrics designed to capture both performance and resource behavior in distributed quantum execution. Each column reflects a distinct aspect of the compilation and scheduling process.

The **Baseline Latency** represents the total execution time obtained using a baseline compiler that schedules entanglement generation on demand without advanced look-ahead or splitting optimizations. This value serves as a reference point for evaluating improvements.

The **Ours Latency** reports the corresponding execution time achieved using SwitchQNet’s network-aware scheduling. Improvements in this metric primarily arise from overlapping EPR generation with computation, reducing idle waiting caused by switch reconfiguration and communication delays.

The **Improvement Factor** is defined as the ratio of baseline latency to SwitchQNet latency. A value greater than one indicates that SwitchQNet achieves a reduction in end-to-end execution time.

The **Number of Cross-Rack EPR Pairs** quantifies how many entangled

pairs must be generated across racks to support non-local interactions. This metric reflects the inherent communication demand of the benchmark and increases with greater distribution across racks.

The **Number of In-Rack EPR Pairs** counts entanglement generated within a rack. These pairs are typically cheaper to generate and are often used to support collective generation and post-split recovery.

The **Ours: Number of Distilled EPR Pairs** captures the additional in-rack EPR pairs consumed during entanglement distillation, particularly after cross-rack splitting. This metric directly contributes to the EPR overhead introduced by latency-hiding strategies.

The **Baseline Wait Time** and **Ours Wait Time** represent the cumulative time spent waiting for switch reconfiguration and communication resources under the baseline and SwitchQNet schedulers, respectively. The **Additional Wait Time** is the difference between these two values and highlights cases where increased buffering or speculative EPR generation introduces extra waiting that is nevertheless offset by latency reduction elsewhere.

Finally, the **Retry Overhead** records the fraction of execution affected by scheduling retries due to deadlock or congestion. Across all QAOA experiments, this value remains negligible, indicating that QAOA’s communication pattern aligns well with SwitchQNet’s scheduling model and does not induce pathological behavior.

Taken together, these metrics provide a holistic view of how distributed quantum programs behave under network constraints, and they clarify why QAOA exhibits smaller but stable performance gains compared to more communication-intensive benchmarks.

Table 4.2: Performance Analysis of SwitchQNet and Baseline Compiler Across Quantum Benchmark Programs Including QAOA. Units of Latency (ms) and wait time are the latency of switch reconfiguration.

Experiment	Benchmark	Baseline: Latency	Ours: Latency	Improv. Factor	#cross-rack EPR	#in-rack EPR	Ours: #dist. EPR	Baseline: Wait Time	Ours: Wait Time	Addit. Wait Time	Retry Overh.
Increase #QPU/rack	MCT-360	1,476	468	3.15 ×	15	144	15	3.03	5.81	2.78	0.00
	MCT-480	2,312	485	4.77 ×	15	240	13	1.98	6.75	4.77	0.00
	MCT-600	3,214	634	5.07 ×	15	432	12	1.17	5.30	4.13	0.00
	MCT-720	4,413	921	4.79 ×	15	624	11	0.87	5.27	4.40	0.00
	QFT-360	78,300	13,504	5.80 ×	810	1,500	715	1.48	6.27	4.79	0.00
	QFT-480	121,728	16,693	7.29 ×	1,080	2,970	1,133	1.30	6.87	5.57	0.00
	QFT-600	169,831	20,041	8.47 ×	1,350	4,920	1,559	1.14	7.00	5.86	0.00
	QFT-720	216,372	23,362	9.26 ×	1,620	7,350	1,915	1.20	7.63	6.43	0.00
	Grover-360	140,813	29,717	4.74 ×	1,800	4,800	1,594	2.03	9.96	7.93	0.00
	Grover-480	156,213	26,943	5.80 ×	1,800	7,200	1,927	2.08	8.73	6.65	0.00
	Grover-600	171,613	25,438	6.75 ×	1,800	9,600	2,057	2.08	8.77	6.68	0.00
	Grover-720	187,013	24,580	7.61 ×	1,800	12,000	2,178	2.07	8.85	6.77	0.00
	RCA-360	83,470	10,127	8.24 ×	603	1,608	531	0.03	9.69	9.66	0.00
	RCA-480	92,259	9,169	10.06 ×	603	2,412	630	0.03	8.49	8.46	0.00
	RCA-600	101,048	8,916	11.33 ×	603	3,216	683	0.03	8.68	8.64	0.00
	RCA-720	109,837	8,592	12.78 ×	603	4,020	710	0.03	8.78	8.75	0.00
	QAOA-360	18,438	9,835	1.87 ×	782	192	409	0.68	19.60	18.92	0.00
	QAOA-480	20,704	10,363	2.00 ×	1,064	282	737	0.56	17.82	17.25	0.00
	QAOA-600	22,992	10,584	2.17 ×	1,348	352	1,028	0.47	16.38	15.91	0.00
	QAOA-720	21,650	11,051	1.96 ×	1,604	456	1,237	0.45	15.93	15.48	0.00
Increase #qubits/QPU	MCT-480	2,312	485	4.77 ×	15	240	13	1.98	6.75	4.77	0.00
	MCT-608	2,312	454	5.09 ×	15	240	15	1.98	6.36	4.39	0.00
	MCT-720	2,312	382	6.05 ×	15	240	16	1.98	8.23	6.25	0.00
	QFT-480	121,728	16,693	7.29 ×	1,080	2,970	1,133	1.30	6.87	5.57	0.00
	QFT-608	155,960	20,781	7.50 ×	1,368	3,762	1,452	1.26	6.92	5.66	0.00
	QFT-720	194,526	24,670	7.89 ×	1,620	4,455	1,772	1.00	7.77	6.77	0.00
	Grover-480	156,213	26,943	5.80 ×	1,800	7,200	1,927	2.08	8.73	6.65	0.00
	Grover-608	150,702	27,412	5.50 ×	1,800	7,200	1,933	1.87	9.54	7.67	0.00
	Grover-720	156,213	25,883	6.04 ×	1,800	7,200	2,122	2.08	9.14	7.06	0.00
	RCA-480	92,259	9,169	10.06 ×	603	2,412	630	0.03	8.49	8.46	0.00
	RCA-608	92,259	9,304	9.92 ×	603	2,412	650	0.03	8.57	8.54	0.00
	RCA-720	92,226	9,395	9.82 ×	603	2,404	658	0.01	9.78	9.77	0.00
	QAOA-480	20,704	10,363	2.00 ×	1,064	282	737	0.56	17.82	17.25	0.00
	QAOA-608	26,952	12,515	2.15 ×	1,356	352	1,048	0.63	17.86	17.23	0.00
	QAOA-720	30,582	14,503	2.11 ×	1,604	440	1,397	0.54	19.88	19.34	0.00
Increase #racks	MCT-240*	1,575	490	3.21 ×	15	144	14	2.74	4.66	1.92	0.00
	MCT-540	6,514	3,069	2.12 ×	99	900	52	3.12	5.14	2.03	0.00
	MCT-960	15,369	5,415	2.84 ×	255	2,304	170	3.21	5.44	2.23	0.00
	QFT-240*	50,195	9,663	5.19 ×	540	1,000	389	1.61	6.25	4.64	0.00
	QFT-540	212,522	28,694	7.41 ×	2,640	4,900	1,802	2.36	6.28	3.93	0.00
	QFT-960	564,973	58,482	9.66 ×	8,100	15,400	4,250	3.05	6.61	3.56	0.00
	Grover-240*	147,468	34,265	4.30 ×	1,800	4,800	1,248	1.00	9.54	8.54	0.00
	Grover-540	365,312	38,648	9.45 ×	4,800	10,800	3,071	0.99	8.63	7.65	0.00
	Grover-960	665,612	39,969	16.65 ×	9,000	19,200	4,576	0.98	7.99	7.01	0.00
	RCA-240*	83,470	11,050	7.55 ×	603	1,608	432	0.03	8.91	8.88	0.00
	RCA-540	213,523	12,666	16.86 ×	1,608	3,618	853	0.08	7.10	7.02	0.00
	RCA-960	399,478	13,240	30.17 ×	3,015	6,432	924	0.03	7.03	7.00	0.00
	QAOA-240*	13,543	8,082	1.68 ×	530	122	191	0.49	12.52	12.03	0.00
	QAOA-540	18,196	9,231	1.97 ×	1,430	126	701	0.34	17.25	16.91	0.00
	QAOA-960	20,000	13,818	1.45 ×	2,706	124	843	0.11	13.66	13.55	0.00
Fat-tree top.	MCT-960	6,910	2,497	2.77 ×	63	960	30	2.44	5.82	3.38	0.00
	QFT-960	421,181	49,568	8.50 ×	4,200	11,610	3,107	1.99	9.40	7.41	0.00
	Grover-960	327,813	39,193	8.36 ×	4,200	14,400	2,420	2.28	12.80	10.52	0.00
	RCA-960	206,373	12,639	16.33 ×	1,407	4,824	896	0.03	10.52	10.49	0.00
Spine-leaf top.	QAOA-960	30,604	21,782	1.41 ×	2,532	266	203	0.30	12.61	12.31	0.00
	MCT-720	4,611	1,008	4.57 ×	39	600	33	2.24	6.60	4.36	0.00
	QFT-720	249,317	34,657	7.19 ×	2,400	6,570	1,767	1.77	9.89	8.12	0.00
	Grover-720	242,013	34,357	7.04 ×	3,000	10,800	2,001	2.22	11.63	9.42	0.00
	RCA-720	149,316	11,418	13.08 ×	1,005	3,618	679	0.03	10.36	10.33	0.00
	QAOA-720	27,249	19,142	1.42 ×	1,788	272	112	0.45	13.05	12.60	0.00

4.4. Latency

As described in the paper, this communication budget is given a graphical representation by organizing the quantum workload across the various types of operations part of the compiler evaluation suite. The required number of in-rack and cross-rack EPR pairs are calculated and the different operation contributions to the overall latency are isolated. In the first step, all latency readings are assigned only to cross-rack communications by fixing the latency of in-rack communications and reconfiguration to 0, and in the next step the latency difference between the two categories is assigned to the reconfiguration setting by fixing the in-rack communication latency as 0. These observations are made as an average across all of the test connection graphs, Clos, Fat-tree and Spine-Leaf topologies.

What we observe as a result of this is that the number percentages shared between in-rack and cross-rack EPR pairs, and the latency percentages shared between in-rack EPR pair generation, reconfiguration and cross-rack EPR pair generation are noticeably different with the introduction of QAOA as a benchmark metric in SwitchQNet. In the updated analysis after QAOA operation contributions are taken into consideration, we observe the following metrics:

Cross-rack EPR pairs now constitute 30.2% of the total required EPR pairs which is a 12% increase from the case without the inclusion of QAOA operations as seen in **Figure 4.1**. They also account for 68% of the overall latency as seen in **Figure 4.2**, showing a 5.3% increase than the previous results from the original study with reconfigurations [68] contributing an additional 27.9% latency resulting in a 4.8% decrease than before, reducing the share of in-rack latency by 0.5%. This observation seems to be in line with the previous claim that latency keeps growing significantly in the system with increased cross-rack communication even when switch reconfigurations are a little less frequent, implying that cross-rack communications have a more significant impact on overall system latency. This also shows the degrading the net fidelity of the system as qubit decoherence rises over time.

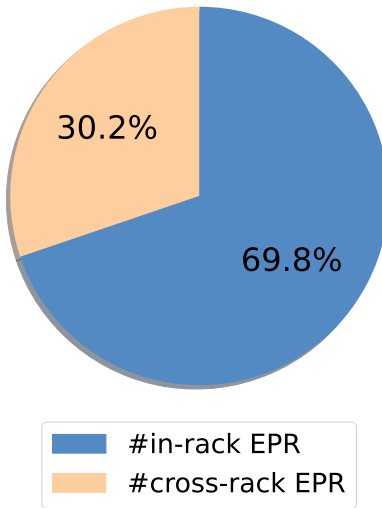


Figure 4.1: Number percentages shared between in-rack and cross-rack EPR pairs including QAOA contributions

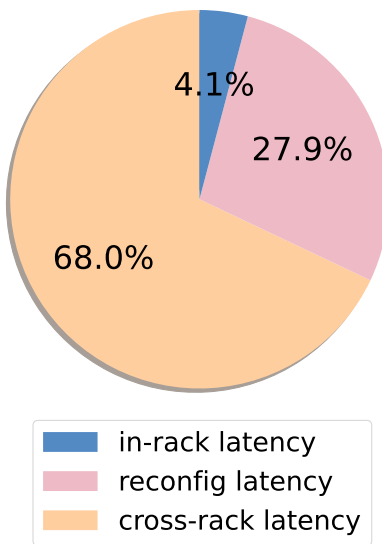


Figure 4.2: Latency percentages shared between in-rack EPR pair generation, reconfiguration and cross-rack EPR pair generation including QAOA contributions

4.5. Compilation Switches

Buffer size: In this test, we check the effect of buffer size when varied while keeping the rest of the parameters unchanged (need to check choice of program (program-480) and architecture from table 1 and figure codes).

As seen in **Figure 4.3**, the buffer size turning point for QAOA is around 11, which takes $11/(30 + 11) = 26.8\%$ of the total number of qubits per QPU (#qubits), different from the turning points of QFT, Grover and RCA observed around 7, as well as MCT which has a turning point at around 20 since it is influenced more strongly by in-rack communications compared to other benchmarks and needs a larger buffer size. For context, in-rack EPR pairs may be captured and stored in the buffer without much restriction by the bandwidth of the network. This observation would imply that QAOA has a moderate requirement of buffer qubits when compared to other benchmarking programs as part of SwitchQNet.

Look-ahead depth: The effects of look-ahead depth on the overall latency of the baseline and the SwitchQNet compiler including the contributions from QAOA are also highlighted in a similar process where it is made to vary and the rest of the parameters remain unchanged.

As noted for the other benchmarking programs in **Figure 4.4**, the latency decreases initially as the look-ahead depth is increased until it reaches somewhat of a saturation point and becomes more stable and the latency does not fluctuate a lot, however we do see a brief increase in latency again around look-ahead depth 12. The same is observed for the improvement factor, first increasing and then soon after reaching stability, with a brief decrease at depth 12.

QAOA shows similar trends to QFT, Grover and RCA, expresses a similar curve trend and has a smaller turning point, with the differences being in a much smaller improvement factor trend across the graph and a slight increase in latency and decrease in improvement factor for higher look-ahead depths.

4.6. Sensitivity analysis

Continuing the evaluation methods from the original paper, we perform an analysis for the sensitivity of the numerous hardware parameters to observe how

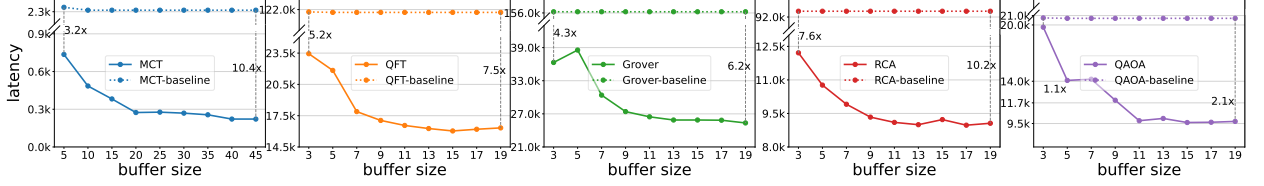


Figure 4.3: Performance improvement of compiler varying with buffer-size including QAOA

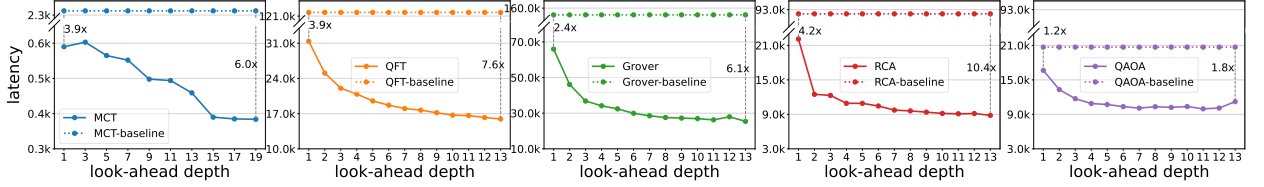


Figure 4.4: Performance improvement of compiler varying with look-ahead depth including QAOA

well the SwitchQNet compiler adapts to a varying supply of EPR pairs, which is expressed in terms of the amount of communication qubits per QPU and the latencies of cross-rack EPR generation and in-rack EPR generation, as well as to a variable quality of EPR pairs, expressed by the fidelity values of cross-rack, in-rack and distilled EPR pairs. The ratios between different latencies and fidelities are the only metrics of importance to measure sensitivity, hence the cross-rack latency and in-rack latency are normalized by reconfiguration latency whereas cross-rack and distilled in-rack fidelity are normalized by the initial in-rack fidelity.

Communication qubit number: The number of communication qubits per QPU are increased from 1 to 6 while keeping all the remaining parameters unchanged (program-480).

The results in **Figure 4.5** show that the overall latency of the baseline implementation as well as the SwitchQNet compiler first decreases and then eventually reaches a saturation point arising from increasing bandwidth resulting from an increase in the number of communication qubits. After considering QAOA contributions, we see that QAOA trends are slightly different compared to the other benchmarking programs. The baseline and SwitchQNet latency converge for QAOA as the number of communication qubits increase. In accordance to these observations, SwitchQNet’s improvement factor after QAOA considerations also differ from the other cases where it decreases and then reaches stability,

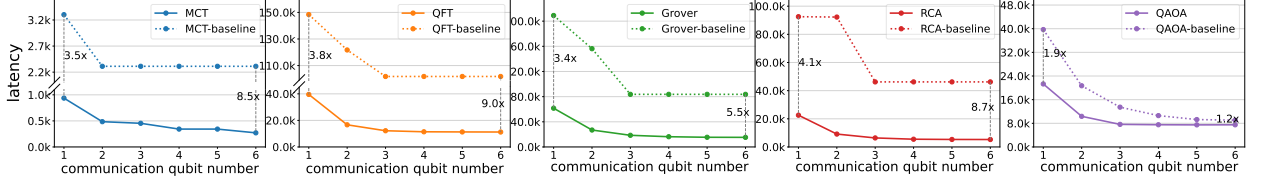


Figure 4.5: SwitchQNet performance improvement with QAOA contributions varying with communication qubits per QPU

showcasing a more effective utilization of bandwidth compared to the baseline implementation for a smaller number of communication qubits.

Cross-rack EPR latency: For this improvement factor test, we evaluate the effect of varying the latency of cross-rack EPR pair generation. To perform this, we take the ratio of cross-rack latency to the reconfiguration latency from 5 to 30 while all the other parameters stay consistent as per Program-480. This means that we vary cross-rack latency from five times the value of the observed reconfiguration latency, all the way to having a cross-rack latency thirty times the reconfiguration latency values.

As we see in **Figure 4.6**, while all the benchmarking programs show a similar trend of having the baseline and SwitchQNet compilers' overall latency increase with cross-rack latency, which makes sense since a longer cross-rack latency would also directly lead to a longer overall latency of a given system, we see that the QAOA case exhibits a slight increase in the improvement factor for the SwitchQNet compiler over the baseline compiler compared with other programs as the cross-rack latency is increased. The overall latency numbers are also much lower for QAOA compared with other programs, meaning that the system experiences fewer effects of latency overall when running QAOA. While only a slight improvement over other programs such that the improvement factors remain significant overall across all programs when having a cross-rack latency between 5 and 30 times the reconfiguration latency, it shows that with larger scaled-up networks, our implementation has the potential for a better overall performance when running the QAOA program implementation on future compiler and even hardware implementations.

In-rack EPR latency:

Similar to the previous test, we evaluate the effect of varying the latency of

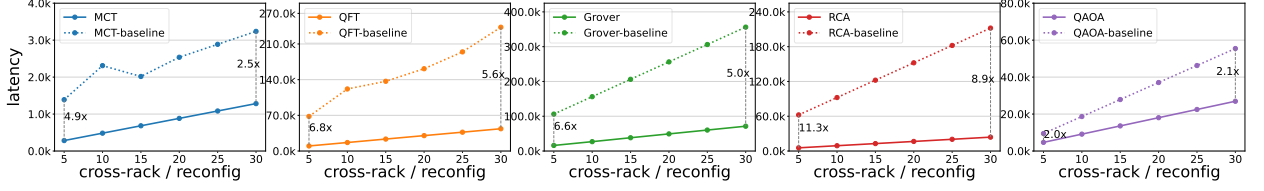


Figure 4.6: SwitchQNet performance improvement with QAOA contributions varying with cross-rack EPR latency normalized by reconfiguration latency

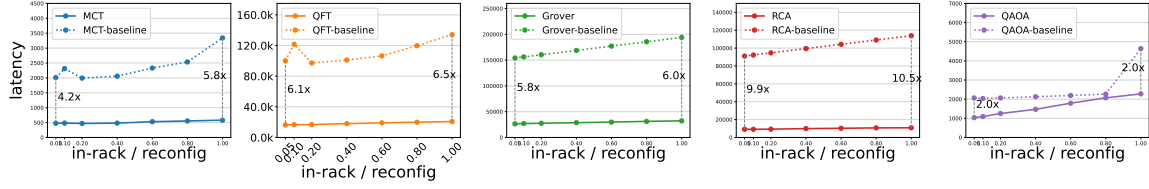


Figure 4.7: SwitchQNet performance improvement with QAOA contributions varying with in-rack EPR latency normalized by reconfiguration latency

in-rack EPR pair generation. To perform this, we take the ratio of in-rack latency to the reconfiguration latency from 0.05 to 1 while all the other parameters stay consistent as per Program-480. This means that we vary in-rack latency from 0.05 times to being equal to the value of the observed reconfiguration latency.

As we see in **Figure 4.7**, similar to the previous test case, since an increase in in-rack latency leads to increased overall latency, here we also see the results remaining consistent with the theory by showing an increase in overall latency of both the baseline compiler and SwitchQNet with increased in-rack latency. However, once again we see some results in contrast to cross-rack latency. Similar to MCT and QFT, QAOA shows some inflection points across the graph with first a decrease in the improvement factor when the in-rack latency is around 80% of the reconfiguration latency, and then shows a sharp increase in the improvement factor when the in-rack and reconfiguration latency ratio is 1 upto around the same values as the beginning when this ratio was the lowest.

Fidelity Analysis: In this part of the sensitivity analysis, we study the assumed physical fidelities of EPR pairs at different stages of the system. Each term corresponds to a different location and preparation method of entanglement in SwitchQNet. In-rack fidelity is the baseline fidelity of raw EPR pairs generated within the same rack. It represents short-distance entanglement within the same

rack, assumed to be relatively reliable but still noisy, used as the input to distillation protocols, and fixed in our analysis to isolate the effect of other variables. For example, A raw, directly generated in-rack EPR pair succeeds with fidelity 0.95. In-rack fidelity of EPR pairs will be used here to set a baseline assumption for the local entanglement quality. The distilled in-rack fidelity represents entanglement distillation applied to in-rack EPR pairs. In this scenario, multiple in-rack EPR pairs are consumed to produce a single higher-fidelity EPR pair, which comes at the cost of extra EPR pairs and extra latency. Distillation trades quantity for quality, improving fidelity by consuming additional raw in-rack EPR pairs. This will be varied to study distillation overhead and model quality-cost tradeoffs. The cross-rack fidelity models long-distance EPR generation between racks. As discussed before, cross-rack links suffer from a number of challenges such as longer fiber lengths leading to higher photon loss and therefore more decoherence. As a result, the fidelity of cross-rack EPR pairs is lower and more variable (noisy) than in-rack EPR pairs. These will be varied to study robustness of the routing protocols and model effects of long-distance noise. As we will see, these are controlled fidelity assumptions that will help observe how sensitive distributed quantum routing is to entanglement quality at different physical layers.

Relative cross-rack fidelity: This test is to observe the effect of changing fidelity of cross-rack EPR pairs. For this, the relative cross-rack fidelity ratio is varied as $F_{\text{cross}}/F_{\text{in}}$, which is varied from 0.79 to 1.0 by fixing the fidelity of in-rack EPR pairs $F_{\text{in}} = 0.95$, i.e. 95%, and sweeping the fidelity of the cross-rack EPR pairs $F_{\text{cross}} \in [0.75, 0.95]$, between 75% and 95%.

$$\frac{F_{\text{cross}}}{F_{\text{in}}} \in \left[\frac{0.75}{0.95}, 1 \right] = [0.79, 1.0]. \quad (4.2)$$

All other parameters remain unchanged. As the relative fidelity approaches 1, the compiler increasingly exploits cross-rack communication by generating additional EPR pairs to hide latency, leading to a modest increase in EPR overhead. From the generated **Figure 4.9**, we see that QAOA’s EPR pair overhead increases by about 38%, more than the other benchmark programs in comparison. As the fidelity of cross-rack and in-rack EPR pairs becomes more similar, the overhead cost of adding more in-rack EPR pairs to maintain communication quality also increases.

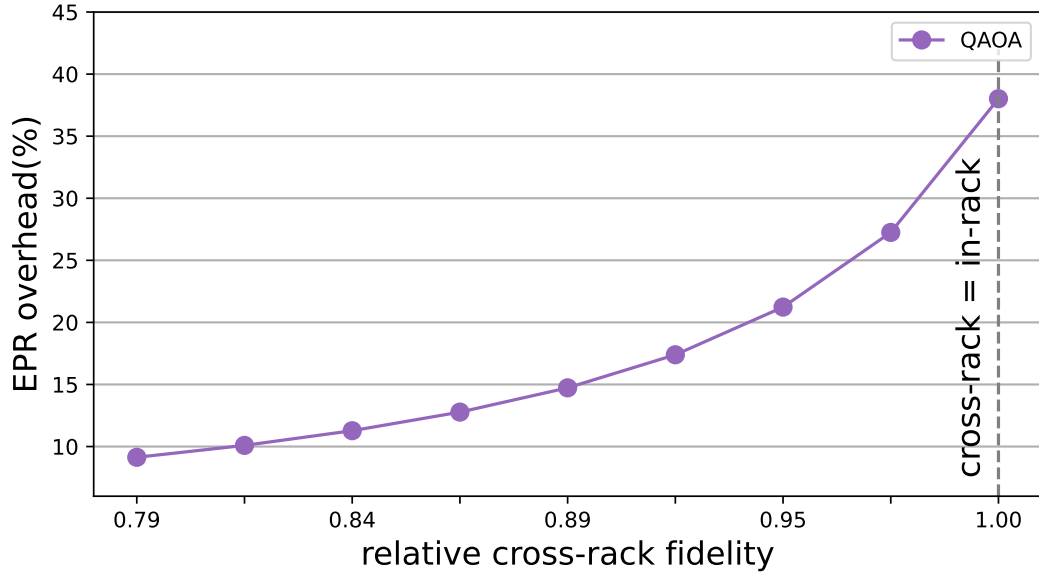


Figure 4.8: SwitchQNet fidelity overhead for QAOA, varying with cross-rack fidelity relative to original in-rack fidelity

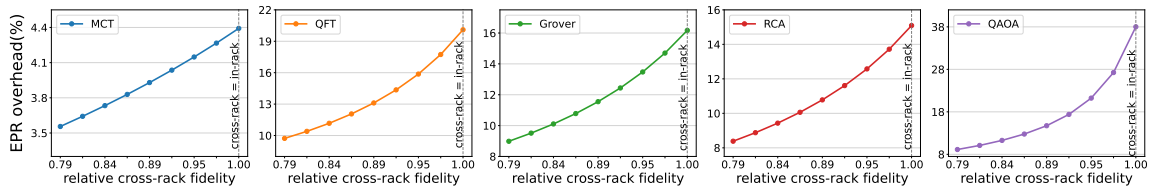


Figure 4.9: SwitchQNet fidelity overhead with QAOA contributions varying with cross-rack fidelity relative to original in-rack fidelity

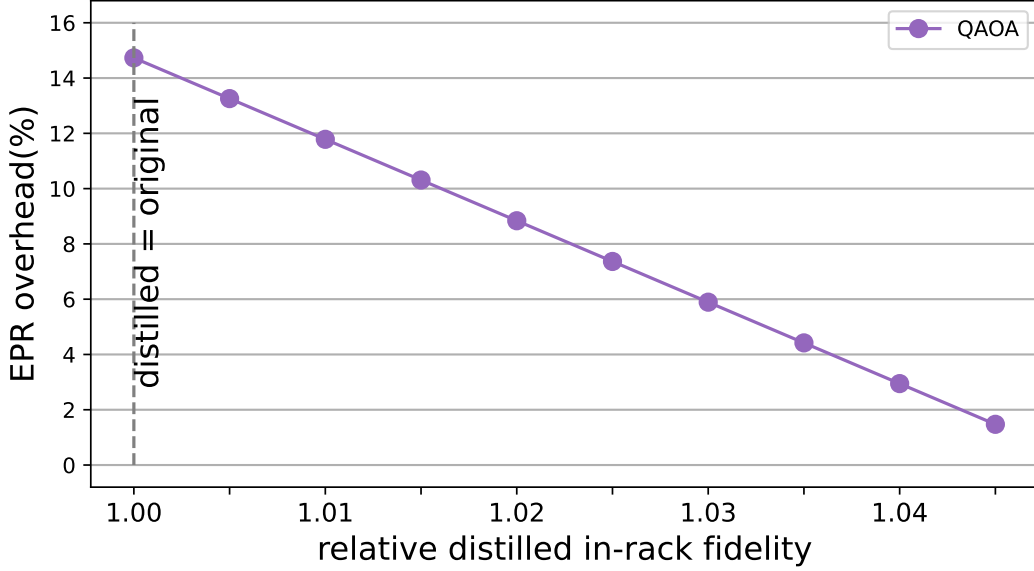


Figure 4.10: SwitchQNet fidelity overhead for QAOA, varying with distilled in-rack fidelity relative to original in-rack fidelity

Relative distilled in-rack fidelity: Similar test as the previous, we use this to observe the effects of relative distilled in-rack fidelity of EPR pairs, where entanglement distillation is updated via better protocols and consuming additional EPR pairs for distillation. The relative distilled in-rack fidelity ratio is varied as $F_{\text{distilled}}/F_{\text{in}}$, which is varied from 1 to 1.047 by fixing the fidelity of in-rack EPR pairs $F_{\text{in}} = 0.95$, i.e. 95%, and sweeping the fidelity of the distilled in-rack EPR pairs $F_{\text{cross}} \in [0.95, 0.99]$, between 95% and 99%. All other parameters remain unchanged.

As we see in **Figure 4.11**, the EPR overhead percentage of SwitchQNet goes down with the relative distilled in-rack fidelity for all benchmark programs including QAOA, which should imply that any fidelity overhead introduced by the additional in-rack EPR pairs may be negligible given that they are distilled to a high fidelity level.

#EPR pairs per distillation: When improving fidelity of distilled in-rack EPR pairs via the consumption of extra in-rack EPR pairs generated collectively within the SwitchQNet compiler, we see in **Figure 4.13** for the case of QAOA as well that this leads to a partial but unimpactful rise in the overall latency as

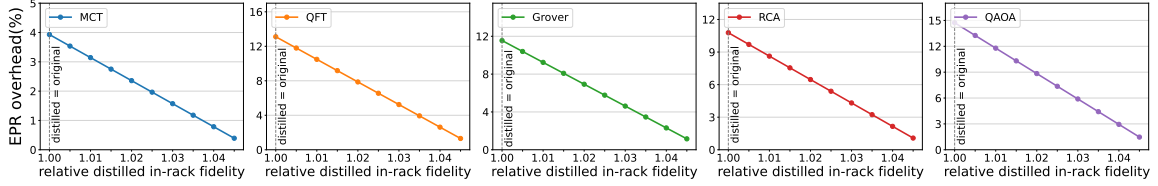


Figure 4.11: SwitchQNet fidelity overhead with QAOA contributions varying with distilled in-rack fidelity relative to original in-rack fidelity

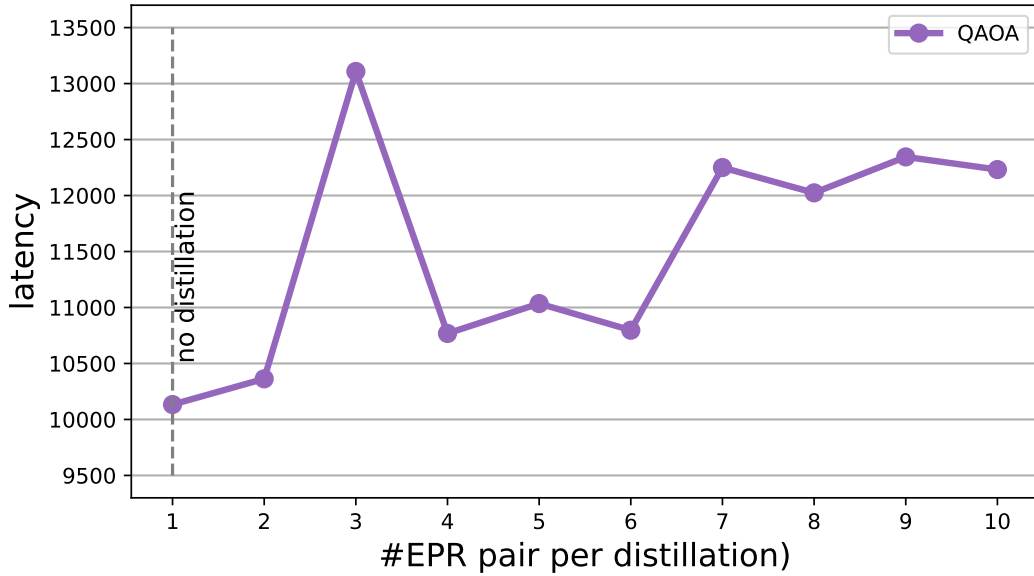


Figure 4.12: SwitchQNet overall latency for QAOA, varying with distilled in-rack fidelity across different EPR pair amounts

the distillation is increased, which remains consistent with results observed across previous benchmark programs.

Overall, the sensitivity analysis demonstrates that the combined use of look-ahead scheduling, join, and split mechanisms consistently yields the highest performance improvement across all benchmark programs, including QAOA. While the magnitude of improvement varies with algorithm structure, the results confirm that coordinated scheduling and communication-aware optimizations are critical for sustaining performance gains under diverse network and system configurations, as summarized in Figure 4.14.

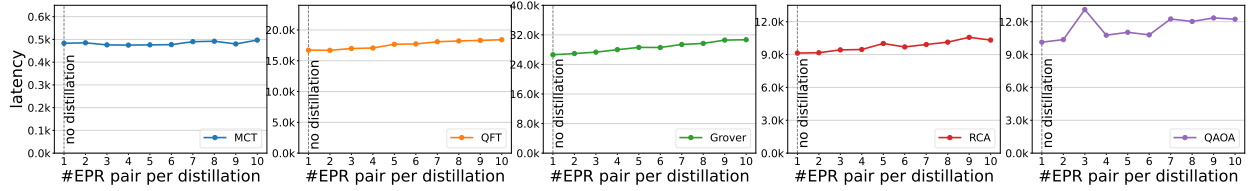


Figure 4.13: SwitchQNet overall latency with QAOA contributions varying with distilled in-rack fidelity across different EPR pair amounts

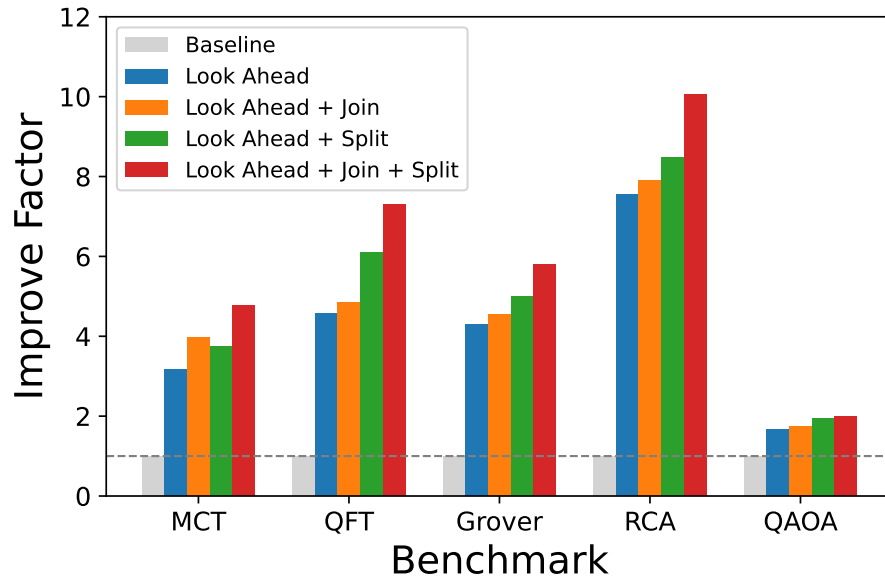


Figure 4.14: SwitchQNet Overall Improvement Factor with QAOA

4.7. QEC Integration

To evaluate the impact of quantum error correction on distributed quantum workloads, we integrate a surface-code-based QEC model with code distance $d = 5$ into the SwitchQNet compilation and scheduling framework. As in the original SwitchQNet study, logical operations are mapped onto physical qubits with additional communication and distillation overheads introduced by QEC, allowing latency and resource consumption to be evaluated under fault-tolerant assumptions. This integration enables a consistent comparison between baseline scheduling and SwitchQNet’s network-aware compilation strategy across multiple benchmark programs, including the newly added QAOA workload.

Table ?? and Fig.4.15 summarize the latency improvement factors achieved under QEC integration. Across previously studied benchmarks, SwitchQNet maintains significant latency reductions relative to the baseline, with improvement factors of $4.05\times$ for MCT, $4.46\times$ for QFT, $6.80\times$ for Grover, and $4.23\times$ for RCA. These results are consistent with the observations reported in Section 5.5 of the original SwitchQNet paper, confirming that the compiler’s scheduling optimizations remain effective even when QEC overheads are incorporated.

For QAOA-480, the improvement factor is more modest at $1.51\times$, reflecting the algorithm’s comparatively sparse and structured communication pattern. Unlike Grover or QFT, which exhibit dense or repetitive cross-rack entanglement demands, QAOA’s graph-dependent interactions result in fewer opportunities for aggressive latency hiding and collective EPR generation under the same QEC constraints. Nevertheless, SwitchQNet still achieves a measurable reduction in end-to-end latency, demonstrating that its network-aware scheduling remains beneficial for optimization workloads even when fault-tolerant communication costs are included.

From a resource perspective, QAOA under QEC shows a balanced use of cross-rack and distilled in-rack EPR pairs, with EPR overhead comparable to other benchmarks. The absence of excessive additional waiting or retry overhead further indicates that QAOA’s interaction structure aligns well with SwitchQNet’s scheduling model under QEC. Note that these results are obtained from the implementation of 3-regular QAOA via a QEC scheme. As per initial testing, we saw some significant improvements by performing the same steps with the implementa-

Table 4.3: QEC Integration: performance of our compiler and the baseline for k-regular graph QAOA with surface code of distance 5.

Exp.	Benchmark- #alg.qubits	Baseline Latency	Ours Latency	Improv. Factor	#cross EPR	#in EPR	Distilled EPR	EPR Over.	Base Wait	Ours Wait	Addit. Wait	Retry Over.
Surface Code	MCT-480	145,202	35,859	4.05 \times	1,920	7,680	2,621	12.01%	0.00	2.40	2.40	1.00
	QFT-480	1,239,922	277,850	4.46 \times	15,360	3,840	19,889	21.81%	0.00	5.02	5.02	1.00
	Grover-480	18,482	2,718	6.80 \times	120	480	180	13.04%	0.00	1.31	1.31	1.00
	RCA-480	16,777	3,965	4.23 \times	180	720	249	12.15%	0.43	3.02	2.59	1.00
	QAOA-480	11,233	7,462	1.51 \times	690	210	648	12.00%	0.00	10.70	10.70	1.00

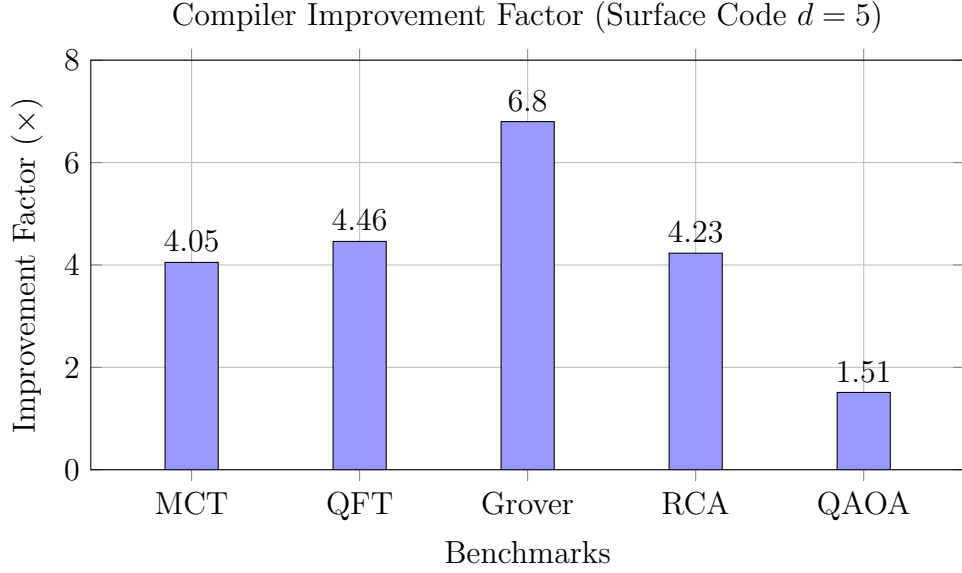


Figure 4.15: Visualization of the latency improvement factors across five benchmarks.

tion of random-QAOA (upto 4.7x), suggesting potential for further improvement. Overall, these results suggest that while 3-regular QAOA does not realize the same magnitude of latency improvement as more communication-intensive benchmarks, it remains compatible with QEC-enabled distributed execution and benefits from compiler-level optimizations. This reinforces the suitability of QAOA as both a realistic application workload and a diagnostic benchmark for studying the interaction between quantum error correction, network topology, and distributed quantum compilation.

Chapter 5

Conclusion and Future Works

This thesis examines the behavior of the Quantum Approximate Optimization Algorithm (QAOA) in a distributed quantum computing (DQC) setting using the SwitchQNet compiler framework. Motivated by the scalability limitations of monolithic quantum hardware and the growing relevance of modular and data-center-style quantum architectures, the work addressed the absence of representative optimization workloads as a key gap in existing evaluations of distributed quantum compilers. By integrating QAOA into SwitchQNet’s underlying routing and scheduling mechanisms, this study enabled a consistent analysis of how a structured, graph-based quantum combinatorial optimization algorithm interacts with network constraints across Clos, fat-tree, and spine-leaf topologies. The experimental results demonstrate that QAOA exhibits distinct performance trends compared to previously studied benchmark programs, particularly with respect to look-ahead depth, communication qubit availability, and interconnect latency (cross-rack switches). In several configurations, QAOA benefited more strongly from SwitchQNet’s latency-hiding strategies, maintaining relatively low absolute latency and stable improvement factors as network parameters were varied.

The sensitivity analyses further show that moderate increases in entanglement resource consumption, including additional EPR generation and distillation, may effectively mitigate communication delays without introducing prohibitive overhead. The QEC-integrated experiments further indicate that SwitchQNet’s network-aware scheduling remains effective under fault-tolerant assumptions, with QAOA exhibiting stable latency reductions despite the additional but small com-

munication and distillation overheads introduced by quantum error correction. These effects are observable even at the modest system scales explored in simulation, indicating that network-aware compilation decisions could meaningfully influence performance well before fault-tolerant regimes are reached. Collectively, the results support the central conclusion that distributed quantum algorithm performance must be evaluated through a joint consideration of algorithmic structure, network topology, and entanglement management, which will play an impactful role in the implementation of hardware architecture design and compilation strategies. By extending SwitchQNet with QAOA and clarifying the correspondence between its conceptual model and implementation, this work establishes a reproducible foundation for future studies of optimization algorithms in distributed quantum environments. More broadly, it positions QAOA as both a practical workload and an effective diagnostic tool for guiding the co-design of quantum algorithms, network architectures, and compilers in emerging quantum data center systems.

Availability

The code is updated under Creative Commons Attribution 4.0 International License derived from Version v1 of the original Zenodo source code at <https://zenodo.org/records/15377656>.

Our codebase is accessible on GitHub currently under https://github.com/starktech23/quantum_network_compiler-minimal_codes which may be subject to changes in the future.

References

- [1] Al-Fares, M., Loukissas, A., and Vahdat, A. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.* 38, 4 (Aug. 2008), 63–74.
- [2] Avin, C., and Schmid, S. Revolutionizing datacenter networks via reconfigurable topologies. *Commun. ACM* 68, 6 (June 2025), 44–53.
- [3] Barral, D., Cardama, F. J., Díaz-Camacho, G., Faílde, D., Llovo, I. F., Mussa-Juane, M., Vázquez-Pérez, J., Villasuso, J., Piñeiro, C., Costas, N., Pichel, J. C., Pena, T. F., and Gómez, A. Review of distributed quantum computing: From single qpu to high performance quantum computing. *Computer Science Review* 57 (Aug. 2025), 100747.
- [4] Bravyi, S., and Kitaev, A. Universal quantum computation with ideal-clifford gates and noisy ancillas. *Physical Review A* 71 (Feb 2005), 022316.
- [5] Briegel, H.-J., et al. Quantum repeaters: The role of imperfect local operations in quantum communication. *Physical Review Letters* 81 (1998), 5932–5935.
- [6] Burt, F., Chen, K.-C., and Leung, K. K. A multilevel framework for partitioning quantum circuits, 2025.
- [7] Cacciapuoti, A. S., Caleffi, M., Tafuri, F., Cataliotti, F. S., Gherardini, S., and Bianchi, G. Quantum internet: Networking challenges in distributed quantum computing. *IEEE Network* 34, 1 (Jan. 2020), 137–143.
- [8] Campbell, E. T., Terhal, B. M., and Vuillot, C. Roads towards fault-tolerant universal quantum computation. *Nature* 549, 7671 (Sep 2017), 172–179.

- [9] Campbell, K., Lawey, A., and Razavi, M. Quantum data centres: a simulation-based comparative noise analysis. *Quantum Science and Technology* 10, 1 (dec 2024), 015052.
- [10] Chakraborty, K., Rozpedek, F., Dahlberg, A., and Wehner, S. Distributed routing in a quantum internet, 2019.
- [11] Chang, X.-Y., Hou, P.-Y., Zhang, W.-G., Meng, X.-Q., Yu, Y.-F., Lu, Y.-N., Liu, Y.-Q., Qi, B.-X., Deng, D.-L., and Duan, L.-M. Hybrid entanglement and error correction in a scalable quantum network node, 2024.
- [12] Choi, H., Davis, M. G., Álvaro G. Iñesta, and Englund, D. R. Scalable quantum networks: Congestion-free hierarchical entanglement routing with error correction, 2023.
- [13] Cirac, J. I., Ekert, A. K., Huelga, S. F., and Macchiavello, C. Distributed quantum computation over noisy channels. *Phys. Rev. A* 59 (Jun 1999), 4249–4254.
- [14] Cisco Systems, I. Network-aware distributed quantum computing compiler. Tech. rep., Cisco Systems, Inc., October 2025. Cisco’s Software Technology for Quantum Computing Scale out.
- [15] Clos, C. A study of non-blocking switching networks. *The Bell System Technical Journal* 32, 2 (1953), 406–424.
- [16] Coopmans, T., Knegjens, R., Dahlberg, A., Maier, D., Nijsten, L., de Oliveira Filho, J., Papendrecht, M., Rabbie, J., Rozpedek, F., Skrzypczyk, M., Wubben, L., de Jong, W., Podareanu, D., Torres-Knoop, A., Elkouss, D., and Wehner, S. Netsquid, a network simulator for quantum information using discrete events. *Communications Physics* 4, 1 (July 2021).
- [17] Coppersmith, D. An approximate fourier transform useful in quantum factoring, 2002.
- [18] Dally, W. J., and Towles, B. P. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, March 2004.

- [19] Dangwal, S., Vittal, S., Seifert, L. M., Chong, F. T., and Ravi, G. S. Variational quantum algorithms in the era of early fault tolerance, 2025.
- [20] Devitt, S. J., Munro, W. J., and Nemoto, K. Quantum error correction for beginners. *Reports on Progress in Physics* 76, 7 (June 2013), 076001.
- [21] DiVincenzo, D. Looking back at the DiVincenzo criteria, feb 2018. Accessed: 2024-05-20.
- [22] DiVincenzo, D. P. Quantum computation. *Science* 270, 5234 (1995), 255–261.
- [23] DiVincenzo, D. P. The physical implementation of quantum computation. *Fortschritte der Physik* 48, 9–11 (Sept. 2000), 771–783.
- [24] Du, Z., Kan, S., Stein, S., Liang, Z., Li, A., and Mao, Y. Hardware-aware compilation for chip-to-chip coupler-connected modular quantum systems, 2025.
- [25] Farhi, E., Goldstone, J., and Gutmann, S. A quantum approximate optimization algorithm, 2014.
- [26] Farrington, N., Porter, G., Radhakrishnan, S., Bazzaz, H. H., Subramanya, V., Fainman, Y., Papen, G., and Vahdat, A. Helios: a hybrid electrical/optical switch architecture for modular data centers. *SIGCOMM Comput. Commun. Rev.* 40, 4 (Aug. 2010), 339–350.
- [27] Ferrari, D., Cacciapuoti, A. S., Amoretti, M., and Caleffi, M. Compiler design for distributed quantum computing. *IEEE Transactions on Quantum Engineering* 2 (2021), 1–20.
- [28] Fowler, A. G., et al. Surface codes: Towards practical large-scale quantum computation. *Physical Review A* 86 (2012), 032324.
- [29] Gottesman, D. The heisenberg representation of quantum computers, 1998.
- [30] Gottesman, D. Theory of fault-tolerant quantum computation. *Physical Review A* 57, 1 (Jan. 1998), 127–137.

- [31] Grover, L. K. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, Association for Computing Machinery (New York, NY, USA, 1996), 212–219.
- [32] Hajdušek, M., and Van Meter, R. Quantum communications, 2023.
- [33] Hong, C. K., Ou, Z. Y., and Mandel, L. Measurement of subpicosecond time intervals between two photons by interference. *Phys. Rev. Lett.* *59* (Nov 1987), 2044–2046.
- [34] Khait, I., Tham, E., Segal, D., and Brodutch, A. Variational quantum eigensolvers in the era of distributed quantum computers, 2023.
- [35] Knörzer, J., Liu, X., Schiffer, B. F., and Tura, J. Distributed quantum information processing: A review of recent progress, 2025.
- [36] Kwiat, P. G., Mattle, K., Weinfurter, H., Zeilinger, A., Sergienko, A. V., and Shih, Y. New high-intensity source of polarization-entangled photon pairs. *Phys. Rev. Lett.* *75* (Dec 1995), 4337–4341.
- [37] Labs, C. Q. Cisco quantum labs announces software that networks quantum computers together and enables new classical applications. Cisco Newsroom, October 2025.
- [38] Leiserson, C. E. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers C-34*, 10 (1985), 892–901.
- [39] Liu, J., Zang, A., Suchara, M., Zhong, T., and Hovland, P. D. Hardware-software co-design for distributed quantum computing, 2025.
- [40] Maslov, D., Dueck, G. W., Miller, D. M., and Negrevergne, C. Quantum circuit simplification and level compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* *27*, 3 (2008), 436–444.
- [41] Nielsen, M. A., and Chuang, I. L. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2011.

- [42] Niu, S., and Todri-Sanial, A. Enabling Multi-programming Mechanism for Quantum Computing in the NISQ Era. *Quantum* 7 (Feb. 2023), 925.
- [43] Ohkura, Y., Satoh, T., and Van Meter, R. Simultaneous execution of quantum circuits on current and near-future nisc systems. *IEEE Transactions on Quantum Engineering* 3 (2022), 1–10.
- [44] Parsons, N., and Calabretta, N. *Optical Switching for Data Center Networks*. Springer Handbooks (SHB). Springer, Germany, Oct. 2020, 795–825.
- [45] Piveteau, C., et al. Distributed quantum computing with compiler-optimized entanglement routing. *arXiv preprint* (2023).
- [46] Pouryoucef, S., Kaur, E., Shapourian, H., Towsley, D., Kompella, R., and Nejabati, R. Benchmarking quantum data center architectures: A performance and scalability perspective, 2026.
- [47] Preskill, J. Quantum computing in the nisc era and beyond. *Quantum* 2 (Aug. 2018), 79.
- [48] Sakuma, D., Tsuno, T., Shimizu, H., Kurosawa, Y., Friedrich, M. T., Teramoto, K., Taherkhani, A., Todd, A., Ueno, Y., Hajdušek, M., Ikuta, R., Meter, R. V., Sasaki, T., and Nagayama, S. Q-fly: An optical interconnect for modular quantum computers, 2025.
- [49] Satoh, R., Hajdusek, M., Benchasattabuse, N., Nagayama, S., Teramoto, K., Matsuo, T., Metwalli, S. A., Pathumsoot, P., Satoh, T., Suzuki, S., and Meter, R. V. Quisp: a quantum internet simulation package. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE (Sept. 2022), 353–364.
- [50] Schmidt, M., Kole, A., Wichette, L., Drechsler, R., Kirchner, F., and Mounzer, E. Exploration of design alternatives for reducing idle time in shor’s algorithm: A study on monolithic and distributed quantum systems, 2025.
- [51] Shapourian, H., Kaur, E., Sewell, T., Zhao, J., Kilzer, M., Kompella, R., and Nejabati, R. Quantum data center infrastructures: A scalable architectural design perspective, 2025.

- [52] Shor, P. W. Algorithms for quantum computation: Discrete logarithms and factoring. *Proceedings of the 35th Annual Symposium on Foundations of Computer Science* (1994), 124–134.
- [53] Stéphane, M. *A Wavelet Tour of Signal Processing (Third Edition)*, third edition ed. Academic Press, Boston, 2009.
- [54] Sundaram, R., Gupta, H., and Ramakrishnan, C. Dqc-qr: Distributing and routing quantum circuits with minimum execution time. *ACM Transactions on Quantum Computing* (July 2025). Just Accepted.
- [55] Tate, J., Beck, P., Clemens, P., Freitas, S., Gatz, J., Girola, M., Gmitter, J., Mueller, H., O’Hanlon, R., Para, V., et al. *IBM and Cisco: Together for a World Class Data Center*. IBM redbooks. IBM Redbooks, 2013.
- [56] Trotter, H. F. On the product of semi-groups of operators. *Proceedings of the American Mathematical Society* 10, 4 (1959), 545–551.
- [57] Van Meter, R. Quantum networking and internetworking. *IEEE Network* 26, 4 (2012), 59–64.
- [58] Van Meter, R. *Quantum Networking*, 1st ed. Wiley-IEEE Press, 2014.
- [59] Van Meter, R., and Devitt, S. J. The path to scalable distributed quantum computing. *Computer* 49, 9 (Sep. 2016), 31–42.
- [60] Van Meter, R., Nemoto, K., Munro, W. J., and Itoh, K. M. Distributed arithmetic on a quantum multicomputer. *SIGARCH Comput. Archit. News* 34, 2 (May 2006), 354–365.
- [61] Van Meter, R., Satoh, R., Benchasattabuse, N., Teramoto, K., Matsuo, T., Hajdusek, M., Satoh, T., Nagayama, S., and Suzuki, S. A quantum internet architecture. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE (Sept. 2022), 341–352.
- [62] Wootters, W. K., and Zurek, W. H. A single quantum cannot be cloned. *Nature* 299 (1982), 802–803.

- [63] Wu, A., Ding, Y., and Li, A. Qucomm: Optimizing collective communication for distributed quantum computing. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '23, Association for Computing Machinery (New York, NY, USA, 2023), 479–493.
- [64] Wu, A., Zhang, H., Li, G., Shabani, A., Xie, Y., and Ding, Y. Autocomm: A framework for enabling efficient communication in distributed quantum programs, 2022.
- [65] Wu, A., Zhang, H., Li, G., Shabani, A., Xie, Y., and Ding, Y. Autocomm: A framework for enabling efficient communication in distributed quantum programs. In *Proceedings of the 55th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '22, IEEE Press (2023), 1027–1041.
- [66] Wu, X., Kolar, A., Chung, J., Jin, D., Zhong, T., Kettimuthu, R., and Suchara, M. Sequence: a customizable discrete-event simulator of quantum networks. *Quantum Science and Technology* 6 (sep 2021).
- [67] Yimsiriwattana, A., and Jr, S. J. L. Generalized ghz states and distributed quantum computing, 2004.
- [68] Zhang, H., Xu, Y., Hu, H., Yin, K., Shapourian, H., Zhao, J., Kompella, R. R., Nejabati, R., and Ding, Y. Switchqnet: Optimizing distributed quantum computing for quantum data centers with switch networks. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ISCA '25, Association for Computing Machinery (New York, NY, USA, 2025), 1449–1463.

Appendix

A. Appendix: Quantum Gate Representations and Circuits

Table 5.1: Single-Qubit Pauli Gates

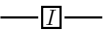

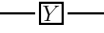
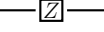


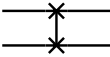
Gate Name	Circuit Symbol	Matrix Representation
Identity		$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
Pauli-X (NOT)		$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli-Y		$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Pauli-Z		$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

Table 5.2: Single-Qubit Clifford and Rotation Gates

Gate Name	Circuit Symbol	Matrix Representation
Hadamard	$\text{---}\boxed{H}\text{---}$	$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
Phase Gate (S)	$\text{---}\boxed{S}\text{---}$	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
$\pi/8$ Gate (T)	$\text{---}\boxed{T}\text{---}$	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$
Rotation R_x	$\text{---}\boxed{R_x(\theta)}\text{---}$	$\begin{pmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{pmatrix}$
Rotation R_y	$\text{---}\boxed{R_y(\theta)}\text{---}$	$\begin{pmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{pmatrix}$
Rotation R_z	$\text{---}\boxed{R_z(\theta)}\text{---}$	$\begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$

Table 5.3: Multi-Qubit Gates with Compressed Matrices

Gate Name	Circuit Symbol	Matrix Representation
CNOT		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
CZ		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$
SWAP		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

B. Appendix: Mathematical Framework of Quantum Information

Linear Operators and Dirac Notation

A linear operator A between vector spaces V and W is a function such that $A(\sum_i a_i |v_i\rangle) = \sum_i a_i A|v_i\rangle$.

- **Adjoint:** The adjoint (or Hermitian conjugate) of an operator A is denoted A^\dagger . In matrix form, this is the conjugate transpose: $(A_{ij})^\dagger = A_{ji}^*$.
- **Unitary Operators:** An operator U is unitary if $U^\dagger U = I$. Unitary operators preserve the inner product: $\langle\psi| U^\dagger U |\phi\rangle = \langle\psi|\phi\rangle$.
- **Hermitian Operators:** An operator A is Hermitian if $A = A^\dagger$. Observables in quantum mechanics are always represented by Hermitian operators.

The Tensor Product

The tensor product \otimes is used to combine Hilbert spaces to form larger systems. If \mathcal{H}_A and \mathcal{H}_B are spaces of dimension m and n , then $\mathcal{H}_A \otimes \mathcal{H}_B$ is a space of dimension mn .

$$(A \otimes B)(|a\rangle \otimes |b\rangle) = A|a\rangle \otimes B|b\rangle \quad (5.1)$$

The Partial Trace and Reduced Density Matrices

In quantum information theory, the partial trace is the fundamental linear algebra operation used to describe the state of a subsystem when it is part of a larger composite system. This is especially relevant in quantum communications when dealing with entangled pairs or environmental noise.

For a composite density matrix ρ_{AB} , the reduced density matrix ρ_A is:

$$\rho_A \equiv \text{Tr}_B(\rho_{AB}) \quad (5.2)$$

The partial trace is uniquely defined by its action on product states:

$$\text{Tr}_B(|a_1\rangle\langle a_2| \otimes |b_1\rangle\langle b_2|) = |a_1\rangle\langle a_2| \text{Tr}(|b_1\rangle\langle b_2|) = |a_1\rangle\langle a_2| \langle b_2|b_1\rangle \quad (5.3)$$

Table 5.4: QEC Integration: performance of our compiler and the baseline for random graph QAOA with surface code of distance 5.

Exp.	Benchmark- #alg.qubits	Baseline: Latency	Ours: Latency	Improv. Factor	#cross-rack EPR (15% infd.)	#in-rack EPR (5% infd.)	Ours: #distilled EPR (3.5% infd.)	EPR Over- head	Baseline: Wait Time	Ours: Wait Time	Addit. Wait Time	Retry Over- head
Surface Code	MCT-480	145,202	35,859	4.05 \times	1,920	7,680	2,621	12.01%	0.00	2.40	2.40	1.00
	QFT-480	1,239,922	277,850	4.46 \times	15,360	3,840	19,889	21.81%	0.00	5.02	5.02	1.00
	Grover-480	18,482	2,718	6.80 \times	120	480	180	13.04%	0.00	1.31	1.31	1.00
	RCA-480	16,777	3,965	4.23 \times	180	720	249	12.15%	0.43	3.02	2.59	1.00
	QAOA-480	642,182	136,617	4.70 \times	7,560	1,820	10,044	15.65%	0.00	4.93	4.93	1.00

Projective Measurements

A projective measurement is described by an observable M , a Hermitian operator on the state space. The observable has a spectral decomposition:

$$M = \sum_m m P_m \quad (5.4)$$

where P_m is the projector onto the eigenspace of M with eigenvalue m . The probability of outcome m for state ρ is:

$$p(m) = \text{Tr}(P_m \rho) \quad (5.5)$$

C. Additional Record of Results

```

{'seed': 1, 'test_mapper_type': 'baseline', 'test_router_type': '
hand', 'test_scheduler_type': 'our', 'test_program_type': '
qaoa', 'test_conn_G_type': 'clos', 'rack_num': 4, '
qpu_per_rack': 4, 'qbit_per_qpu': 30, '
outer_router_edge_weight': 8, 'commqbit_num': 2, '
rack_router_bsm_num': 4, 'cache_size': 10, 'cache_reserve_size
': 2, 'scheduling_reserve_size': 3, 'schedu_depth': 10, '
inter_time': 1, 'switch_time': 10, 'exter_time': 100, '
split_mul': 2, 'look_ahead': True, 'join_map': True, 'split':
True, 'prefix': 'QAOA_single_run', 'father_dir': './
results_qaoa', 'repeat_num': 1, 'auto_retry_length': True, '
retry_length': 50, 'shoot_gap': 200, 'check_CAT_sTP': True, '
check_dTP': True, 'check_split_num': True, 'qec_program':
False, 'code_dist': 5}

routing_result: cross EPR:1096 inter EPR:244
DAG: cross EPR:1096 inter EPR:244
Scheduling start
Scheduling time:13.029343128204346
rack_num 4 qpu_per_rack 4 qbit_per_qpu 30
time_schedule 10287 cross_pair 1096 in_pair 244 post_in_pair 776
cross_wait_time 182224 in_wait_time 162774
retry_num 0 total_step 10287
thruput 0.1302614950908914 avg_wait_time 163.04253308128546
    avg_retry_overhead 1.0

METRICS_JSON:
{ time_schedule : 10287, cross_pair : 1096, in_pair : 244,
  post_in_pair : 776, cross_wait_time : 182224, in_wait_time :
    162774, retry_num : 0, total_step : 10287, throughput :
    0.1302614950908914, cross_EPR_cache_time : null,
  inter_EPR_cache_time : null, distill_pair : null,
  schedule_result_list_len : 2892}

```

Listing 5.1: SwitchQNet QAOA raw execution log output.