SwitchQNet: Optimizing Distributed Quantum Computing for Quantum Data Centers with Switch Networks

Hezi Zhang
hezi@ucsd.edu
University of California
San Diego, USA

Yiran Xu
yix072@ucsd.edu
University of California
San Diego, USA

Haotian Hu
hah041@ucsd.edu
University of California
San Diego, USA

Keyi Yin
keyin@ucsd.edu
University of California
San Diego, USA

Hassan Shapourian
hshapour@cisco.com
Cisco Quantum Lab
San Jose, USA

Jiapeng Zhao
penzhao2@cisco.com
Cisco Quantum Lab
San Jose, USA

Ramana Rao Kompella
rkompell@cisco.com
Cisco Quantum Lab
San Jose, USA

Reza Nejabati
rnejabat@cisco.com
Cisco Quantum Lab
San Jose, USA

Yufei Ding
yufeiding@ucsd.edu
University of California
San Diego, USA

Abstract

Distributed Quantum Computing (DQC) provides a scalable architecture by interconnecting multiple quantum processor units (QPUs). Among various DQC implementations, quantum data centers (QDCs) — where QPUs in different racks are connected through reconfigurable optical switch networks — are becoming feasible in the near term. However, the latency of cross-rack communications and dynamic switch reconfigurations poses unique challenges to communications in QDCs, significantly increasing the overall latency, thereby also reducing the overall fidelity. In this paper, we address these challenges by introducing a novel compiler that optimizes scheduling of communications across the program and network layers. Our evaluation shows that it reduces the overall latency by 8.02× over prior approaches with a small overhead and can be integrated with quantum error correction (QEC) to facilitate fault-tolerant quantum computing (FTQC). We have open-sourced our codes at https://zenodo.org/records/15377656.

CCS Concepts

• **Computer systems organization** → **Architectures**; **Other architectures**; **Quantum computing**;

Keywords

Distributed Quantum Computing (DQC), Quantum Data Center (QDC), Compilation

ACM Reference Format:
Hezi Zhang, Yiran Xu, Haotian Hu, Keyi Yin, Hassan Shapourian, Jiapeng Zhao, Ramana Rao Kompella, Reza Nejabati, and Yufei Ding. 2025. 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), June 21–25, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3695053.3731046

1 Introduction

Distributed quantum computing (DQC) [3, 18, 19, 24, 28] has emerged as a promising approach to address scalability challenges in quantum computing. By enabling quantum communication between multiple quantum processing units (QPUs), DQC extends the capabilities of single-chip systems to multi-node architectures. Among various implementations of DQC, quantum data centers (QDCs) [5, 7, 41, 45, 46, 51] have gained significant attention from industry as a practical near-term solution. Unlike early visions of DQC that focused on long-distance quantum communication across cities, QDCs aim to connect multiple QPUs within a local range of a single room or facility [30, 34, 49, 50, 59]. Their relatively short communication distances and controlled environment mitigate photon loss and various noise, making them well-suited for near-term computational needs.

Recently, an emerging scalable architecture has been proposed [59] for local QDCs, aligning with near-term hardware capabilities. As shown in Fig. 1, it connects multiple racks of QPUs via a dynamic optical network with reconfigurable optical switches, which, unlike long-range DQC settings, requires no memory on each switch. Cross-rack switches (core switches), depicted in orange in Fig. 1, can leverage existing classical optical switch technology, while only a small number of specialized quantum switches, depicted in blue, need to be deployed on top of racks (ToR switches) to enable EPR pair generation for quantum communication. This significantly enhances the practicality and cost-efficiency of QDCs, making them a more viable DQC infrastructure in the near future.

The primary bottleneck in DQC lies in inter-QPU communication, which is significantly slower and more error-prone than QPU computation. Existing software solutions [29, 31, 37, 70, 71] aim to reduce latency and improve fidelity by minimizing the number of EPR pairs required by communication. However, these approaches

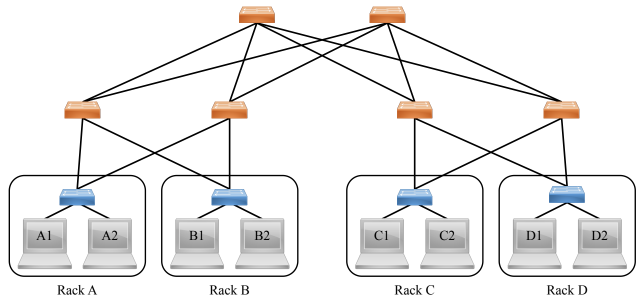

Figure 1: QDC with QPUs connected by a switch network.

show substantial inefficiencies when applied to the QDC architecture due to two key factors. **First**, the distinction between classical core switches and quantum ToR switches requires converters to facilitate their interaction. This added complexity results in a higher latency (∼10 ms) for cross-rack EPR pair generation than in-rack EPR pair generation (∼0.1 ms). **Second**, each switch reconfiguration introduces an additional latency of ∼ 1 ms, due to the need to suppress photon loss. Therefore, frequent switch reconfigurations can significantly extend communication latency, especially for in-rack communications.

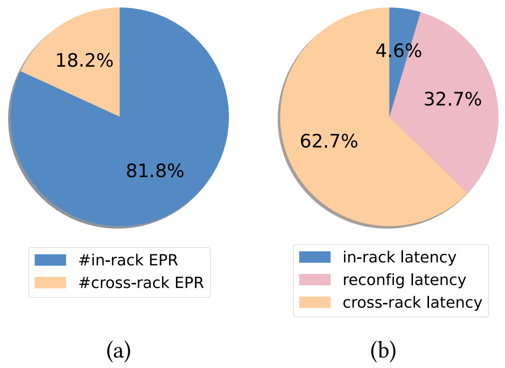

Figure 2: (a) Number percentages of in-rack and cross-rack EPR pairs. (b) Latency percentages of in-rack EPR pair generation, reconfiguration and cross-rack EPR pair generation.

In Fig. 2, we illustrate the communication budget by profiling the quantum workload across different operations. Specifically, we first count the required number of in-rack and cross-rack EPR pairs, then isolate the contributions of different operations to the overall latency by: (1) setting the latency of both in-rack communication and reconfiguration to 0, attributing all latency to cross-rack communication; (2) setting only the latency of in-rack communication to 0, attributing the latency difference between the two cases to reconfiguration. As shown in Fig. 2, while cross-rack EPR pairs constitute only 18.2% of the total required EPR pairs, they account for 62.7% of the overall latency, with reconfigurations contributing an additional 32.7%. This implies that cross-rack communication and frequent switch reconfigurations substantially increase latency, which would also degrade the overall fidelity due to the increased qubit decoherence over time.

To address these challenges in QDCs, a co-optimization across the program layer and the network layer is required in terms of EPR pair scheduling. Existing DQC systems, such as long-range repeater networks [8, 17, 52], decouple the scheduling of EPR pairs and the generation of them into a program layer and a network layer, with the scheduler merely considering the program-layer communication demands and the network layer being responsible for generating the scheduled EPR pairs according to current bandwidth and resource availability. However, given the known computation tasks for QDCs, the compiler can schedule EPR pair generations by considering both the upcoming communication demands of the quantum program and the bandwidth and resource availability on switches and QPUs. In this way, a larger optimization space can be enabled for hiding the latencies of cross-rack communication and switch reconfiguration.

At a high level, this co-optimization is achievable by leveraging two key quantum features. First, in quantum systems, preparation of EPR pairs can be decoupled from actual communications, as they do not carry any program data. Even if the switches in QDCs are memoryless, these EPR pairs can be stored temporarily on QPUs by allocating a portion of computation qubits as buffer [70]. This storage allows for a flexible look-ahead scheduling of EPR pairs, which allows for optimized timing of EPR generation when combining an analysis of program patterns with a management of buffer resources. Second, an EPR pair between source s and destination d can be generated by merging multiple pre-existing EPR pairs along a path between s and d, referred to as *entanglement swapping*. Unlike that in repeater networks, entanglement swapping on QPUs can be performed fast and deterministically by gates and measurements. With an awareness of bandwidth and buffer availability, this can enable a more efficient network exploitation through a flexible strategy that generates EPR pairs by part.

However, leveraging these features in a compiler is highly non-trivial. **First**, they can introduce fidelity overheads in two ways. (1) The storage of EPR pairs in buffer can bring a risk of decoherence, causing the fidelity of EPR pairs to decrease over time. (2) The strategy of generating EPR pairs by part necessitates generation of additional EPR pairs, which can compromise the overall fidelity as EPR generations are more error-prone than gates on QPUs. These fidelity overheads impose stricter constraints on QDCs than classical data centers (DCs), preventing a straightforward application of pre-fetching techniques in classical DCs. **Second**, the flexibilities in scheduling can lead to complications such as deadlocks (i.e., generations of EPR pairs need to wait for each other's consumption) and buffer congestion (i.e., required buffer occupied by data qubits or other EPR pairs). These need to be eliminated in the compilation stage to ensure successful program execution.

To this end, we present an efficient compiler to strategically leverage those quantum features, striking a balance between latency reduction and fidelity overhead, while eliminating the risks of deadlock and buffer congestion. The compiler analyzes the programs by identifying gate dependencies, looking ahead into near-future gates and extracting their requirements for EPR pairs. With an awareness of the network heterogeneity and the availability of bandwidth and buffer resources, these EPR pairs are then scheduled with the following two optimizations.

First, it **parallelizes cross-rack EPR pair generations** by splitting them into parts, with each split resulting in a new cross-rack pair and some additional in-rack pairs (referred to as *post-split EPR pairs*). In this way, even if the source or destination QPU of an EPR pair is busy, a substitute cross-rack EPR pair can be generated between the source and destination racks through a same-rack QPU with the busy QPU. Once the busy QPU becomes available, it can communicate with that same-rack QPU by generating an additional in-rack EPR pair promptly, thereby completing the generation of the original cross-rack EPR pair through an entanglement swapping. **To mitigate the fidelity overhead** brought by additional in-rack EPR pairs, we prepare multiple copies of each post-split in-rack EPR pair and enhance its fidelity by entanglement distillation [11, 38] using the extra copies.

Second, it further **hides the latency of the additional in-rack EPR pairs**, either from split or for distillation, by a collective generation of them, so that they are prevented from becoming a new communication bottleneck. That is, given that the latency of reconfiguration is much longer than that of in-rack EPR pair generation, we can collect in-rack EPR pairs between the same QPUs and generate them together when the channel between the QPUs is available. In this way, we reduce the average latency of in-rack EPR pair generation by avoiding frequent reconfiguration. The collection of in-rack EPR pairs is guided by the look-ahead program analysis of our compiler. This program-aware guidance reduces the wait time of both in-rack and cross-rack EPR pairs before their usage, thereby **minimizing the fidelity overhead** brought by the storage of prepared EPR pairs.

To **prevent potential deadlocks and buffer congestion** caused by the flexible scheduling, we propose a resource managing approach that is aware of bandwidth and buffer status. It consists of several rules for scheduling and splitting EPR pairs, along with a retry mechanism for EPR scheduling. These rules mitigate the risk of deadlock and buffer congestion by adapting to current bandwidth availability and ensuring enough buffer size for future EPR pairs, leaving only rare cases that are hard to avoid. When those rare cases happen, we resolve them by incorporating an efficient retry mechanism that downgrades the scheduling strategy to a more conservative one which ensures a deadlock-free and congestion-free scheduling.

Our contributions in this paper can be summarized as the following:

- We propose a compiler to overcome communication challenges in QDCs through a flexible scheduling that is aware of both program patterns and network resources, while preventing potential deadlock and buffer congestion brought by the flexibility.
- We improve the communication efficiency by overlapping the latencies of cross-rack communications, at the cost of incurring additional in-rack communications, with the latency of in-rack communications further reduced by a collective generation.
- We minimize the fidelity overhead by incorporating entanglement distillation for the additional in-rack communications and reducing the wait time of prepared EPR pairs through a program-aware guidance.
- Through a comprehensive simulation with practical hardware parameters, we demonstrate an 8.02× reduction in communication latency over previous approaches while maintaining a low fidelity overhead.

2 Background and Related Work

2.1 Quantum Communication Protocols

Communication between different QPUs can be achieved through EPR pairs (wavy lines in Fig. 3) established between the QPUs with different protocols. The Cat protocol [72] (Fig. 3(a)) can realize a block of remote control gates sharing the same control qubit without transferring data from a QPU to another. In contrast, the teleportation (TP) protocol (Fig. 3(b)) allows any pattern of remote gates by transferring a data qubit from a QPU to another through teleportation (from q_0 to q'_c in Fig. 3(b)).

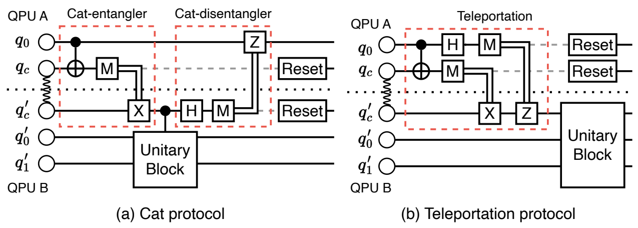

Figure 3: (a) Cat protocol and (b) TP protocol for realizing inter-QPU gates with inter-QPU EPR pairs.

2.2 Quantum Data Center (QDC)

QDC networks As illustrated by Fig. 4, a QDC connects QPUs through a local optical network consisting of optical fibers and reconfigurable optical switches. Edges in the network present possible physical channels connecting nodes (i.e., switches and QPUs) through optical fibers. Channel capacity can be increased via multiplexing, e.g., multiple optical fibers or multiple wavelengths through a single fiber, represented by an edge weight $w > 1$. For instance, Fig. 4(b)(c) demonstrate a case of edge weight 2 where each pair of nodes are connected by 2 fibers.

Optical switch The reconfiguration latency of switches ranges from multiple of 100 ns to 100 ms, with the latency increasing with the number of ports [63]. Moreover, a faster switch reconfiguration also leads to an increased photon loss [63]. For switches with about 16×16 ports (or 32 ports), the reconfiguration latency is typically ~ 1 ms with a photon loss rate around 1 dB [65], while the latency of larger switches 1024×1024 ports commercially available are typically around 100 - 200ms. In this paper, we will adopt the value of 1 ms if not stated otherwise. Each ToR switch can be equipped with Bell-state measurement (BSM) devices to facilitate EPR pair generation. These BSM devices can be shared among QPUs in the same rack by ToR reconfiguration, as shown in Fig. 4(c).

EPR pair generation As depicted in Fig. 4(d), the generation of EPR pairs requires dedicated communication qubits on each QPU with spin-photon interface between the stationary communication

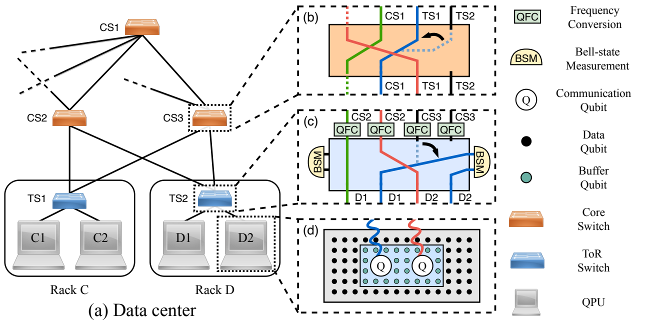

Figure 4: (a) A QDC network with (b) classical core switches and (c) quantum ToR switches equipped with a QFC on each outward port and some BSMs. (d) Each QPU has two dedicated communication qubits and many computation qubits, with the computation qubits in the light blue box indicating a buffer initially allocated on the QPU.

qubits and flying photonic qubits. Fig. 5 illustrates the EPR generation protocol [14, 51], where optical switches are eliminated for simplicity.

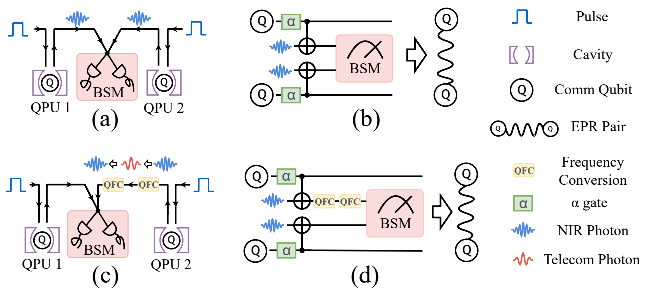

Figure 5: In-rack EPR pair generation: physical process (a) and corresponding circuit (b). Cross-rack EPR pair generation: physical process (c) and corresponding circuit (d).

For in-rack communication, as shown Fig. 5(a), communication qubits are prepared in the superposition state $\sqrt{\alpha}\,|\uparrow\rangle + \sqrt{1-\alpha}\,|\downarrow\rangle$ and then driven the spin to emit a photon via spontaneous emission, leading to an entangled spin-photon state. Next, the two photonic qubits are directed towards a beam splitter and are eventually measured by the two single-photon detectors, as depicted in the pink box. We only post-select the single detection events to project the spin-spin state into $|\phi_+\rangle = (|\uparrow\downarrow\rangle + |\uparrow\downarrow\rangle)/\sqrt{2}$. This effectively implements a BSM with $2\alpha(1-\alpha)$ success probability. Fig. 5(b) shows the equivalent quantum circuit, where the emission process effectively acts as a CNOT gate between the spin and photonic qubit. For cross-rack communication, the protocol is similar but requires quantum frequency converters (QFCs), as shown in Fig. 5(c)(d). This is because our ToR switches and BSM devices operate at the near-infrared (NIR) where commercially available optical fiber or switches are not optimized (in terms of the photon loss rate). To extend the communication range to other racks, we convert the NIR photon into telecom regime and back via bidirectional QFCs [12, 13, 40, 58, 68].

EPR rate and fidelity We now estimate the rates and fidelity for in-rack and cross-rack EPR pair generation with some hardware parameters available from the literature. We note that our EPR generation process is probabilistic due to the probabilistic BSM and due to photon loss during transmission from spin to fiber and as they travel through the switches. Hence, the success probability is found to be $p = 2\alpha\eta$ where α is the initial state parameter and $\eta \ll 1$ denotes the overall photon transmission rate (i.e., overall photon loss rate is $1-\eta$). As a result, we consider a repeat-until-success protocol which keeps trying to generate an EPR pair until getting a positive signal in the BSM. Let τ_0 be the operation time for each attempt, then the average time for a successful EPR pair generation is $\tau = \tau_0/p$. Considering hardware parameters [51, 58, 64, 76] $\alpha = 0.05$, $\eta = 0.1$ (i.e., 10 dB loss), and $\tau_0^{-1} \sim 1$ MHz, we obtain $\tau_{\text{ToR}} = 0.1$ ms for in-rack EPR pair generation. For the cross-rack communication, the EPR generation rate is reduced by a factor of 100 (i.e., 20 dB additional loss) as the transmission rate is further reduced due to the signal attenuation in the second NIR switch, and two QFC devices, leading to $\tau_{\text{inter}} = 10$ ms.

For EPR fidelity, because of false positive signals, we obtain a noisy Bell state in the form of $\hat{\rho} = (1-\alpha)\,|\phi_+\rangle\langle\phi_+| + \alpha\,|\uparrow\uparrow\rangle\langle\uparrow\uparrow|$, and the resulting fidelity is then given by $F = 1-\alpha$. With $\alpha = 0.05$, we obtain $F_{\text{ToR}} = 0.95$ for in-rack EPR pair generation. We further assume that the fidelity is dropped by another 10% due to conversion infidelity in QFCs [40, 64, 76], resulting in a lower fidelity $F_{\text{inter}} = 0.85$ for cross-rack EPR pair generation.

2.3 Related Work

DQC compilation Over the last few years, various compilers have been developed for DQC. Some previous work optimizes qubit placement [4, 9, 10, 25–27, 77] or optimizes communications when routing program qubits among QPUs [29, 31, 37, 70, 71]. These approaches are orthogonal to our compiler and can be combined with our optimizations. Some compilers [56, 66] have also been proposed to implement DQC without inter-QPU communications by cutting large circuits into smaller ones and running them on different devices, yet they require non-scalable classical post-processing. Furthermore, recent work [39] has proposed an efficient implementation of surface code for DQC. Our work can be considered as a higher-level optimization than it, as each of our qubits can be considered as either a physical qubit or a logical qubit.

Long-range quantum repeater networks Long-range quantum communication leverages quantum repeaters to mitigate significant photon loss over extended distances, requiring specialized memory qubits and quantum devices at each repeater for probabilistic entanglement swapping. Previous work have proposed architecture designs to ensure scalability of repeater networks for random workload (e.g., congestion-free on repeater memories, size-independent threshold for error-corrected entanglement, distance-independent communication rates [21, 22, 54, 55]). Moreover, recent work has proposed making quantum repeater networks compatible with classical networks by designed protocols on the control plane [73]. However, these repeater networks are designed for long-range quantum communication rather than local-range efficient quantum computing. In contrast, QDCs operate within shorter distances, eliminating the need for repeaters, and can leverage

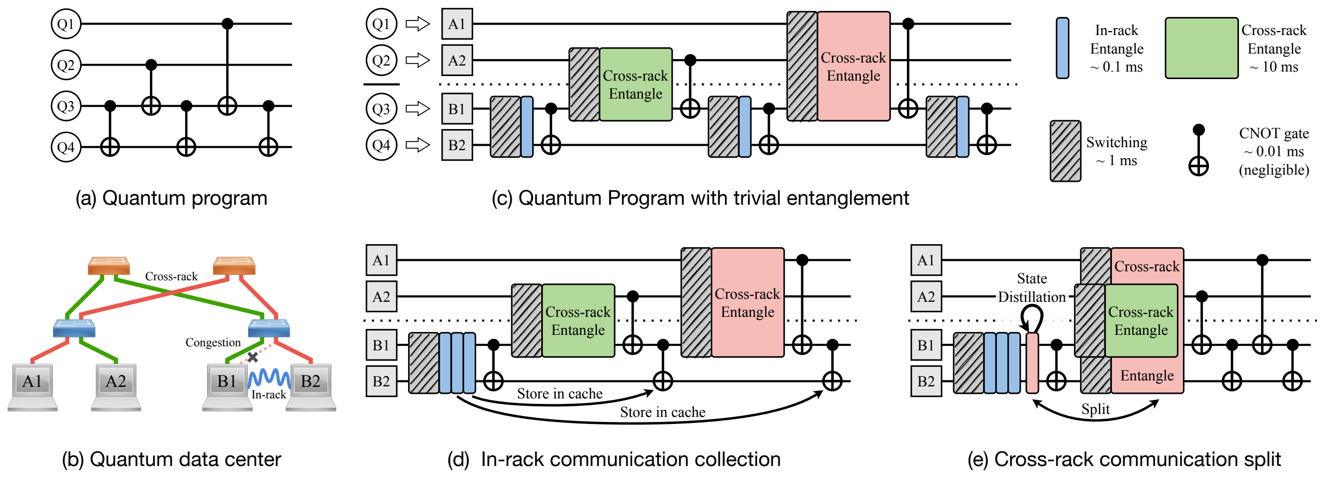
Q1
Q2
Q3
Q4
(a) Quantum program
A1
A2
B1
B2
Cross-rack Entangle
(c) Quantum Program with trivial entanglement
In-rack Entangle ~ 0.1 ms
Cross-rack Entangle ~ 10 ms
Switching ~ 1 ms
CNOT gate ~ 0.01 ms (negligible)
Cross-rack
Congestion
In-rack
(b) Quantum data center
Store in cache
Store in cache
(d) In-rack communication collection
State Distillation
Cross-rack
Entangle
Split
(e) Cross-rack communication split

Figure 6: Deployment of a quantum program (a) on to a quantum data center (b). The in-rack communication time is reduced from 3.3 ms (c) to 1.3 ms (d) by a collection. The overall communication time is reduced from 23.3 ms (d) to 12.4 ms (e) by splitting a congested cross-rack communication into a non-congested cross-rack communication and an in-rack communication, where the in-rack one can be distilled at a low cost to enhance its fidelity.

off-the-shelf optical switches with minimal quantum devices. This makes them better suited for meeting computational needs with near-term hardware capabilities. For repeater networks, EPR distribution protocols [42, 54, 60, 74, 75] have been designed to maximize EPR throughput by strategically selecting successful links among memory qubits. However, these protocols are tailored for repeater networks and reduce to simple shortest-path searches in memoryless QDCs, thus cannot address the unique challenges of QDCs effectively.

Classical DC While classical and quantum data centers (DCs) [1, 67] both exhibit hierarchical network structures with higher cost for cross-rack communication than in-rack communication, their underlying communication models are fundamentally different in terms of data replicability and fidelity sensitivity. These differences impose stricter constraints on QDCs and necessitate more sophisticated compilation techniques.

In classical DCs, data can be freely replicated, stored indefinitely, and retransmitted without fidelity degradation. Consequently, techniques such as pre-transmission, dynamic rerouting, and flexible buffer management are highly effective. In contrast, communication in QDCs rely on EPR pairs that can be stored for only limited time, and consumed only once, rendering classical approaches ineffective. Furthermore, while classical DCs can reduce cross-rack communication by rerouting data through in-rack paths with negligible impact, QDCs require additional in-rack EPR pairs when splitting communication paths. Each additional in-rack EPR generation introduces fidelity overhead, creating a fundamental trade-off between latency reduction and fidelity maintenance.

Our compilation framework addresses these quantum-specific constraints by enabling collective and parallel EPR generation, applying entanglement distillation to mitigate fidelity degradation, ensuring deadlock- and congestion-free scheduling in the compilation stage to guarantee program progress. These techniques have no direct classical analogue and are essential to achieving low-latency, high-fidelity communication in QDCs.

3 Motivation

This section introduces the optimizations in our compiler with a motivating example, including collective generation of in-rack EPR pairs and parallelized generation of cross-rack EPR pairs. To facilitate these two optimizations, techniques for resolving deadlock and buffer congestion are also required, which will be introduced with more technical details in the next section.

Fig. 6(c) demonstrates an example of scheduling EPR pair generations without a look-ahead analysis of the program pattern. It deploys a circuit in Fig. 6(a) to the QDC in Fig. 6(b), scheduling EPR pair generations (depicted as rectangles in Fig. 6(c)) right before they are needed by inter-QPU CNOT gates, with switch reconfiguration represented by the shaded rectangles in Fig. 6(c). Specifically, each line in Fig. 6(c) represents a qubit on a different QPU, with the three gates between QPU B_1 and B_2 requiring in-rack EPR pairs and the other two gates requiring cross-rack EPR pairs according to Fig. 6(b). Adopting latencies of in_rack = 0.1 ms, reconfig = 1 ms and cross_rack = 10 ms, neglecting the much shorter gate execution time, the execution of the 5 remote gates requires 25.3 ms in total (Fig. 6(c)). As will be seen, this can be reduced to 12.4 ms with the following optimizations (Fig. 6(d)(e)).

The first optimization is the collective generation of near-future in-rack EPR pairs, which reduces their average latency by avoiding frequent switch reconfigurations. Through a look-ahead into the program, the compiler finds that three in-rack pairs between QPU B_1 and B_2 are needed in the near future (blue rectangles), which takes 0.3 ms only. However, the switch reconfigurations for enabling the optical channel for three times takes 3 ms (shaded rectangles before the blue ones). To reduce this overhead, it schedules the generations of these three in-rack EPR pairs collectively, as shown by the consecutive blue rectangles in Fig. 6(d). These generated EPR pairs are then stored temporarily in the buffer before usage. With this collection, the time for generating the three in-rack EPR pairs is reduced from 3.3 ms to 1.3 ms, with the overall communication time reduced from 25.3 ms to 23.3 ms.

The second optimization is the parallelization of near-future cross-rack EPR pairs, which maximizes the utilization of network bandwidth by splitting congested EPR pairs into non-congested ones. In Fig. 6(d), generations of the two cross-rack EPR pair (A_2, B_1) and (A_1, B_1) are sequential, as QPU B_1 can communicate with only one other QPU at a time (assuming edge weight = 1). That is, in Fig. 6(b), the green path conflicts with the red dashed path. To parallelize them, we can split the congested EPR pair (A_1, B_1) by allowing B_1 to borrow bandwidth from B_2, a QPU in the same rack as B_1. Specifically, the cross-rack EPR pair (A_1, B_1) is split into a new cross-rack EPR pair (A_1, B_2) and an additional in-rack EPR pair (B_1, B_2), followed by an entanglement swapping between these two pairs once they are both prepared. In this way, we can generate the new cross-rack EPR pair (A_1, B_2) in parallel with the originally conflicted (A_2, B_1) and store it in the buffer, while scheduling an additional in-rack EPR pair (B_1, B_2) in the collective generation, as shown in Fig. 6(d). This further reduces the overall latency from 23.3 ms (Fig. 6(d)) to 12.4 ms (Fig. 6(e)).

As the split of cross-rack communication incurs additional in-rack EPR pairs, it poses a risk of reducing the overall fidelity of quantum computing. To mitigate this, we enhance the fidelity of these additional in-rack EPR pairs by scheduling multiple copies of each and implementing entanglement distillation among them, as shown in Fig. 6(e). This can be achieved with a low cost as the multiple copies can be scheduled collectively with other in-rack EPR pairs. Given that in-rack EPR pairs have a 95% fidelity, a distillation by two copies results in an EPR pair of > 96.5% fidelity (where we approximated our input states as Werner states with 93.6% success probability [11, 38]). This fidelity can be further enhanced by using more copies. Note that cross-rack EPR pairs and original (non-split) in-rack EPR pairs can also be distilled upon requests. This can be easily accommodated by our framework, which is equivalent to an increased latency of EPR pair generation.

4 Framework Design

4.1 Preprocessing

Our compiler can be used in combination with previous compilers [70, 71] that reduce the required EPR pairs. Considering the dynamic reconfigurability of QDCs, these compilers for static network topology should be applied by assuming a full connection between QPUs. In our experiments of Section 5, we will combine with the buffer-aware compilation technique in [70]. Its output is a list of required EPR pairs, with corresponding QPUs and communication protocols specified (i.e., Cat vs. TP, see Section 2).

To leverage its minimization of EPR pairs, we transform its output to serve as input to our compiler through preprocessing. Specifically, we convert this EPR list to a directed acyclic graph (DAG) by imposing dependencies among them, with each node in DAG representing an EPR pair, and each directed edge representing a dependency. Specifically, whenever the involved QPUs of two EPR pairs overlap, we add an edge from the earlier one in the output list to the later one.

This DAG needs to be utilized appropriately by our compiler, as the imposed dependency can deviate from the real dependency due to the following reasons. First, even if two EPR pairs are independent according to the DAG, they may not be able to be scheduled concurrently due to bandwidth contention. Second, even if two EPR pairs are dependent according to the DAG, they may be able to be scheduled concurrently, as the overlapped QPUs between them may have multiple communication qubits to generate both pairs. However, these inaccurate dependencies can be used as a rough guidance in our scheduling process, with the above cases handled automatically by our compiler.

4.2 EPR Scheduling

This subsection introduces the conditions for EPR pair scheduling, with in-rack collection introduced in the explanation of in-rack vs. cross-rack EPR pairs. Due to the limited communication qubits and network bandwidth, at each time slice it depends on four conditions to decide whether an EPR between two QPUs can be scheduled. We will first list them as below and then explain.

Scheduling Conditions:

(1) **Hard:** available communication qubits on both QPUs
(2) **Hard:** available BSMs on the rack of either QPU
(3) **Hard:** available optical channel between the QPUs
(4) **Soft:** buffer_size + avail_comm ≥ threshold · not_in_front_layer (threshold ≥ #comm_qubits per QPU) on both QPUs

Hard vs. soft conditions The first three conditions are mandatory resource requirements for EPR generation. By checking these conditions for a certain EPR pair, we ensure that this EPR pair can be supplied at the moment by the network. That is, an EPR pair between QPU A and B can be generated only if there are available communication qubits on both A and B, there is an available BSM on the ToR switch of either A's rack or B's rack, and there is an available path in the network between A and B.

The fourth condition is optional, depending on whether the EPR pair is in the front layer of the DAG (i.e., not_in_front_layer = True / False), as explained later. It aims to prioritize the buffer utilization of front-layer EPR pairs to mitigate buffer congestion, which not only improves the overall compilation performance, but also reduces the incurrence of retries that will be introduced in Section 4.5.

In-rack vs. cross-rack Due to our collective in-rack EPR pair generation strategy, the third condition can be different when applied to in-rack and cross-rack EPR pairs. For a cross-rack EPR pair between QPU A and B, this condition requires an available optical channel between A and B that is not occupied by any other communication. For an in-rack EPR pair, this condition is checked in two steps. We first check whether this EPR pair can share an occupied channel with other in-rack EPR pairs between A and B. If so, we combine it with those EPR pairs by scheduling it right after them. If not, we then check if there is an available optical channel that is not occupied by any other communication.

Front layer vs. non-front layer The fourth condition only applies to EPR pairs that are not in the front layer of the DAG. For EPR pairs in the front layer, we allow their generations as long as the three mandatory conditions are satisfied, since they are

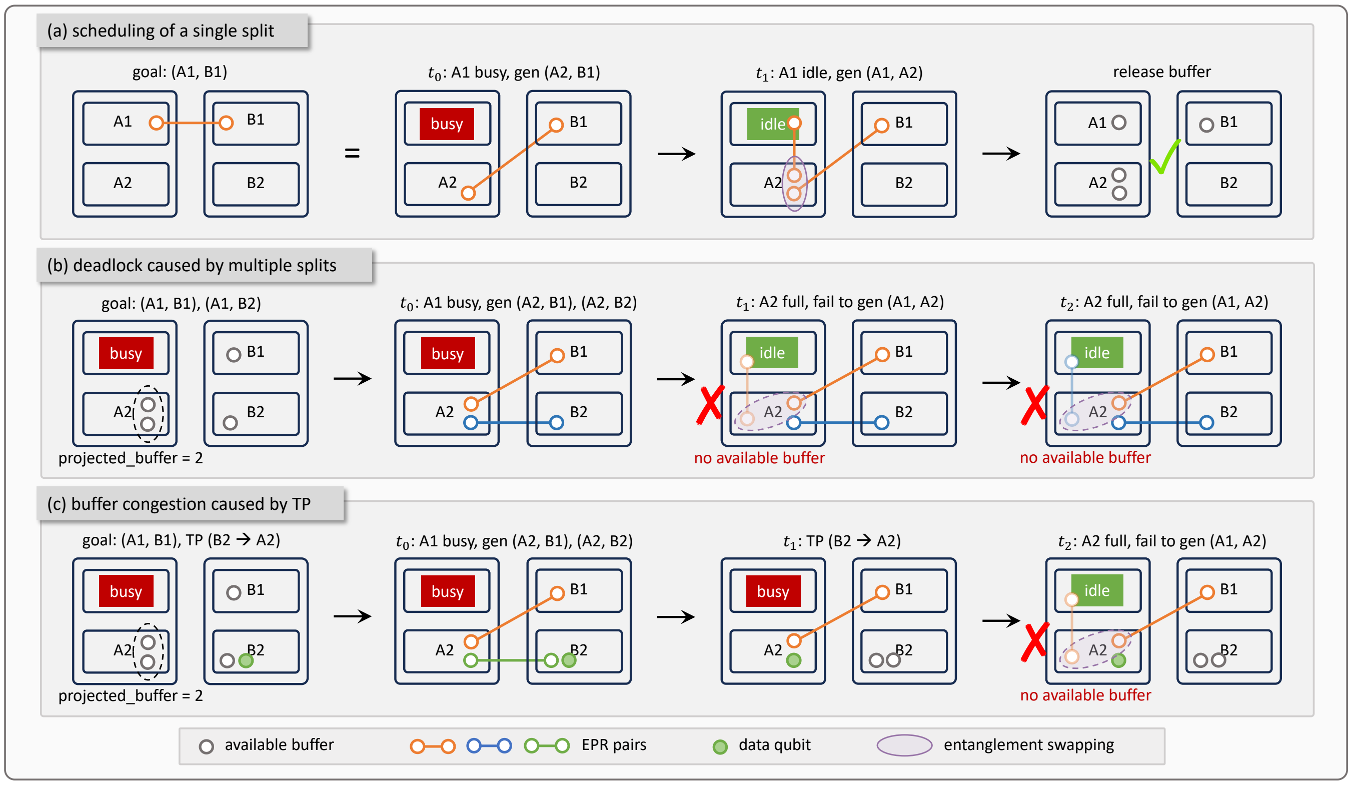
(a) scheduling of a single split
goal: (A1, B1)
t_0: A1 busy, gen (A2, B1)
t_1: A1 idle, gen (A1, A2)
release buffer
(b) deadlock caused by multiple splits
goal: (A1, B1), (A1, B2)
t_0: A1 busy, gen (A2, B1), (A2, B2)
t_1: A2 full, fail to gen (A1, A2)
t_2: A2 full, fail to gen (A1, A2)
projected_buffer = 2
no available buffer
(c) buffer congestion caused by TP
goal: (A1, B1), TP (B2 → A2)
t_0: A1 busy, gen (A2, B1), (A2, B2)
t_1: TP (B2 → A2)
t_2: A2 full, fail to gen (A1, A2)
busy
idle
A1
A2
B1
B2
available buffer
EPR pairs
data qubit
entanglement swapping

Figure 7: (a) A normal EPR split. (b) Deadlock caused by multiple EPR splits. (c) Buffer congestion caused by TP communication.

required by the program urgently. However, for those not in the front layer, which are less urgent, we add this condition to mitigate buffer congestion, requiring the total number of available buffer qubits and available communication qubits on each involved QPU to exceed an adjustable threshold.

4.3 Cross-rack EPR Split

This subsection introduces the conditions for EPR pair split. While cross-rack split can help maximize the utilization of network bandwidth, its flexibility can bring a risk of deadlocks or buffer congestion when the involved QPUs do not have enough buffer available. As a result, we impose the following conditions to mitigate this risk, first listing them and then explain. For the rare cases that cannot be prevented by these conditions, they will be resolved by our retry mechanism that will be introduced in Section 4.5.

Split Conditions:

- **Hard:** available communication qubits on another QPU in the same rack as the busy QPU
- **Soft:** projected_buffer − reserved_buffer ≥ m_{QPU} for each QPU involved in the post-split EPR pairs

Hard Condition The hard condition ensures that there are available communication qubits on another QPU in the same rack as the busy QPU. We illustrate the hard condition by an example in Fig. 7(a). When a QPU A_1 is busy, the cross-rack EPR pair (A_1, B_1) can be split into a new cross-rack pair (A_2, B_1) and an additional in-rack pair (A_1, A_2). This requires that QPU A_2 in the same rack as A_1 has an available communication qubit. This split is followed by an entanglement swapping that merges the two pairs into the required EPR pair (A_1, B_1) and a buffer release after (A_1, B_1) is consumed by communication.

Soft Condition The soft condition aims to ensure that there is enough buffer on the involved QPUs to generate all the post-split EPR pairs. In the soft condition, the projected_buffer of a QPU is defined as the buffer size it would have if all currently scheduled EPR pairs were consumed by the execution of corresponding communications. This reflects the maximum buffer size available in the near term. When calculating this variable, if an EPR pair is scheduled for a Cat protocol, the buffer size of each involved QPU should increase by 1 after the communication. In contrast, if the EPR pair is scheduled for a TP protocol with data teleported from QPU A to QPU B, then the buffer size of A should increase by 2 after the communication, while the buffer size of B should remain unchanged, as the released buffer qubit on B would be occupied by the teleported data qubit.

With this variable, a basic (but not sufficient) condition for EPR split is that the projected_buffer on each QPU should at least be able to accommodate all EPR pairs induced by an individual split, i.e., projected_buffer ≥ m_{QPU}, with m_{QPU} being the number of post-split EPR pairs each involved QPU needs to store. For example, if an EPR pair (A, B) is split into (A, A') and (A', B), then there should be $m_A = m_B = 1$ and $m_{A'} = 2$, as each of A and B needs to store only one EPR pair, while A' needs to store two EPR pairs.

However, this basic condition is not sufficient to prevent deadlock caused by multiple EPR splits. We illustrate this deadlock risk with an example in Fig. 7(b). The two orange EPR pairs (A_1, A_2) and (A_2, B_1) are the post-split pairs of an EPR pair (A_1, B_1), while the two blue EPR pairs (A_1, A_2) and (A_2, B_2) are the post-split pairs of an EPR pair (A_1, B_2). Supposing QPU A_2 has projected_buffer = 2, while the two post-split cross-rack pairs are scheduled to be generated simultaneously, then the post-split in-rack EPR pairs would never have a chance to be scheduled due to a deadlock. This is because the orange in-rack pair will be waiting for buffer release from the blue cross-rack pair, which requires the blue in-rack pair to be generated first; while the blue in-rack pair will be waiting for buffer release from the orange cross-rack pair, which in turn requires the orange in-rack pair to be generated first.

To accommodate simultaneous splits of multiple EPR pairs, we need to prevent such deadlock by reserving enough buffer size on QPUs. Specifically, for each EPR split, we reserve m_{QPU} buffer qubits from projected_buffer until the post-split EPR pairs are all scheduled, with m_{QPU} being the number of post-split EPR pairs on each involved QPU as explained earlier. This is implemented by tracking a variable reserved_buffer on each QPU, which is initiated as 0. For each split, we increase the reserved_buffer of each QPU by m_{QPU} when the first post-split EPR pair of this split is scheduled, then decrease them by m_{QPU} once all post-split pairs of this split are scheduled. With this reservation, the condition for an EPR split becomes projected_buffer − reserved_buffer $\geq m_{QPU}$.

4.4 Post-split distillation

Since the split of cross-rack communications requires additional in-rack EPR pairs, it reduces the overall fidelity since the fidelity of EPR pairs is much lower than the local gates on each QPU. To improve the overall fidelity, we incorporate an entanglement distillation for the post-split in-rack EPR pairs. Given the 95% fidelity of in-rack EPR pairs, a distillation of two EPR pairs enhances the fidelity to > 96.5%. However, the additional EPR pair required by the distillation also needs to occupy a buffer qubit on the QPU. As a result, if an EPR pair (A, B) is split into (A, A') and (A', B), then we should increase m_{QPU} to $m_A = 2$, $m_{A'} = 3$ and $m_B = 1$.

When k-pair distillation is considered, a feasible strategy is distilling with the $k-1$ sacrificed pairs sequentially [38, 43], as these k pairs are generated sequentially through an in-rack optical channel. This strategy reduces the total wait time of EPR pairs, preventing the sacrificed EPR pairs from error exposure, and is efficient in buffer utilization, i.e., the reserved m_{QPU} does not need to be further increased as the $k-1$ sacrificed EPR pairs can reuse the same buffer qubit sequentially. Depending on QPU performance, it may be worthy to wait for the generation of all the k pairs and conduct a parallel distillation [35]. In this case, m_{QPU} should be further increased to $m_A = k$, $m_{A'} = k+1$ and $m_B = 1$.

4.5 Algorithm

This subsection describes the overall algorithm of our compiler, which provides each QPU and switch with a deadlock-free and congestion-free communication schedule that dictates their operations at each time slice. Specifically, the compiler optimizes the communication time slice by time slice, looking ahead into near-future EPR demand of each time slice, performing in-rack collection and cross-rack split, scheduling EPR pairs through the proposed conditions to prevent deadlocks and buffer congestion. For the rare cases that could not be prevented, it retries the scheduling with a conservative strategy that ensures the elimination of any deadlock or buffer congestion.

Look-ahead scheduling At each time slice t_0, our compiler looks into the subgraph of the first l layers of the DAG, maximizing the number of scheduled EPR pairs greedily. This scheduling consists of two rounds, with the first round for the scheduling of regular EPR pairs and the second round for EPR split and the scheduling of post-split EPR pairs.

In the first round, it tries scheduling each node in the subgraph in the ascending order of their layers. If the EPR pair represented by a node satisfies the scheduling conditions, then we schedule it, removing the node and updating the dependencies between the remaining nodes in the subgraph. Otherwise, we continue with the next node until no EPR pair in the first l layers satisfies the conditions.

Then we conduct a second round of scheduling if network bandwidth remains. Specifically, we try splitting each node in the remaining subgraph in the ascending order of node layer. If an EPR pair satisfies the split conditions, we split the EPR pair, scheduling the post-split cross rack pair and adding the post-split in-rack pairs into the subgraph. Then we continue to the next node in the updated subgraph until all nodes in the subgraph are traversed. After that, we repeat these two rounds for time slice $t_0 + 1$.

Auto retry While the conditions for scheduling and EPR split can significantly mitigate deadlock and buffer congestion, it is still possible that they may occur occasionally. For example, in Fig. 7(c), the green EPR pair is for a TP communication that teleports a qubit from B_2 to A_2. While a qubit on A_2 would be released after EPR pair (A_2, B_2) is consumed by this TP communication, this released qubit would be taken by the teleported data, making the remaining buffer size insufficient for generating (A_1, A_2) and (A_2, B_1), i.e., the post-split EPR pairs of (A_1, B_2).

To completely eliminate deadlocks and buffer congestion in the compilation, we implement an automatic retry mechanism that reverts to a saved state and retries scheduling with a gradually more conservative strategy if a deadlock or congestion occurs. The most conservative approach would be a strict on-demand EPR pair generation, which follows the exact order of EPR pair generations set by the pre-processing stage, scheduling the generation of each EPR pair right before it is required by an inter-QPU communication. This ensures a deadlock-free and congestion-free scheduling in the worst case.

While ensuring progress, the strict on-demand strategy significantly limits opportunities for parallelizing cross-rack communications. To improve this, we enhance the strategy into a buffer-assisted on-demand strategy that allows parallelization of communications between non-overlapping QPU pairs through buffer utilization. Specifically, this strategy checks the list of EPR pairs provided by the pre-processing in their strict order. If an EPR pair, say between A and B, satisfies the hard scheduling conditions, it then schedules and stores this EPR pair if either its predecessors do not involve A and B or if those predecessors are also successfully scheduled. Note that this strategy does not guarantee deadlock-free and congestion-free communication as the strict on-demand strategy does.

To summarize, whenever a failure occurs, our retry mechanism first reverts to a previously saved state and applies this buffer-assisted on-demand strategy. If the issue persists, it then falls back to the strict on-demand strategy to ensure progress.

5 Evaluation

5.1 Experiment Setup

Architecture setups In the primary experiment, we evaluate our framework on the CLOS network architecture [1, 67] as shown in Fig. 1, with 25% of the total computation qubits reserved as buffer, as listed in Table 1. Each QPU is assumed to have two communication qubits. To allow all communication qubits in a rack to work in parallel, we assume each ToR switch to have #BSMs = 2× #QPUs / rack and each switch to be a #BSMs × #BSMs switch (or 2× #BSMs ports), We take a look-ahead depth of 10, adopting 0.1ms, 1ms and 10ms as the latencies of in-rack EPR pair generation, switch configuration and cross-rack EPR pair generation respectively, according to Section 2. The settings of experiments are listed in Table 1. These parameters will be further varied in subsequent experiments.

Table 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

Benchmark programs We select a set of benchmark programs including building blocks of quantum applications and practical quantum algorithms: multi-control target gate (MCT) [48], Quantum Fourier transform (QFT) [57], Grover's Algorithm (Grover) [20], Ripple-Carry Adder (RCA) [23]. Among them, MCT represents key non-Clifford operations essential for universality and benchmarks the efficiency of multi-qubit gate decomposition. QFT and RCA [57] are crucial components of Shor's algorithm [61] as well as other applications in signal processing [6, 47] and cryptography [2]. Grover's Algorithm (Grover) [36] is also an important algorithm, which can provide quadratic quantum speed up for unstructured search problems. For Grover's algorithm, we consider the secret string with all ones and repeat the iteration by 100 times. Moreover, we increase the complexity of the RCA circuit by repeating the adder for 100 iterations, which effectively adapts it to a sum calculation.

Metrics We evaluate the compilation performance with four metrics. The first is the overall communication latency of compiled programs, abbreviated as 'latency', normalized by the latency of switch reconfiguration. We ignore the computation time within each QPU as it is much faster than inter-QPU communication.

The second and the third metrics together measure the fidelity overhead. Specifically, the second metric is the EPR overhead, indicating the number of additional distilled EPR pairs, arising from communication split, in a weighted manner. According to the 15% and 5% infidelity of cross-rack and in-rack EPR pairs (see Section 2), we count them by weights 1 and 0.33, respectively. For the distilled in-rack EPR pairs, we count each of them by a weight 0.23 based on their $< 3.5\%$ infidelity. The third metric is the average wait time of EPR pairs in buffer, which is also normalized by reconfiguration latency. We list these two overheads separately as the effect of wait time depends on the coherence time of computation qubits, which is not decided by the network but varies with different QPU technology.

The fourth metric is retry overhead, which indicates the compilation time overhead caused by the retry mechanism. Specifically, this is defined as the ratio between the total number of time steps tried in the compilation process and the number of time steps in the compilation result. Hence this value would be 1 if no retry occurs in the compilation.

Baseline Due to the lack of optimized compilers for the emerging hierarchical QDC architecture [59], we construct our baseline by combining state-of-the-art compilation techniques for general DQC [70] with shortest-path buffer-assisted on-demand EPR generation. First, similar to the preprocessing step in Section 4.1, we obtain a list of EPR pairs required by each benchmark program, with the number of those EPR pairs minimized through a buffer-aware compilation pass [70], where the buffer size is set to the same as our framework. Then, we schedule the generation of each EPR pair with the buffer-assisted on-demand strategy as explained in Section 4.5. Although in principle, this strategy does not ensure deadlock-free and congestion-free communication as the strict on-demand strategy does, we did not observe fatal issues with this strategy during experiments.

5.2 Primary Experiment Result

Outperformance Table 2 presents the comparison of our framework with the baseline as the number of qubits per QPU increases, as the number of QPU per rack increases and as the number of racks increases. Besides CLOS, we also include two other commonly used network topologies, i.e., spine-leaf topology [67] and fat tree topology [21]. It can be seen that our compiler significantly reduces the overall latency, with an average improvement factor of 8.02.

The results of the baseline stay stable with #qubits per QPU, as the increase of #qubits per QPU of primarily affects QPU computation rather than inter-QPU communication. Only the QFT results of the baseline vary with #qubits per QPU, since it has a denser communication pattern than other benchmarks. In contrast, the latency of our compiler varies with #qubits per QPU, as the allocated buffer size increases with #qubits per QPU. This leads to a slightly increased improvement as #qubits per QPU increases.

The improvement of our compiler increases with #QPUs per rack, showing the ability of our compiler to mitigate the bandwidth contention caused by the increased #QPUs. It also increases with

Table 2: Performance of our compiler and the baseline. Units of latency and wait time are the latency of switch reconfiguration.

Experiment	Benchmark	Baseline: Latency	Ours: Latency	Improv. Factor	#cross-rack EPR (15% infidelity)	#in-rack EPR (5% infidelity)	Ours: #distilled EPR (3.5% infidelity)	EPR Overhead	Baseline: Wait Time	Ours: Wait Time	Additional Wait Time	Retry Overhead
Increase #qubits/QPU	MCT-480	2,312	485	**4.77×**	15	240	13	3.09%	1.98	6.75	4.77	1.00
	MCT-608	2,312	454	**5.09×**	15	240	15	3.55%	1.98	6.36	4.39	1.00
	MCT-720	2,312	382	**6.05×**	15	240	16	3.78%	1.98	8.23	6.25	1.00
	QFT-480	121,728	16,693	**7.29×**	1,080	2,970	1,133	11.32%	1.30	6.87	5.57	1.00
	QFT-608	155,960	20,781	**7.50×**	1,368	3,762	1,452	11.44%	1.26	6.92	5.66	1.00
	QFT-720	194,526	24,670	**7.89×**	1,620	4,455	1,772	11.75%	1.00	7.77	6.77	1.00
	Grover-480	156,213	26,943	**5.80×**	1,800	7,200	1,927	9.67%	2.08	8.73	6.65	1.00
	Grover-608	150,702	27,412	**5.50×**	1,800	7,200	1,933	9.70%	1.87	9.54	7.67	1.00
	Grover-720	156,213	25,883	**6.04×**	1,800	7,200	2,122	10.55%	2.08	9.14	7.06	1.00
	RCA-480	92,259	9,169	**10.06×**	603	2,412	630	9.46%	0.03	8.49	8.46	1.00
	RCA-608	92,259	9,304	**9.92×**	603	2,412	650	9.73%	0.03	8.57	8.54	1.00
	RCA-720	92,226	9,395	**9.82×**	603	2,404	658	9.86%	0.01	9.78	9.77	1.00
Increase #QPUs/rack	MCT-360	1,476	468	**3.15×**	15	144	15	5.26%	3.03	5.81	2.78	1.00
	MCT-480	2,312	485	**4.77×**	15	240	13	3.09%	1.98	6.75	4.77	1.00
	MCT-600	3,214	634	**5.07×**	15	432	12	1.73%	1.17	5.30	4.13	1.00
	MCT-720*	4,413	921	**4.79×**	15	624	11	1.14%	0.87	5.27	4.40	1.00
	QFT-360	78,300	13,504	**5.80×**	810	1,500	715	11.30%	1.48	6.27	4.79	1.00
	QFT-480	121,728	16,693	**7.29×**	1,080	2,970	1,133	11.32%	1.30	6.87	5.57	1.00
	QFT-600	169,831	20,041	**8.47×**	1,350	4,920	1,559	10.85%	1.14	7.00	5.86	1.00
	QFT-720*	216,372	23,362	**9.26×**	1,620	7,350	1,915	9.89%	1.20	7.63	6.43	1.00
	Grover-360	140,813	29,717	**4.74×**	1,800	4,800	1,594	9.86%	2.03	9.96	7.93	1.00
	Grover-480	156,213	26,943	**5.80×**	1,800	7,200	1,927	9.67%	2.08	8.73	6.65	1.00
	Grover-600	171,613	25,438	**6.75×**	1,800	9,600	2,057	8.76%	2.08	8.77	6.68	1.00
	Grover-720*	187,013	24,580	**7.61×**	1,800	12,000	2,178	8.06%	2.07	8.85	6.77	1.00
	RCA-360	83,470	10,127	**8.24×**	603	1,608	531	9.81%	0.03	9.69	9.66	1.00
	RCA-480	92,259	9,169	**10.06×**	603	2,412	630	9.46%	0.03	8.49	8.46	1.00
	RCA-600	101,048	8,916	**11.33×**	603	3,216	683	8.69%	0.03	8.68	8.64	1.00
	RCA-720*	109,837	8,592	**12.78×**	603	4,020	710	7.86%	0.03	8.78	8.75	1.00
Increase #racks	MCT-240	1,575	490	**3.21×**	15	144	14	4.93%	2.74	4.66	1.92	1.00
	MCT-540	6,514	3,069	**2.12×**	99	900	52	2.95%	3.12	5.14	2.03	1.15
	MCT-960	15,369	5,415	**2.84×**	255	2,304	170	3.73%	3.21	5.44	2.23	1.22
	QFT-240	50,195	9,663	**5.19×**	540	1,000	389	9.41%	1.61	6.25	4.64	1.00
	QFT-540	212,522	28,694	**7.41×**	2,640	4,900	1,802	8.96%	2.36	6.28	3.93	1.00
	QFT-960	564,973	58,482	**9.66×**	8,100	15,400	4,250	6.97%	3.05	6.61	3.56	1.00
	Grover-240	147,468	34,265	**4.30×**	1,800	4,800	1,248	7.89%	1.00	9.54	8.54	1.01
	Grover-540	365,312	38,648	**9.45×**	4,800	10,800	3,071	7.86%	0.99	8.63	7.65	1.00
	Grover-960	665,612	39,969	**16.65×**	9,000	19,200	4,576	6.48%	0.98	7.99	7.01	1.00
	RCA-240	83,470	11,050	**7.55×**	603	1,608	432	8.13%	0.03	8.91	8.88	1.00
	RCA-540	213,523	12,666	**16.86×**	1,608	3,618	853	6.61%	0.08	7.10	7.02	1.00
	RCA-960	399,478	13,240	**30.17×**	3,015	6,432	924	4.01%	0.03	7.03	7.00	1.00
Spine-leaf topology	MCT-720	4,611	1,008	**4.57×**	39	600	33	3.12%	2.24	6.60	4.36	1.00
	QFT-720	249,317	34,657	**7.19×**	2,400	6,570	1,767	8.24%	1.77	9.89	8.12	1.00
	Grover-720	242,013	34,357	**7.04×**	3,000	10,800	2,001	6.61%	2.22	11.63	9.42	1.00
	RCA-720	149,316	11,418	**13.08×**	1,005	3,618	679	6.69%	0.03	10.36	10.33	1.00
Fat-tree topology	MCT-960	6,910	2,497	**2.77×**	63	960	30	1.79%	2.44	5.82	3.38	2.34
	QFT-960	421,181	49,568	**8.50×**	4,200	11,610	3,107	8.24%	1.99	9.40	7.41	1.00
	Grover-960	327,813	39,193	**8.36×**	4,200	14,400	2,420	5.90%	2.28	12.80	10.52	1.00
	RCA-960	206,373	12,639	**16.33×**	1,407	4,824	896	6.48%	0.03	10.52	10.49	1.00

#racks, exhibiting the capability of our compiler of effectively utilizing the increased cross-rack bandwidth. These results demonstrate the scalability of our compiler. Furthermore, the improvement factors of the other two network topologies are at a similar level with CLOS network, which demonstrates the general applicability of our compiler to various network topologies.

Overhead It can also be seen from Table 2 that these improvements are achieved with a small overhead. First, our compiler requires the generation of 7.41% more (weighted) EPR pairs on average, which decreases as the #QPUs per rack or #racks increases. Second, our compiler increases the wait time of generated EPR pairs in the buffer by only 6.51× reconfiguration latency on average. This wait time is acceptable since it is even less than the latency of generating one cross-rack EPR pair. Third, the retry overhead is very close to 1 for most programs, indicating that the retry rarely occurs in the compilation. Only one of the programs (i.e., MCT-960 in fat-tree topology) presents a relative high retry overhead of 2.34. This is because a very early scheduled EPR pair generation caused a buffer congestion for very late EPR pairs, causing the compiler to retry multiple times.

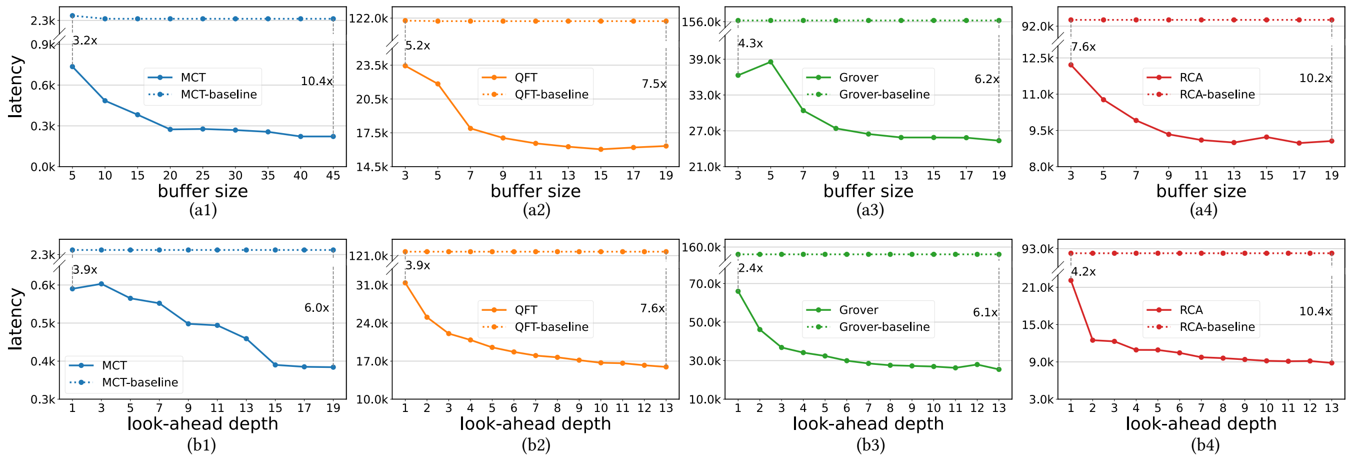

Figure 8: Performance improvement of our compiler varying with (a) buffer size and (b) look-ahead depth.

5.3 Choice of Hyper-parameters

Buffer size We illustrate the effect of varying buffer size while keeping all other parameters the same as program-480 in Table 1. Fig. 8(a) shows the overall latency of baseline and our compiler as the buffer size increases. The latency of our compiled programs first decreases with the buffer size and then becomes stable, with the improvement factor first increasing and then becoming stable. The turning points of QFT, Grover and RCA are around 7, which takes $7/(30+7) = 18.9\%$ of the total #qubits per QPU. In contrast, the turning point of MCT is around 20, which is much larger than other benchmarks. This is because MCT is much more dominated by in-rack communications than other benchmarks. It can benefit from a larger buffer size, as in-rack EPR pairs can be collected and stored in the buffer with less restriction by network bandwidth.

Look-ahead depth We also illustrate the effect of varying look-ahead depth while keeping all other parameters the same as program-480 in Table 1. Fig. 8(b) shows the overall latency of baseline and our compiler as the look-ahead depth increases. The latency of our compiler first decreases with the look-ahead depth and then becomes stable, with the improvement factor first increasing

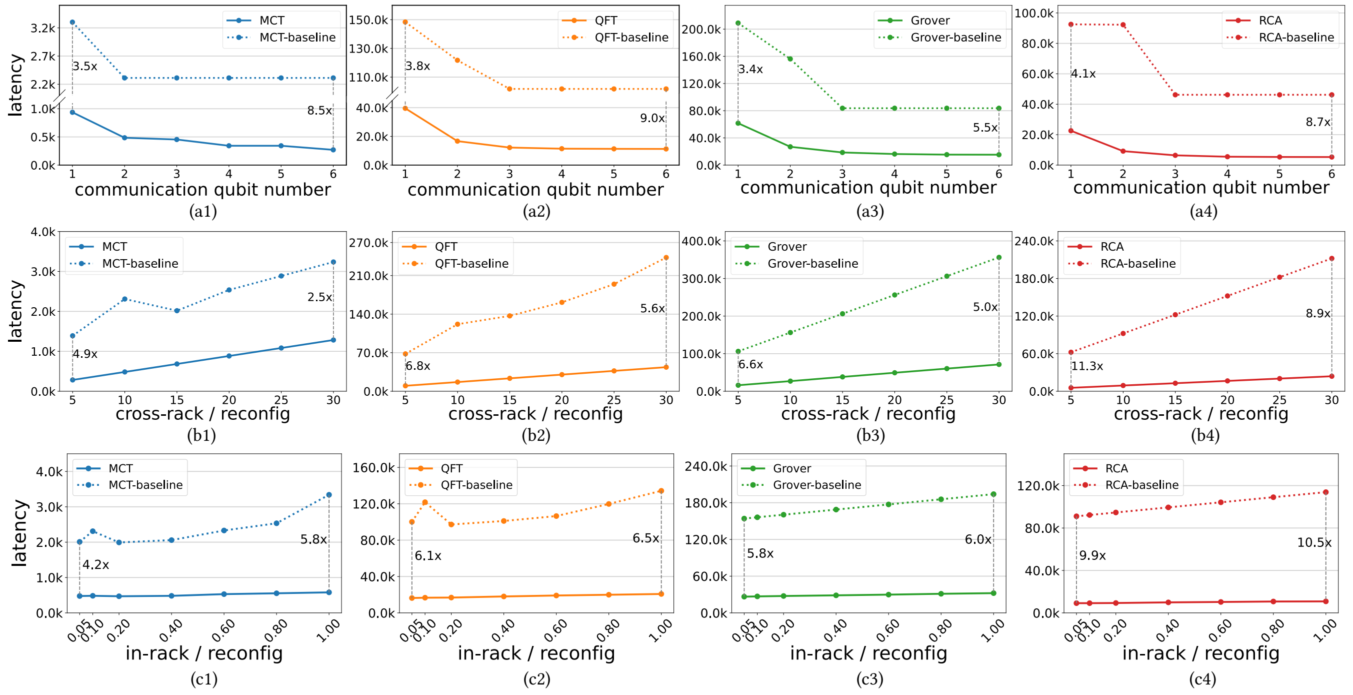

Figure 9: Performance improvement of our compiler varying with (a) #communication qubits per QPU, (b) cross-rack EPR latency and (c) in-rack EPR latency (both normalized by reconfiguration latency).

and then becoming stable. Similar to buffer size, the turning point of MCT is larger than other benchmarks. It can benefit from a larger look-ahead depth, as an increased look-ahead depth leads to an increased collection of its more dominant in-rack communications.

5.4 Sensitivity Analysis

This subsection provides a sensitivity analysis for various hardware parameters, demonstrating the adaptability of our compiler to a varying EPR supply (i.e., the number of communication qubits per QPU and the latencies of cross-rack and in-rack EPR generation) and a varying EPR quality (i.e., fidelity of cross-rack, in-rack and distilled EPR pairs). Since only the ratios between different latencies and fidelity matter, cross-rack and in-rack latency will be normalized by reconfiguration latency, while cross-rack and distilled in-rack fidelity will be normalized by the original in-rack fidelity.

Communication qubit number We illustrate the effect of varying #communication qubits per QPU by increasing it from 1 to 6, keeping all other parameters the same as program-480 in Table 1. As shown in Fig. 9(a), the overall latency of both baseline and our compiler decreases first and then becomes stable, which naturally arises from the increased bandwidth by communication qubits. The improvement factor of our compiler first increases and then becomes stable, which demonstrates its more effective utilization of network bandwidth than the baseline.

Cross-rack EPR latency We illustrate the effect of varying the latency of cross-rack EPR pair generation by increasing its ratio to reconfiguration latency from 5 to 30, keeping all other parameters the same as program-480 in Table 1. As shown in Fig. 9(b), the overall latency of both baseline and our compiler increases with cross-rack latency. These trends arise naturally from the fact that a longer cross-rack EPR latency leads to a longer overall latency. The improvement factor of our compiler decreases with an increased cross-rack latency, but remains significant even if cross-rack latency is as large as 30× reconfiguration latency.

In-rack EPR latency We illustrate the effect of varying the latency of in-rack EPR pair generation by increasing its ratio to reconfiguration latency from 0.05 to 1, keeping all other parameters the same as program-480 in Table 1. As shown in Fig. 9(c), the overall latency of both baseline and our compiler increases with in-rack latency. These trends arise naturally from the fact that a longer in-rack EPR latency leads to a longer overall latency. In contrast to cross-rack latency, the improvement factor of our compiler slightly increases with the in-rack latency.

Relative cross-rack fidelity We analyze the effect of varying the fidelity of cross-rack EPR pairs. We fix the fidelity of in-rack EPR pairs to 95%, varying the fidelity of cross-rack EPR pairs from 75% to 95%, with the ratio between cross-rack and in-rack ones being from 0.79 to 1. All other parameters are kept the same as program-480 in Table 1. As shown in Fig. 10(a), the EPR overhead of our compiler slightly increases as this fidelity ratio becomes closer to 1. This is because our compiler adopts a tradeoff of incurring additional for hiding latencies of cross-rack communications. With a smaller fidelity distinction between cross-rack and in-rack pairs, the cost of additional in-rack EPR pairs would appear more significant.

Relative distilled in-rack fidelity We also analyze the effect of varying the fidelity of distilled in-rack EPR pairs, as the distillation

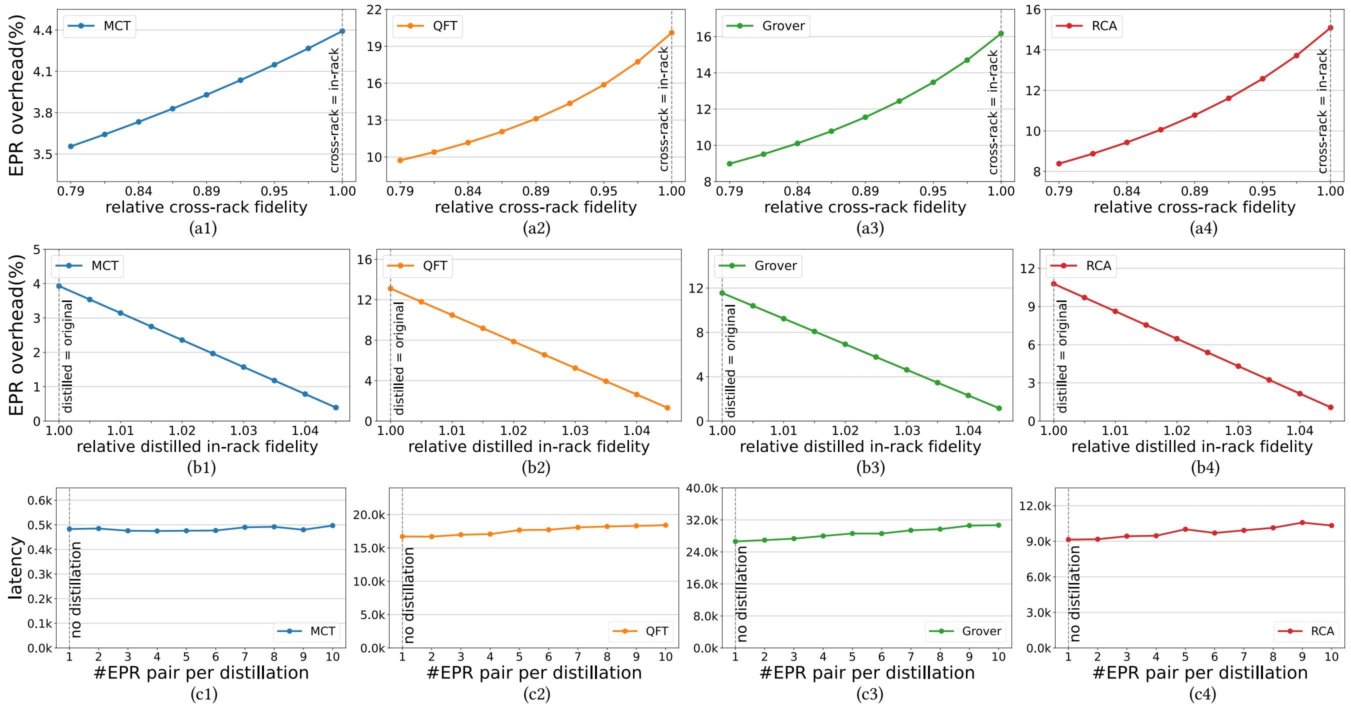

Figure 10: Fidelity overhead of our compiler varying with (a) cross-rack fidelity and (b) distilled in-rack fidelity (both relative to the original in-rack fidelity). (c) The overall latency varying with in-rack distillation through different numbers of EPR pairs.

Table 3: QEC Integration: performance of our compiler and the baseline with surface code of distance 5.

Experiment	Benchmark-#alg_qubits	Baseline: Latency	Ours: Latency	Improv. Factor	#cross-rack EPR (15% infidelity)	#in-rack EPR (5% infidelity)	Ours: #distilled EPR (3.5% infidelity)	EPR Overhead	Baseline: Wait Time	Ours: Wait Time	Additional Wait Time	Retry Overhead
Surface Code ($d = 5$)	MCT-64	145,202	35,859	**4.05×**	1,920	7,680	2,621	12.01%	0.00	2.40	2.40	1.00
	QFT-64	1,239,922	277,850	**4.46×**	15,360	3,840	19,889	21.81%	0.00	5.02	5.02	1.00
	Grover-64	18,482	2,718	**6.80×**	120	480	180	13.04%	0.00	1.31	1.31	1.00
	RCA-64	16,777	3,965	**4.23×**	180	720	249	12.15%	0.43	3.02	2.59	1.00

can be improved by advancing distillation protocols or sacrificing more EPR pairs for the distillation. We fix the fidelity of in-rack EPR pairs to 95%, varying the fidelity of distilled in-rack EPR pairs form 95% to 99.5%, with the ratio between distilled in-rack and (non-distilled) in-rack ones being from 1 to 1.047. All other parameters are kept the same as program-480 in Table 1. As shown in Fig. 10(b), the EPR overhead of our compiler decreases rapidly with this fidelity ratio. That means if we can distill the additional in-rack EPR pairs to a high enough fidelity, the fidelity overhead brought by them will become negligible.

#EPR pairs per distillation While the fidelity of distilled in-rack EPR pairs can be enhanced by sacrificing more EPR pairs, the generation of these sacrificed EPR pairs can also increase the overall latency. However, our experiments show that this increase is not significant. This is because all the sacrificed EPR pairs are in-rack pairs, which can be generated collectively in our compiler. As shown in Fig. 10(c), the overall latency is increased by only 7.4% on average as the number of EPR pairs used in each distillation increases from 1 (i.e., no distillation) to 10.

5.5 Integration of QEC

We demonstrate the capability of our compiler to integrate QEC by an experiment with programs encoded in surface code, one of the most promising QEC codes [33]. We decompose programs into the Clifford + T basis [53], implementing logical operations with lattice surgery [44, 69], along with a magic state factory to facilitate logical T gates [32]. Therefore, the logical qubits include both algorithmic qubits required by the quantum program and additional qubits to facilitate quantum computation (e.g., magic states for implementing logical T gates [15]).

We adopt an architecture steup similar to the primary experiment, which consists of 4 racks, with each rack containing 4 QPUs, each QPU containing 2 communication qubits. Each benchmark contains 64 algorithmic qubtis. On each QPU, 4 algorithmic qubits of code distance $d = 5$ are arranged in a lattice, separated by ancilla qubits with a spacing of d [44], surrounded by a magic state factory at the periphery of the QPU. The EPR pairs can be stored with a low resource overhead [16, 62] to prevent from decoherence, until participating syndrome measurements [39]. Specifically, each QPU has a buffer of 12 logical qubits with a code distance of 6, which are encoded in the [[72,12,6]] LDPC code [16]. This leads to a buffer qubit overhead that is much less than the number of qubits used for computation.

As shown in Table 3, our framework achieves a reduction in latency by an average factor of 4.89, with an average EPR pair overhead of 14.75%, an average additional wait time of 2.83, and an average retry overhead of 1.00 (i.e., no retry occurs). This demonstrates the applicability of our framework in fault tolerant quantum computing.

6 Conclusion

In this work, we provide in-depth analysis and discussion of the compilation challenges of scaling up quantum computing with QDCs based on reconfigurable optical switch network. We propose a compiler that optimizes the quantum communication in QDCs across both the program and network layers, which employs a look-ahead EPR scheduling along with two key optimizations: collective in-rack EPR pair generation and parallelized cross-rack EPR pairs generation. This compiler reduces the overall communication latency by a factor of 8.02, with an overhead of requiring 7.41% more EPR generation and increasing the wait time of EPR pairs by 6.51× switch reconfiguration latency. We also demonstrate its capability of integrating QEC by an evaluation on surface code. Moreover, we have open-sourced our codes to facilitate further research and collaboration within the community.

7 Open-source Codes

Our codes have been open-sourced and can be downloaded from https://zenodo.org/records/15377656.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. This work is supported in part by Cisco Research, NSF 2048144, NSF 2422169 and NSF 2427109.

References

[1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review* 38, 4 (2008), 63–74.

[2] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. 2016. Post-quantum key {Exchange—A} new hope. In *25th USENIX Security Symposium (USENIX Security 16)*. 327–343.

[3] Pablo Andr'es-Mart'inez and Chris Heunen. 2019. Automated distribution of quantum circuits via hypergraph partitioning. *Physical Review A* (2019).

[4] Pablo Andres-Martinez and Chris Heunen. 2019. Automated distribution of quantum circuits via hypergraph partitioning. *Physical Review A* 100, 3 (2019), 032308.

[5] James Ang, Gabriella Carini, Yanzhu Chen, Isaac Chuang, Michael Demarco, Sophia Economou, Alec Eickbusch, Andrei Faraon, Kai-Mei Fu, Steven Girvin, et al. 2024. ARQUIN: architectures for multinode superconducting quantum computers. *ACM Transactions on Quantum Computing* 5, 3 (2024), 1–59.

[6] Alán Aspuru-Guzik, Anthony D Dutoi, Peter J Love, and Martin Head-Gordon. 2005. Simulated quantum computation of molecular energies. *Science* 309, 5741 (2005), 1704–1707.

[7] David Awschalom, Karl K Berggren, Hannes Bernien, Sunil Bhave, Lincoln D Carr, Paul Davids, Sophia E Economou, Dirk Englund, Andrei Faraon, Martin Fejer, et al. 2021. Development of quantum interconnects (quics) for next-generation information technologies. *Prx Quantum* 2, 1 (2021), 017002.

[8] Koji Azuma, Sophia E Economou, David Elkouss, Paul Hilaire, Liang Jiang, Hoi-Kwong Lo, and Ilan Tzitrin. 2022. Quantum repeaters: From quantum networks to the quantum internet. *arXiv preprint arXiv:2212.10820* (2022).

[9] Jonathan M Baker, Casey Duckering, Alexander Hoover, and Frederic T Chong. 2020. Time-sliced quantum circuit partitioning for modular architectures. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*. 98–107.

[10] Robert Beals, Stephen Brierley, Oliver Gray, Aram W Harrow, Samuel Kutin, Noah Linden, Dan Shepherd, and Mark Stather. 2013. Efficient distributed quantum computing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 469, 2153 (2013), 20120686.

[11] Charles H Bennett, Gilles Brassard, Sandu Popescu, Benjamin Schumacher, John A Smolin, and William K Wootters. 1996. Purification of noisy entanglement and faithful teleportation via noisy channels. *Physical review letters* 76, 5 (1996), 722.

[12] Eric Bersin, Matthew Grein, Madison Sutula, Ryan Murphy, Yan Qi Huan, Mark Stevens, Aziza Suleymanzade, Catherine Lee, Ralf Riedinger, David J Starling, et al. 2024. Development of a Boston-area 50-km fiber quantum network testbed. *Physical Review Applied* 21, 1 (2024), 014024.

[13] Eric Bersin, Madison Sutula, Yan Qi Huan, Aziza Suleymanzade, Daniel R Assumpcao, Yan-Cheng Wei, Pieter-Jan Stas, Can M Knaut, Erik N Knall, Carsten Langrock, et al. 2024. Telecom networking with a diamond quantum memory. *PRX Quantum* 5, 1 (2024), 010303.

[14] Hans KC Beukers, Matteo Pasini, Hyeongrak Choi, Dirk Englund, Ronald Hanson, and Johannes Borregaard. 2023. Tutorial: Remote entanglement protocols for stationary qubits with photonic interfaces. *arXiv preprint arXiv:2310.19878* (2023).

[15] Michael Beverland, Vadym Kliuchnikov, and Eddie Schoute. 2022. Surface code compilation via edge-disjoint paths. *PRX Quantum* 3, 2 (2022), 020342.

[16] Sergey Bravyi, Andrew W Cross, Jay M Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J Yoder. 2024. High-threshold and low-overhead fault-tolerant quantum memory. *Nature* 627, 8005 (2024), 778–782.

[17] H-J Briegel, Wolfgang Dür, Juan I Cirac, and Peter Zoller. 1998. Quantum repeaters: the role of imperfect local operations in quantum communication. *Physical Review Letters* 81, 26 (1998), 5932.

[18] Angela Sara Cacciapuoti, Marcello Caleffi, Francesco Tafuri, Francesco Saverio Cataliotti, Stefano Gherardini, and Giuseppe Bianchi. 2019. Quantum internet: Networking challenges in distributed quantum computing. *IEEE Network* 34, 1 (2019), 137–143.

[19] Marcello Caleffi, Michele Amoretti, Davide Ferrari, Daniele Cuomo, Jessica Illiano, Antonio Manzalini, and Angela Sara Cacciapuoti. 2022. Distributed quantum computing: a survey. *arXiv:2212.10609* (2022).

[20] Yu-Fang Chen, Kai-Min Chung, Ondřej Lengál, Jyun-Ao Lin, Wei-Lun Tsai, and Di-De Yen. 2023. An Automata-based Framework for Verification and Bug Hunting in Quantum Circuits (Technical Report). *arXiv preprint arXiv:2301.07747* (2023).

[21] Hyeongrak Choi, Marc G Davis, Álvaro G Iñesta, and Dirk R Englund. 2023. Scalable quantum networks: Congestion-free hierarchical entanglement routing with error correction. *arXiv preprint arXiv:2306.09216* (2023).

[22] Hyeongrak Choi, Mihir Pant, Saikat Guha, and Dirk Englund. 2019. Percolation-based architecture for cluster state creation using photon-mediated entanglement between atomic memories. *npj Quantum Information* 5, 1 (2019), 104.

[23] Steven A Cuccaro, Thomas G Draper, Samuel A Kutin, and David Petrie Moulton. 2004. A new quantum ripple-carry addition circuit. *arXiv preprint quant-ph/0410184* (2004). https://doi.org/10.48550/arXiv.quant-ph/0410184

[24] Daniele Cuomo, Marcello Caleffi, and Angela Sara Cacciapuoti. 2020. Towards a distributed quantum computing ecosystem. *IET Quantum Communication* 1, 1 (2020), 3–8.

[25] Davood Dadkhah, Mariam Zomorodi, Seyed Ebrahim Hosseini, Pawel Plawiak, and Xujuan Zhou. 2022. Reordering and partitioning of distributed quantum circuits. *IEEE Access* 10 (2022), 70329–70341.

[26] Omid Daei, Keivan Navi, and Mariam Zomorodi-Moghadam. 2020. Optimized quantum circuit partitioning. *International Journal of Theoretical Physics* 59, 12 (2020), 3804–3820.

[27] Zohreh Davarzani, Mariam Zomorodi-Moghadam, Mahboobeh Houshmand, and Mostafa Nouri-Baygi. 2020. A dynamic programming approach for distributing quantum circuits by bipartite graphs. *Quantum Information Processing* 19 (2020), 1–18.

[28] C Delle Donne, M Iuliano, B Van Der Vecht, GM Ferreira, H Jirovská, TJW Van Der Steenhoven, A Dahlberg, M Skrzypczyk, D Fioretto, M Teller, et al. 2025. An operating system for executing applications on quantum network nodes. *Nature* 639, 8054 (2025), 321–328.

[29] Stephen DiAdamo, Marco Ghibaudi, and James Cruise. 2021. Distributed quantum computing and network control for accelerated vqe. *IEEE Transactions on Quantum Engineering* 2 (2021), 1–21.

[30] Duncan Earl, K Karunaratne, Jason Schaake, Ryan Strum, Patrick Swingle, and Ryan Wilson. 2022. Architecture of a First-Generation Commercial Quantum Network. *arXiv preprint arXiv:2211.14871* (2022).

[31] Davide Ferrari, Angela Sara Cacciapuoti, Michele Amoretti, and Marcello Caleffi. 2021. Compiler design for distributed quantum computing. *IEEE Transactions on Quantum Engineering* 2 (2021), 1–20.

[32] Austin G Fowler and Craig Gidney. 2018. Low overhead quantum computation using lattice surgery. *arXiv preprint arXiv:1808.06709* (2018).

[33] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. 2012. Surface codes: Towards practical large-scale quantum computation. *Physical Review A—Atomic, Molecular, and Optical Physics* 86, 3 (2012), 032324.

[34] Scarlett Gauthier, Gayane Vardoyan, and Stephanie Wehner. 2023. A Control Architecture for Entanglement Generation Switches in Quantum Networks. In *Proceedings of the 1st Workshop on Quantum Networks and Distributed Quantum Computing*. 38–44.

[35] Craig Gidney. 2023. Tetrationally compact entanglement purification. *arXiv preprint arXiv:2311.10971* (2023).

[36] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 212–219.

[37] Thomas Häner, Damian S Steiger, Torsten Hoefler, and Matthias Troyer. 2021. Distributed quantum computing with QMPI. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–13.

[38] Ziyue Jia and Lin Chen. 2024. On fidelity-oriented entanglement distribution for quantum switches. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2024).

[39] Junpyo Kim, Dongmoon Min, Jungmin Cho, Hyeonseong Jeong, Ilkwon Byun, Junhyuk Choi, Juwon Hong, and Jangwoo Kim. 2024. A Fault-Tolerant Million Qubit-Scale Distributed Quantum Computer. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 1–19.

[40] Can M Knaut, Aziza Suleymanzade, Y-C Wei, Daniel R Assumpcao, P-J Stas, Yan Qi Huan, Bartholomeus Machielse, Erik N Knall, Madison Sutula, Gefen Baranes, et al. 2024. Entanglement of nanophotonic quantum memory nodes in a telecom network. *Nature* 629, 8012 (2024), 573–578.

[41] Nicholas LaRacuente, Kaitlin N Smith, Poolad Imany, Kevin L Silverman, and Frederic T Chong. 2022. Modeling short-range microwave networks to scale superconducting quantum computation. *arXiv preprint arXiv:2201.08825* (2022).

[42] Jian Li, Mingjun Wang, Kaiping Xue, Ruidong Li, Nenghai Yu, Qibin Sun, and Jun Lu. 2022. Fidelity-guaranteed entanglement routing in quantum networks. *IEEE Transactions on Communications* 70, 10 (2022), 6748–6763.

[43] Jian Li, Mingjun Wang, Kaiping Xue, Ruidong Li, Nenghai Yu, Qibin Sun, and Jun Lu. 2022. Fidelity-guaranteed entanglement routing in quantum networks. *IEEE Transactions on Communications* 70, 10 (2022), 6748–6763.

[44] Daniel Litinski. 2019. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum* 3 (2019), 128.

[45] Junyu Liu, Connor T Hann, and Liang Jiang. 2023. Data centers with quantum random access memory and quantum networks. *Physical Review A* 108, 3 (2023), 032610.

[46] Junyu Liu and Liang Jiang. 2024. Quantum data center: Perspectives. *IEEE Network* (2024).

[47] Stephane Mallat. 1999. A wavelet tour of signal processing.

[48] Dmitri Maslov, Gerhard W Dueck, D Michael Miller, and Camille Negrevergne. 2008. Quantum circuit simplification and level compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 3 (2008), 436–444.

[49] Francesco Mazza, Marcello Caleffi, and Angela Sara Cacciapuoti. 2024. Quantum LAN: On-demand network topology via two-colorable graph states. In *2024 International Conference on Quantum Communications, Networking, and Computing (QCNC)*. IEEE, 127–134.

[50] Francesco Mazza, Caitao Zhan, Joaquin Chung, Rajkumar Kettimuthu, Marcello Caleffi, and Angela Sara Cacciapuoti. 2024. Simulation of Entanglement-Enabled Connectivity in QLANs using SeQUeNCe. *arXiv preprint arXiv:2411.11031* (2024).

[51] Christopher Monroe, Robert Raussendorf, Alex Ruthven, Kenneth R Brown, Peter Maunz, L-M Duan, and Jungsang Kim. 2014. Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects. *Physical Review A* 89, 2 (2014), 022317.

[52] William J. Munro, Koji Azuma, Kiyoshi Tamaki, and Kae Nemoto. 2015. Inside Quantum Repeaters. *IEEE Journal of Selected Topics in Quantum Electronics* 21, 3 (2015), 78–90. https://doi.org/10.1109/JSTQE.2015.2392076

[53] Michael A Nielsen and Isaac L Chuang. 2010. *Quantum computation and quantum information*. Cambridge university press.

[54] Mihir Pant, Hari Krovi, Don Towsley, Leandros Tassiulas, Liang Jiang, Prithwish Basu, Dirk Englund, and Saikat Guha. 2019. Routing entanglement in the quantum internet. npj Quantum Information 5 (1): 1–9. *arXiv preprint arXiv:1708.07142* (2019).

[55] Ashlesha Patil, Mihir Pant, Dirk Englund, Don Towsley, and Saikat Guha. 2022. Entanglement generation in a quantum network at distance-independent rate. *npj Quantum Information* 8, 1 (2022), 51.

[56] Tianyi Peng, Aram W Harrow, Maris Ozols, and Xiaodi Wu. 2020. Simulating large quantum circuits on a small quantum computer. *Physical review letters* 125, 15 (2020), 150504.

[57] Lidia Ruiz-Perez and Juan Carlos Garcia-Escartin. 2017. Quantum arithmetic with the quantum Fourier transform. *Quantum Information Processing* 16 (2017), 1–14.

[58] Uday Saha, James D Siverns, John Hannegan, Qudsia Quraishi, and Edo Waks. 2023. Low-Noise Quantum Frequency Conversion of Photons from a Trapped Barium Ion to the Telecom O-band. *ACS Photonics* 10, 8 (2023), 2861–2865.

[59] Hassan Shapourian, Eneet Kaur, Troy Sewell, Jiapeng Zhao, Michael Kilzer, Ramana Kompella, and Reza Nejabati. 2025. Quantum Data Center Infrastructures: A Scalable Architectural Design Perspective. *arXiv preprint arXiv:2501.05598* (2025).

[60] Shouqian Shi and Chen Qian. 2020. Concurrent entanglement routing for quantum networks: Model and designs. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication.* 62–75.

[61] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review* 41, 2 (1999), 303–332.

[62] Samuel Stein, Shifan Xu, Andrew W Cross, Theodore J Yoder, Ali Javadi-Abhari, Chenxu Liu, Kun Liu, Zeyuan Zhou, Charles Guinn, Yufei Ding, et al. 2024. Architectures for Heterogeneous Quantum Error Correction Codes. *arXiv preprint arXiv:2411.03202* (2024).

[63] Michal Stepanovsky. 2019. A comparative review of MEMS-based optical cross-connects for all-optical networks from the past to the present day. *IEEE Communications Surveys & Tutorials* 21, 3 (2019), 2928–2946.

[64] L J Stephenson, D. P. Nadlinger, B. C. Nichol, Shuoming An, P. Drmota, Timothy G. Ballance, K. Thirumalai, Joseph Francis Goodwin, David M. Lucas, and Chris Ballance. 2019. High-Rate, High-Fidelity Entanglement of Qubits Across an Elementary Quantum Network. *Physical review letters* 124 11 (2019), 110501. https://api.semanticscholar.org/CorpusID:208268137

[65] HUBER SUHNER. [n. d.]. "Polatis technology-directlight beam-steering all-optical switch". https://www.polatis.com/polatis-all-optical-switch-technology-lowestloss-highest-performancedirectlight-beam-steering.asp. Accessed: 2024.

[66] Wei Tang, Teague Tomesh, Martin Suchara, Jeffrey Larson, and Margaret Martonosi. 2021. Cutqc: using small quantum computers for large quantum circuit evaluations. In *Proceedings of the 26th ACM International conference on architectural support for programming languages and operating systems.* 473–486.

[67] Jon Tate, Pall Beck, Peter Clemens, Santiago Freitas, Jeff Gatz, Michele Girola, Jason Gmitter, Holger Mueller, Ray O'Hanlon, Veerendra Para, et al. 2013. *IBM and Cisco: together for a world class data center.* IBM Redbooks.

[68] Tim van Leent, Matthias Bock, Florian Fertig, Robert Garthoff, Sebastian Eppelt, Yiru Zhou, Pooja Malik, Matthias Seubert, Tobias Bauer, Wenjamin Rosenfeld, et al. 2022. Entangling single atoms over 33 km telecom fibre. *Nature* 607, 7917 (2022), 69–73.

[69] George Watkins, Hoang Minh Nguyen, Keelan Watkins, Steven Pearce, Hoi-Kwan Lau, and Alexandru Paler. 2024. A high performance compiler for very large scale surface code computations. *Quantum* 8 (2024), 1354.

[70] Anbang Wu, Yufei Ding, and Ang Li. 2023. Qucomm: Optimizing collective communication for distributed quantum computing. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture.* 479–493.

[71] Anbang Wu, Hezi Zhang, Gushu Li, Alireza Shabani, Yuan Xie, and Yufei Ding. 2022. Autocomm: A framework for enabling efficient communication in distributed quantum programs. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO).* IEEE, 1027–1041.

[72] Anocha Yimsiriwattana and Samuel J Lomonaco Jr. 2004. Generalized GHZ states and distributed quantum computing. *arXiv preprint quant-ph/0402148* (2004).

[73] SJ Ben Yoo, Sandeep Kumar Singh, Mehmet Berkay On, Gamze Gül, Gregory S Kanter, Roberto Proietti, and Prem Kumar. 2024. Quantum wrapper networking. *IEEE Communications Magazine* 62, 3 (2024), 76–81.

[74] Yangming Zhao and Chunming Qiao. 2021. Redundant entanglement provisioning and selection for throughput maximization in quantum networks. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications.* IEEE, 1–10.

[75] Yangming Zhao, Gongming Zhao, and Chunming Qiao. 2022. E2E fidelity aware routing and purification for throughput maximization in quantum networks. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications.* IEEE, 480–489.

[76] Yiru Zhou, Pooja Malik, Florian Fertig, Matthias Bock, Tobias Bauer, Tim van Leent, Wei Zhang, Christoph Becher, and Harald Weinfurter. 2024. Long-lived quantum memory enabling atom-photon entanglement over 101 km of telecom fiber. *PRX Quantum* 5, 2 (2024), 020307.

[77] Mariam Zomorodi-Moghadam, Mahboobeh Houshmand, and Monireh Houshmand. 2018. Optimizing teleportation cost in distributed quantum circuits. *International Journal of Theoretical Physics* 57 (2018), 848–861.