Online Scheduling and Distribution of Quantum Circuits over Quantum Networks

Yukun Yang, Himanshu Gupta
Department of Computer Science, Stony Brook University, Stony Brook, NY 11790, USA

***Abstract*—Emerging quantum networks enable *distributed quantum computing* (DQC) by interconnecting modest, noisy quantum processors. However, practical deployments face tight memory budgets, probabilistic entanglement generation, and decoherence, all while serving multiple users. In realistic settings—cloud execution, iterative/interactive workloads and concurrent jobs—circuits do not arrive in a single batch but as a stream. This creates a fundamentally online problem: the system must react to stochastic arrivals, balance latency and throughput, and remain stable under varying loads without foresight of future jobs.**

We study *Online Scheduling and Distribution of Quantum Circuits* (OS-DQC) and formalize the online scheduling problem for quantum networks: circuits arrive over time and must be executed without knowledge of future arrivals. At each *event time* (arrival or completion), the system decides which waiting circuits to launch and how to place their qubits, while accounting for congestion from in-flight circuits and constraints due to finite memories and decoherence. The objective is to minimize the mean completion time and makespan under these physical and resource constraints.

I. Introduction

Quantum networks are emerging as the connective tissue for distributed quantum computation: small processors linked by photonic channels can share entanglement and coordinate nonlocal operations across distance. Near-term deployments remain modest and multi-tenant, so workloads look less like coordinated experiments and more like a stream of independently issued jobs. Circuits appear at arbitrary times, vary in size and urgency, and must coexist on scarce quantum memories and limited, unreliable links. Without advance knowledge, the scheduler must decide in real time which circuits to execute and where to place them, maintaining low latency and high throughput while keeping the platform stable as demand fluctuates.

We therefore take an online view: arrivals, contention, and progress coevolve in real time, and performance is governed not only by circuit structure but also by admission and placement choices, coherence-aware batching of on-demand entanglement, and congestion management on the shared links.

Prior Work. The foundational results on quantum repeaters and long-distance entanglement established the substrate for networked quantum computation [1]. Vision and systemization work toward a quantum internet that articulates layered designs in which link-level entanglement services support network and application control, with prototypes validating these interfaces [2, 3, 4]. On the network side, the entanglement distribution has been studied under probabilistic links and finite memories, with algorithms that optimize distribution (routing, fusion, purification) and resource usage in the presence of stochastic success [5, 6, 7]. These efforts characterize the communication substrate and the constraints that any higher-level scheduler must respect.

In parallel, the DQC and compilation communities have explored placement, routing, and partitioning across devices, including qubit routing and architecture-aware mapping [8, 9, 10, 11], compiler and runtime support for multi-device execution [12, 13, 14, 15], and entanglement-assisted distribution mechanisms over general network topologies [16, 17, 18]. Closest to our setting, a recent batched multi-circuit study considers efficient execution when all jobs are known in advance [19]; here we address streaming arrivals and study online admission and execution without lookahead. Classical scheduling offers latency-oriented guidance, for example, the optimality of the shortest remaining processing time in exact sizes, while our service times are shaped by congestion and coherence limits [20]. Finally, placement objectives in DQC naturally connect to quadratic assignment formulations, for which approximation and heuristic results are available [21].

Our Contributions. In this paper, we formulate and address the problem of online scheduling and distribution of quantum circuits. Our key contributions are as follows.

- *Event-Driven Online DQC with an Oracle.* We snapshot the in-flight state, maintain congestion-aware routing, and compute placements on currently free memories. An oracle maps any candidate launch set to predicted per-circuit finishing times and the latest finishing time, keeping the policy layer independent of single-circuit mechanics.
- *Online Admissions.* We implement three rules—FIFO; Shortest (earliest predicted finish); MMFT (smallest marginal tail)—and admit one circuit at a time, refreshing after each commit until no fit remains. Our evaluation also compares these with an offline global scheduler (GreedySC-Heuristic).
- *Unified EP-Multiset Batching.* We unify unfinished EPs with those induced by newly admitted circuits into a single multiset; partition into sequential batches that respect memory/channel capacities and link concurrency; and order batches so that every EP is consumed within the decoherence threshold.

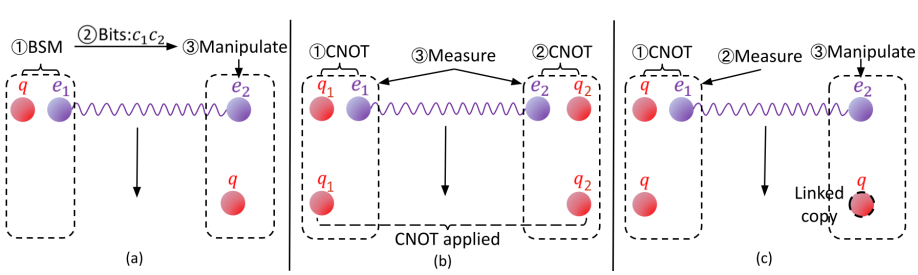

Fig. 1: Quantum communication. (a) Teleportation, (b) Telegate, (c) Cat-Entanglement. Dashed boxes are the network nodes; the initial (final) state of the qubits is at the top (bottom).

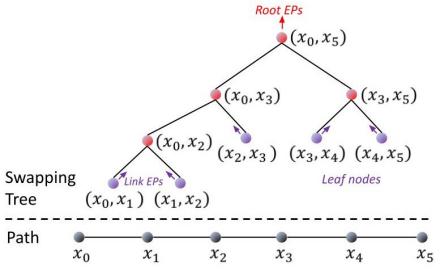

Fig. 2: Swapping tree to generate remote EPs.

II. Background

Quantum Circuit and Network. We represent a quantum circuit C over a set of qubits $Q = \{q_1, q_2, \ldots\}$ as an ordered list of gate tuples: A single-qubit/unary gate on q_i at the sequence index $\ell \in \{1, 2, \ldots\}$ is written as (q_i, ℓ), and a two-qubit/binary gate on (q_i, q_j) at the sequence index ℓ is (q_i, q_j, ℓ). The sequence index ℓ gives a linear order consistent with dependencies: gates that act on the same logical qubit appear in program order, and any gate that is conditioned on a prior measurement appears after that measurement. The quantum network is a connected graph $G = (V, E)$. Each node $u \in V$ is a quantum computer (QC) with M_u quantum memories. Each edge $e \in E$ denotes a quantum channel between its incident nodes, with an associated classical side channel.

Qubit Assignment. A placement η assigns each circuit qubit to exactly one quantum memory in the network. For a binary gate (q_i, q_j, ℓ), let u be the node that hosts $\eta(q_i)$ and v be the node that hosts $\eta(q_j)$. The gate is *local* if $u = v$ and *remote* otherwise.

Remote Gate Primitives. For any *remote* gate (q_i, q_j, ℓ) whose operands are hosted at nodes u and v under μ, we realize the interaction using one of three standard primitives (see Figs. 1(a)–(c)). A maximally-entangled pair (EP) between nodes u and v, denoted $\mathrm{EP}(u, v)$, is a shared Bell pair across the endpoints; consuming one EP enables a single instantiation of any primitive below.

Teleportation. This primitive relocates a data qubit to co-locate the operands: create $\mathrm{EP}(u, v)$, perform a Bell-state measurement at u, send two classical bits to v, and apply Pauli corrections so that the logical qubit now resides at v [22]. See Fig. 1(a). Each teleportation consumes one EP; teleporting the qubit back later requires an additional EP.

Telegate. This primitive enacts a remote CNOT/CZ in place without relocating data qubits [13]: create $\mathrm{EP}(u, v)$, have each side locally couple its data qubit to its half of the EP, then measure the EP halves, and apply the corresponding Pauli corrections on the data qubits to complete the nonlocal gate. See Fig. 1(b). Each telegate invocation consumes one EP.

Cat–Entanglement (Linked Copy). This primitive creates a read-only linked copy of a control qubit to one or more target qubits using shared entanglement; subsequent fan-out CNOT/CZ gates with the *same* control reuse the link until a single-qubit gate acts on the control or the link expires [23, 24]. See Fig. 1(c). Each link setup consumes one EP per control–target pair; reusing the link for additional gates does not consume additional EPs until disentanglement/expiration. When a unary operation needs to be executed on the control qubit, a disentanglement operation is performed, which destroys the linked copies; afterwards, the unary operation is applied.

Remote Gate Primitive Selection. All three primitives consume EPs: teleportation and telegate use one EP per invocation, while a cat–entanglement uses one EP per control–target link in setup and can be reused across multiple gates. Teleportation transports one operand to the node where follow-up operations can proceed locally—useful when many subsequent gates involve that qubit at the destination. Telegate executes the remote gate in place without relocating data qubits—well suited for one-off interactions or when moving a qubit would later force additional moves. Cat–entanglement amortizes its setup over several two-qubit gates that share the same control; it is particularly natural for CZ-only subcircuits, and the link must be torn down before any subsequent unary gate on the control (after which a fresh setup would be required).

Generating EPs over Remote Nodes. One simple way to distribute an EP between nodes u and v is to create a Bell pair locally and physically send its two halves to u and v; however, direct transport of qubits over long distances is highly lossy and can introduce irrecoverable errors. Instead, for each remote gate between u and v, we select a path $p : u \rightsquigarrow v$ in G and use *entanglement swapping*: first pump each edge on p to obtain fresh short-range Bell pairs (and, if needed, extra pairs for purification), then perform Bell-state measurements at intermediate nodes to make them into an end-to-end $\mathrm{EP}(u, v)$. For example, Fig. 2 illustrates a swapping tree that generates an EP between x_0 and x_5 using pairs along the path $x_0, x_1, x_2, x_3, x_4, x_5$. Different swapping trees (orders/groupings of swaps) on the same path can incur different generation latencies [25, 26]. The selection of swapping trees may be guided either by a baseline shortest-path cost derived from physical link parameters or by a congestion-aware effective path cost that increases with current load.

Generation Latency of EPs. EP generation is stochastic at the link level and uses shared network resources along p, so multiple binary gates may contend when scheduled together. When generating several EPs concurrently, these shared resources must be allocated across requests, which in turn affects completion time.

Decoherence and Fidelity of EPs. *Fidelity* measures how close a realized state is to the ideal and degrades with storage

time and operations; the time-driven component is called *decoherence*. We enforce a *decoherence threshold* τ: an EP created at time t must be consumed by $t+\tau$. To counteract low fidelity, we employ *entanglement purification*, which probabilistically distills a higher-fidelity pair from multiple raw EPs at the cost of additional pairs and delay [1, 7].

III. Problem Formulation

We start by defining the key terms and models used throughout this paper; with these in place, we then present the formal problem statement and objectives.

Memory Graph and Computers. We model the network at the *memory* level using a weighted undirected graph

$$G_M = (V_M, E_M, w, \mathrm{cap}),$$

where each $v \in V_M$ is a quantum memory; each $e = (u, v) \in E_M$ is a link on which Bell pairs (EPs) can be generated; $w(e) > 0$ is a base per-link cost (e.g., a distance/latency proxy); and $\mathrm{cap}(e) \in \mathbb{Z}_{\geq 1}$ is the number of parallel channels on e. Memories belong to quantum computers (QCs); an edge is *remote* if it connects memories on different QCs and *local* otherwise. Throughout, the cost of *local* (intra-QC) gates/paths is negligible compared to the remote generation of EP.

Congestion-Aware Effective Distances. Running circuits consumes EPs and thereby increases link load. Let $y_e \in \mathbb{Z}_{\geq 0}$ be the number of *unfinished* EPs whose currently planned routes include edge e. We inflate the base weight $w(e)$ by a dimensionless congestion parameter $\beta \geq 0$:

$$w_{\text{eff}}(e) \;=\; w(e)\Big(1 + \beta\,\frac{y_e}{\mathrm{cap}(e)}\Big).$$

For any memory pair (u, v), the effective distance is the shortest-path metric on these inflated weights,

$$B_{\text{eff}}(u, v) \;=\; \min_{P \in \mathcal{P}(u,v)} \sum_{e \in P} w_{\text{eff}}(e),$$

where $\mathcal{P}(u, v)$ is the set of simple paths that join u and v. $\beta = 0$ recovers the base metric B, and a larger β makes links of high use relatively more expensive. When focusing on *remote* communication, intra-QC edges are given zero weight and carry no load contribution (i.e., $w_{\text{eff}}(e) = 0$ and $y_e = 0$ for local edges), so only inter-QC paths affect B_{eff}. This distance is then used to locate the highly interacting QC neighbors and to steer new EP routes away from congested links.

Circuits and Flow. Let n be the number of circuit qubits, and let $G_2(C)$ be the collection of two-qubit gate instances for the circuit C. We construct the symmetric flow matrix $A(C) \in \mathbb{Z}_{\geq 0}^{n \times n}$ as follows:

1) Initialize $A(C)$ as the zero matrix $n \times n$.
2) For each gate instance $g \in G_2(C)$ acting on qubits (i, j) with $i \neq j$, increase $A_{ij}(C)$ and $A_{ji}(C)$ by one.

By construction, A is symmetric with a zero diagonal and records pairwise interaction counts. Equivalently, for any distinct i and j, $A_{ij}(C)$ equals the number of two-qubit gates in C that act on the qubits $\{i, j\}$. This summary is the input to the assignment of the qubits: a larger $A_{ij}(C)$ indicates a stronger interaction between i and j and encourages them to be closer together under B_{eff}.

Time and Arrivals. Each circuit C arrives at time t_c generated by a homogeneous Poisson process with rate $\lambda > 0$. Equivalently, if $t_1 < t_2 < \cdots$ are arrival times, the inter-arrival gaps $\Delta_k := t_k - t_{k-1}$ are i.i.d. $\mathrm{Exp}(\lambda)$ with $\mathbb{E}[\Delta_k] = 1/\lambda$, and the number of arrivals in $[0, t]$ is $\mathrm{Poisson}(\lambda t)$. Arrivals are *exogenous* (independent of scheduling and execution). Upon arrival, C joins the waiting pool until admitted. We record its start time s_c at admission and its completion time e_c.

Qubit-Allocation Function μ. At time t, the qubit-allocation function for circuit C is denoted μ_t[1]. It returns an *injective* map

$$\mu_t(C) :\; Q(C) \hookrightarrow V_M^{\text{free}}(t).$$

Here $Q(C)$ is the set of qubits of C, and $V_M^{\text{free}}(t) \subseteq V_M$ are the memories not reserved by the active circuits at time t. Feasibility requires $|Q(C)| \leq |V_M^{\text{free}}(t)|$; otherwise, C is deferred. Once admitted, η_c remains fixed until C completes (no migration). For completeness, Section VII also explores a limited dynamic reassignment (e.g., teleport-and-carry at batch boundaries) that preserves the decoherence window τ; unless stated otherwise, our baseline results use fixed placements.

Online Scheduling and Distribution of Quantum Circuits. (`OS-DQC`). Given the quantum network with congestion-aware effective distances and an exogenous arrival process of circuits with arrival times, decide *online*, at each decision time t, the following for any waiting circuit C to minimize the makespan subject to network/fidelity constraints:

- *Admission / Start Time:* Defer C or admit with a start time $s_c \geq t_c$.
- *Qubit Allocation:* If admitted at s_c, choose an injective mapping $\mu_{s_c}(C)$.
- *Execution Scheme:* Construct a plan that schedules all remote EPs required by C in $\mu_{s_C}(C)$, choosing routes and swapping trees with respect to w_{eff} (i.e., distances in B_{eff}). The plan coordinates across all admitted circuits to respect channel capacities on shared links. EP scheduling follows the circuit's gate order: each EP is consumed when its corresponding two-qubit gate executes, and its generation is timed to occur within the decoherence window. See [18] for details on the consumption–generation coupling.

Objective. Our default objective is to minimize the makespan of all the circuits:

$$\min \;\max_c e_c,$$

and we also report the mean waiting time $\frac{1}{|\{C\}|}\sum_c (e_c - t_c)$ in the evaluation. All decisions are made online (without look-ahead), using only the information available up to each decision time t.

[1]Notation: μ_t is the *operator* (a function of the current state and the circuit C). When C is admitted at time s_c, we write the realized placement as $\eta_c := \mu_{s_c}(C)$, which is an injective map $Q(C) \hookrightarrow V_M^{\text{free}}(s_c)$. In formulas later, we sometimes write $\mu_t(C)$ when the output map itself is intended; otherwise, we use η_c to emphasize the committed placement.

Complexity. Even under simplified assumptions (single circuit, fixed B_{eff}, infinite channels and $\tau = \infty$), the *qubit-allocation* subproblem [18],

$$\min_{\eta_C} \mathsf{cost}\big(\eta_C \mid A(C),\, B_{\text{eff}}\big) = \min_{\eta_C} \sum_{i<j} A_{ij}(C)\, B_{\text{eff}}\big(\eta_C(i),\, \eta_C(j)\big),$$

is precisely the *Quadratic Assignment Problem* (QAP), a (strongly) NP-hard problem; thus, NP-hardness already arises at the allocation step and persists in the online setting.

IV. Related Work

DQC Primitives and Partitioning (Single Circuit). Nonlocal gates across QPUs are realized via teleportation and cat-entangler/disentangler constructions, which enable distributing a monolithic circuit over multiple devices [22, 24]. Modern DQC compilers formalize distribution as communication minimization: hypergraph partitioning of the circuit interaction graph [27], optimization of teleportation/ebit usage during distribution [28], and circuit partitioning strategies tailored for distributed execution [12]. These works primarily target *communication cost* (e.g., remote-gate count) for a *single known* circuit and typically assume on-demand remote operations with fixed costs.

Latency-/Time-Aware DQC and Distributed Scheduling. Beyond ebit/min-cut style objectives, time-aware approaches explicitly incorporate execution latency under connectivity and remote-operation costs; e.g., formulations that couple mapping with a latency model to reduce overall circuit time [18]. Other lines consider distributed execution pipelines once a circuit is partitioned, proposing practical orchestration/scheduling mechanisms across devices [15, 16, 17]. Related systems work studies joint qubit placement and entanglement provisioning to reduce the completion time of a single DQC task; e.g., QuPEP [29] combines offline placement with online entanglement provisioning during task execution. At the quantum-network layer, Cicconetti et al. [30] study online scheduling of end-to-end entanglement requests under stochastic link availability, proposing FIFO-style variants for request selection. Overall, the above efforts focus either on entanglement-request scheduling or on single-job/fixed-set execution; instead, we study online admission of circuits under shared memory contention with decoherence-aware EP batching.

Positioning. Relative to the above, our work is an *online* multi-circuit formulation: we handle streaming arrivals, make joint admission/placement decisions under finite memories, use congestion-aware effective distances arising from in-flight jobs, and enforce generation–consumption coupling so nonlocal resources are consumed before decohering. This bridges DQC compilation with queueing-style constraints that prior single-circuit or offline methods typically abstract away.

V. High-Level Approach

We tackle OS-DQC with a single *event-driven loop* executed at each decision time.

Decision Time. A *decision time* t is the earliest instant at which (i) one or more new circuits arrive, or (ii) one or more running circuits complete. Events with the same timestamp are processed together.

EP Batches. An *EP batch* is a stage of EP generation in which tasks run concurrently; batches run sequentially and are chosen to satisfy the decoherence threshold τ.

Unified Loop at Time t. Let $\mathcal{U}(t)$ be the unfinished EP tasks after closing any circuits that are completed at t, and let $\mathcal{W}(t)$ be the waiting pool (not-yet-admitted circuits with arrival time $\leq t$).

1) *Snapshot State.* Snapshot the current network state (G_M, B_{eff}); block memories already in use by $\mathcal{U}(t)$; rebuild B_{eff} to reflect the load of $\mathcal{U}(t)$ and determine the currently free memories.
2) *Select Bundle.* Initialize $S \leftarrow \emptyset$. Repeatedly add *one* circuit from $\mathcal{W}(t)$ using the admission and packing algorithms of Section VI, maintaining feasibility on the currently free resources and respecting τ. Each add-candidate evaluation invokes the *prediction oracle* below (Step 3), which returns a placement and a cached batch plan (hence a predicted completion time) under the current snapshot. Stop when no feasible or beneficial addition remains.
3) *Place and Schedule (Prediction Oracle).* For each $c \in S$, compute the placement of the qubits η_c in the free memories with respect to B_{eff}. Build the new EP demand $\mathcal{E}(S)$ and compute a batch sequence for $\mathcal{U}(t) \cup \mathcal{E}(S)$ starting at t.
4) *Commit Outputs.* For each admitted circuit, record its placement η_c and predicted completion time. The cached batch plan is committed without recomputation; any work extending beyond t becomes the updated unfinished set $\mathcal{U}(t^+)$; the next decision time is the earliest of the next arrival or the next completion.

VI. Online Admission and Packing Algorithms

We follow the event-driven runtime in Section V: at each decision time t, we repeatedly rank/evaluate candidates by invoking the prediction oracle (Step 3 of the Unified Loop) and then commit a chosen circuit by adopting its cached plan without recomputation.

Prediction Oracle. At decision time t, the prediction oracle evaluates a candidate circuit and returns (i) a heuristic qubit placement η for the QAP objective and (ii) a cached EP-batch plan for executing the candidate together with any currently unfinished EP work. The predicted completion time[2] (and tail when needed) is read from this cached plan under η. All oracle inputs are available online at time t (e.g., topology/link parameters, the current unfinished-task set, and the currently free memories), and it assumes no knowledge of future arrivals or future stochastic outcomes.

[2]Because EP latencies are stochastic, predicted completion times are approximate. However, decisions are re-made at every event time (arrival/completion), so errors do not accumulate across long horizons; the policies primarily rely on the *relative* ranking of candidates under the same oracle, and the system corrects course as actual completions trigger the next decision.

Algorithm 1. FIFO Admission (FIFO). Admit circuits strictly by arrival time, provided that each circuit can start *now*. The goal is predictable, low-overhead operation and fairness to arrival order. The steps are as follows:

1) *Select Circuit.*
   - Scan the waiting pool in arrival order.
   - Choose the first circuit whose qubit demand fits the currently free memories at t.
2) *Commit.*
   - Run a single prediction for the selected circuit at t to obtain its cached plan and finish time.
   - Commit to the network using the cached plan.
3) *Iterate.*
   - Refresh the snapshot.
   - Repeat until no arrival-ordered circuit is feasible.

Algorithm 2. Shortest-Time Packing (Shortest). Greedily favor circuits that *finish soonest under current congestion*. This targets high instantaneous throughput and reduces head-of-line blocking by launching "easy" jobs first. The steps are as follows:

1) *Pick Candidates.*
   - Gather all waiting circuits that fit the currently free memories at t.
2) *Select Circuit.*
   - For each candidate circuit:
     - Run one prediction at t to obtain its predicted latency.
   - Select the circuit with the smallest latency[3]; break the ties by earlier arrival.
3) *Commit.*
   - Adopt the cached plan of the selected circuit.
4) *Iterate.*
   - Refresh the snapshot.
   - Repeat until no feasible candidate.

Algorithm 3. Greedy Min–Marginal Finishing Time (MMFT). Minimize the *one-step increase in the current makespan*. `MMFT` compares the marginal effect of each candidate on the remaining completion time of active circuits and admits the circuit that least extends the tail of the system, balancing throughput with tail-latency control and avoiding unnecessary congestion spikes. The steps are as follows:

1) *Pick Candidates.*
   - Gather all waiting circuits that fit the currently free memories at t.
2) *Select Circuit.*
   - Establish the baseline tail T_n: the remaining time until all currently running circuits are finished.
   - For each candidate:
     - Predict the resulting tail T_c if it starts now in the same snapshot t.
     - Compute its marginal increase $T_c - T_n$.
   - Choose the circuit with the smallest marginal increase; break the ties by earlier arrival.
3) *Commit.*
   - Adopt the cached plan of the selected circuit.
4) *Iterate.*
   - Refresh the snapshot.
   - Repeat until no feasible candidate.

[3]To prevent starvation under sustained load, one can add a max-wait override: If the waiting time of any circuit exceeds $W_{\max}$, admit it by arrival order regardless of the predicted finish time; our evaluation uses finite workloads and did not require this safeguard.

Oracle Overhead and Scalability. The policies differ mainly in how many oracle evaluations they perform per decision time: `FIFO` evaluates at most the first feasible arrival-ordered candidate, whereas `Shortest`/`MMFT` may evaluate multiple feasible candidates at the same t to obtain predicted completion times (and `MMFT` marginal tails). If $K(t)$ candidates are evaluated at time t, the oracle overhead scales linearly with $K(t)$ and can be bounded by limiting the candidate set (e.g., an arrival-ordered prefix) and reusing cached predictions within the same decision time.

VII. Event-Time Generalizations for Online DQC

We present two event-time generalizations that (i) leverage qubit teleportation to adjust placements during execution and (ii) optionally defer launches when free capacity is scarce and dispersed across many QCs, favoring compact placements over fragmented ones. Both operate locally at the decision time t and integrate with the same event-driven loop and EP batching used throughout.

Teleport-and-Carry. Allow limited, well-scoped relocation of qubits via teleportation at decision times to relieve hotspots, shorten effective paths in B_{eff}, and consolidate activity where helpful, while keeping already-issued work intact.

Action at t. Consider moves only at EP batch boundaries or when a clear benefit is predicted (e.g., lower tail or freed bottleneck). Select a small set of qubits with the highest gain-to-cost ratio, teleport them to nearby free memories (prefer intra-QC or short-hop), and update the placement η and the future EP demand for gates that have not yet been generated.

Fragmentation-Aware Admission. Aggressive packing into many small, scattered free slots increases fragmentation and later contention. At time t, if free memories are scarce and spread over many QCs, we may defer the launch even when a circuit technically fits, favoring placements that use fewer QCs and larger contiguous blocks.

Action at t. Summarize the free capacity by QC (e.g., largest block per QC, number of QCs with free slots, dispersion across QCs). Admit only if the newcomer can be placed with a small QC footprint or sufficiently large per-QC blocks; otherwise wait for the next event so completions can consolidate capacity. Prefer launches that preserve large blocks and avoid cross-QC stitching.

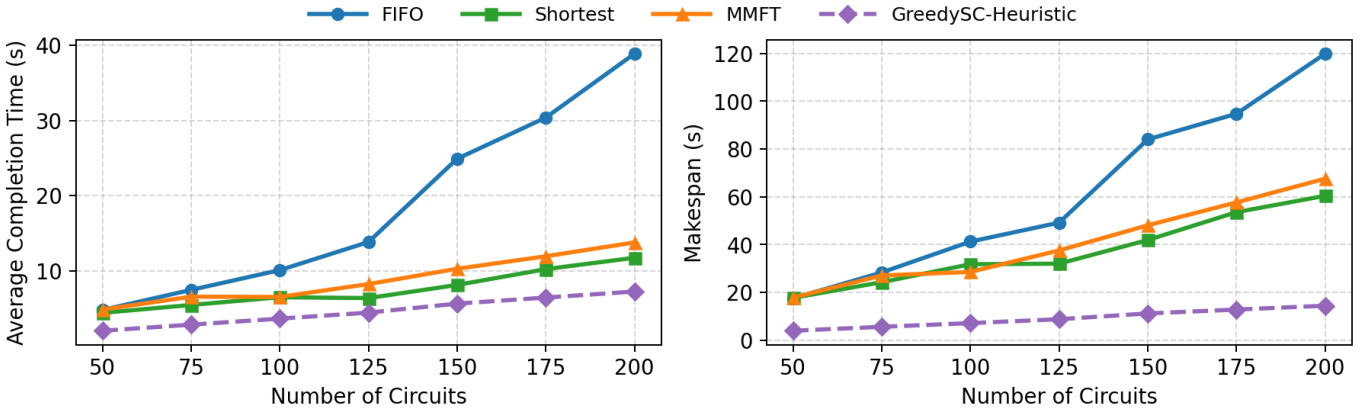

Fig. 3: The total number of circuits per stream is varied. Panels show (a) average completion time and (b) makespan. Arrivals followed a Poisson process and all other network and runtime parameters were held fixed. Curves compared `FIFO`, `Shortest`, and `MMFT` within the same event-driven loop for the online case. `GreedySC-Heuristic` was included as the offline baseline for comparison.

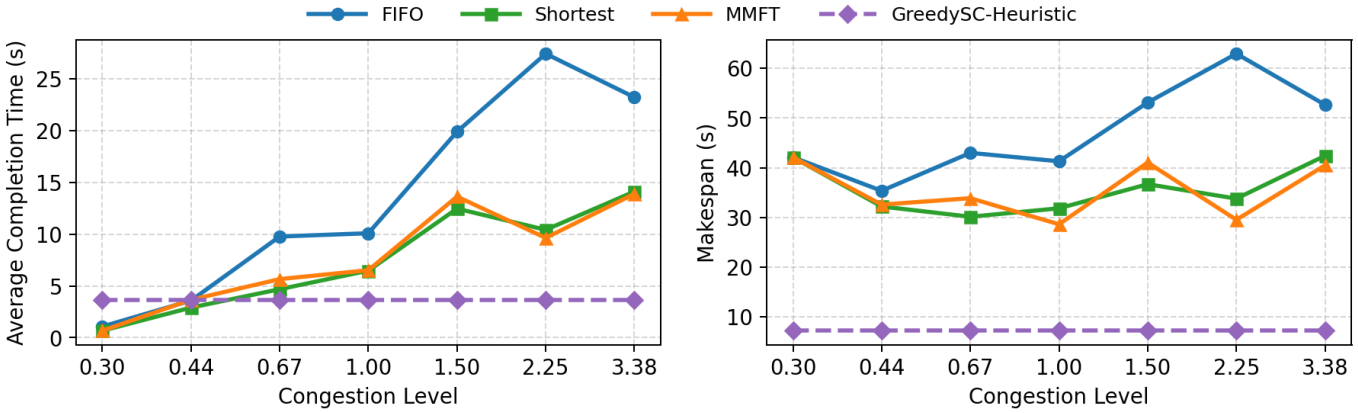

Fig. 4: The offered load is varied by scaling the arrival rate (reported as a congestion level on the x-axis; higher values indicate heavier load). Panels show (a) average completion time and (b) makespan. Curves compare `FIFO`, `Shortest`, and `MMFT` for the online case, with `GreedySC-Heuristic` included as the offline baseline for comparison; the arrival rate does not affect the offline case.

VIII. Evaluation

In this section, we evaluate the `OS-DQC` scheduler developed in Secs. V–VI under the full execution model: congestion-aware distances, finite memories, stochastic EP generation with batching and a decoherence threshold. We compare admission/launch policies that share the same arrival stream of circuits and the same EP-generation back end, and we focus on completion-centric metrics (mean completion time and makespan). We use NetSquid [31] as the quantum-network simulator to capture probabilistic link behavior, swapping across paths, channel concurrency, and qubit decoherence.

Generating Random Network. The network has 10 quantum computers, each with 10 quantum memories (total 100 memories). We generate a Waxman host graph [32] by placing the computers uniformly at random in a $100\,\mathrm{km} \times 100\,\mathrm{km}$ square and creating distance-biased links.

Network Parameters. Following [25], we set: atomic-BSM success probability 0.4 with latency $10\,\mu$s; optical-BSM success probability 0.3; atom–photon pair generation time $50\,\mu$s with success probability 0.33; and a decoherence threshold of 1 s.

Generating Random Circuits. For the incoming circuits, we generate random quantum circuits with four knobs: number of circuits (default 100), qubits per circuit (uniform in 20–30), gates per qubit (default 50), and the fraction of two-qubit gates (fixed at 0.5). Each circuit is built gate by gate: at each step, the gate type is chosen at random. Operands are then selected uniformly (two distinct qubits for two-qubit gates; one qubit otherwise). For evaluation, we balance the size distribution by giving equal probability to each qubit count in 20–30, and we reuse the identical circuit set across all policies.

Choosing the Poisson Arrival Rate. We set the arrival rate λ in three steps.

1) *Probe Time.* Run a randomly sampled 20-qubit circuit and a 30-qubit circuit on the network and average their completion times; denote this average by T_p.
2) *Baseline Rate.* Set the default Poisson rate to
$$\lambda_0 = \frac{1}{0.75\,T_p},$$
i.e., the mean inter-arrival gap is $0.75\,T_p$.

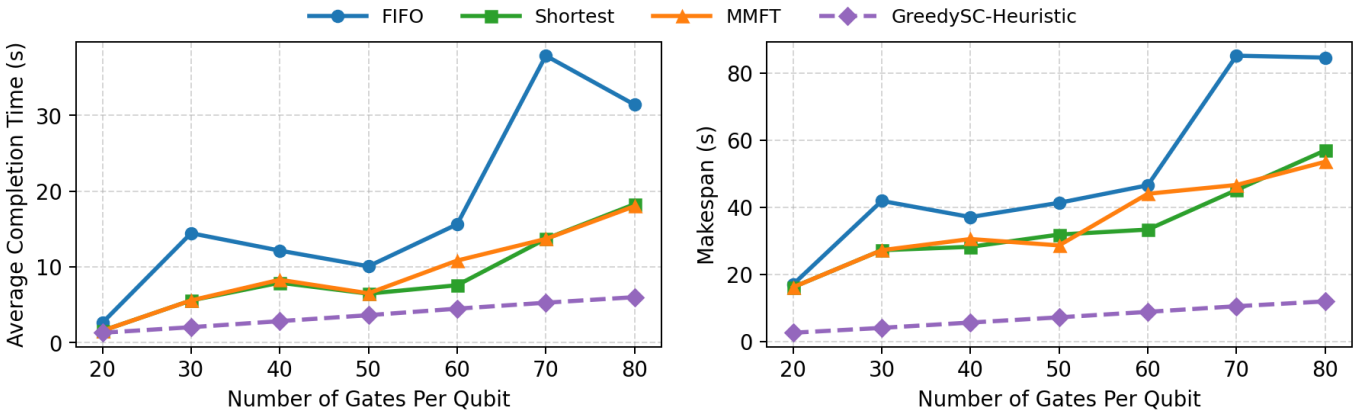

Fig. 5: The number of gates per qubit is varied. Panels show (a) average completion time and (b) makespan. Each point aggregated results from circuits constructed at the indicated gate count under identical network and arrival settings. Curves compared `FIFO`, `Shortest`, and `MMFT` for the online case, with `GreedySC-Heuristic` included as the offline baseline for comparison.

3) *Arrival Intensity Control.* Introduce an arrival rate multiplier $r > 0$—also shown as the *congestion level*—to vary traffic:
$$\lambda = r\,\lambda_0 \qquad \text{(equivalently, mean gap } = 0.75\,T_p/r\text{)}.$$
The larger r means more traffic and more congestion.

Algorithms Compared. The evaluation compared three *online* admission/launch policies—`FIFO`, `Shortest`, and `MMFT` —together with an *offline* baseline, `GreedySC-Heuristic` [19]. The online policies use a common snapshot and prediction oracle at each decision time, whereas the offline baseline has full lookahead over the entire workload and is therefore insensitive to the arrival rate. `GreedySC-Heuristic` is included because, in that previous work, it provided the best performance among *polynomial-time* batching schedulers; the offline optimal scheduler there is exponential-time and thus not suitable as a per-decision baseline in our online setting.

Metrics. For online policies, the metrics reported are the average completion time $\frac{1}{n}\sum_{c=1}^{n}(e_c - t_c)$ and makespan $\max_{1\le c\le n} e_c$, where t_c and e_c are the arrival and completion times of the circuit C, and n is the total number of circuits. For the offline baseline, the average completion time is measured analogously with a common release time (all $t_c \equiv 0$), i.e., $\frac{1}{n}\sum_{c=1}^{n} e_c$; consequently, the chosen arrival rate (congestion level) does not affect the offline metric.

Evaluation Results. Each experiment varied one parameter at a time, while others remained at their defaults. We observe the following:

1) Circuit Number. Both metrics increase with circuit number. `Shortest` is consistently best across the plotted settings, while `FIFO` is worst under growing contention. `GreedySC-Heuristic` (offline) remains below all online curves, with a widening gap at a larger circuit number.
2) Congestion Level. Both metrics increase with arrival rate. `Shortest` and `MMFT` perform similarly, with `MMFT` showing larger fluctuations[4]; `FIFO` is worst. `GreedySC-Heuristic` is insensitive to the rate. At low congestion, `GreedySC-Heuristic` may slightly underperform the online policies because its global batching can introduce avoidable waiting, whereas the online policies can admit and finish circuits sequentially with little queueing.
3) Gate Number. Both metrics increase with gate number. `Shortest` and `MMFT` are comparable; `FIFO` is the worst. `GreedySC-Heuristic` remains the best across the sweep.
4) Qualitative Robustness. Across all sweeps, the qualitative ordering was consistent. `GreedySC-Heuristic` (offline) generally formed a lower envelope; among the online policies, `Shortest` and `MMFT` were close- `Shortest` typically achieved the best average completion time (often best overall); `FIFO` consistently underperformed[5]. Gaps widened with increasing congestion, larger circuit numbers, and deeper circuits. At very low congestion, `GreedySC-Heuristic` 's global batching could be slightly worse than the online policies' sequential admission.

IX. Conclusion

We studied `OS-DQC` on the quantum network and presented an event–driven framework that maintains *congestion-aware routing* via B_{eff}, uses it to place qubits, and to batch EP generation on-demand under a decoherence window τ. We instantiated three online admissions—`FIFO`, `Shortest`, and `MMFT` —and compared them against an offline global scheduler (`GreedySC-Heuristic`). Across circuit, contention,

[4]`MMFT` can be more volatile because it optimizes a *marginal* predicted tail: when the system is near saturation, small snapshot changes (or small prediction differences between candidates) can flip the marginal comparison, leading to policy churn and higher variance even when mean performance is comparable.

[5]`FIFO` remains attractive in operational regimes where scheduler overhead must be minimal or strict arrival-order fairness is required, even if its completion-time metrics are worse under moderate/heavy contention.

and gate sweeps, `FIFO` underperformed; `Shortest` was often the best among the online algorithms. Overall, per-event prediction coupled with decoherence-aware batching yields robust online performance without lookahead.

REFERENCES

[1] H.-J. Briegel, W. Dür, J. I. Cirac, and P. Zoller, "Quantum repeaters: the role of imperfect local operations in quantum communication," *Physical Review Letters*, vol. 81, no. 26, p. 5932, 1998.

[2] S. Wehner, D. Elkouss, and R. Hanson, "Quantum internet: A vision for the road ahead," *Science*, vol. 362, no. 6412, p. eaam9288, 2018.

[3] A. Dahlberg, P. Skrzypczyk, T. Coopmans, J. Wubben, F. Rozpedek, M. Pompili, A. Stolk, P. Pawełczak, R. Knegjens, H. Mendes, D. Elkouss, R. Hanson, and S. Wehner, "A link layer protocol for quantum networks," in *Proceedings of ACM SIGCOMM*, 2019.

[4] M. Pompili, P. C. Humphreys *et al.*, "Experimental demonstration of entanglement delivery using a quantum network stack," *npj Quantum Information*, vol. 8, no. 1, p. 110, 2022.

[5] M. Pant, H. Krovi, D. Towsley, L. Tassiulas, S. Wehner, G. M. D'Ariano, I. Kerenidis, J. H. Shapiro, and S. Guha, "Routing entanglement in the quantum internet," *npj Quantum Information*, vol. 5, no. 1, p. 25, 2019.

[6] X. Fan, C. Zhan, H. Gupta, and C. R. Ramakrishnan, "Optimized distribution of entanglement graph states in quantum networks," *IEEE Transactions on Quantum Engineering*, vol. 4, pp. 1–16, 2023.

[7] X. Fan, Y. Yang, H. Gupta, and C. R. Ramakrishnan, "Distribution and purification of entanglement states in quantum networks," in *Proceedings of the IEEE International Conference on Quantum Communications and Networking (QCNC)*, 2025.

[8] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, "On the qubit routing problem," in *Theory of Quantum Computation, Communication, and Cryptography*, 2019.

[9] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the ibm qx architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2018.

[10] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, "Qubit allocation as a combination of subgraph isomorphism and token swapping," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–29, 2019.

[11] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "t-ket: a retargetable compiler for NISQ devices," *Quantum Sci. and Tech.*, 2020.

[12] O. Daei, K. Navi, and M. Zomorodi-Moghadam, "Optimized quantum circuit partitioning," *International Journal of Theoretical Physics*, vol. 59, no. 12, pp. 3804–3820, 2020.

[13] D. Cuomo, M. Caleffi, K. Krsulich, F. Tramonto, G. Agliardi, E. Prati, and A. S. Cacciapuoti, "Optimized compiler for distributed quantum computing," *ACM Transactions on Quantum Computing*, vol. 4, no. 2, feb 2023.

[14] D. Ferrari, A. S. Cacciapuoti, M. Amoretti, and M. Caleffi, "Compiler design for distributed quantum computing," *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1–20, 2021.

[15] R. G Sundaram, H. Gupta, and C. Ramakrishnan, "Efficient distribution of quantum circuits," in *35th International Symposium on Distributed Computing (DISC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[16] R. G. Sundaram, H. Gupta, and C. Ramakrishnan, "Distribution of quantum circuits over general quantum networks," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2022, pp. 415–425.

[17] R. G. Sundaram and H. Gupta, "Distributing quantum circuits using teleportations," in *2023 IEEE International Conference on Quantum Software (QSW)*. IEEE, 2023, pp. 186–192.

[18] R. G. Sundaram *et al.*, "Distributed quantum computation with minimum circuit execution time over quantum networks," in *IEEE QCE*, 2024.

[19] Y. Yang, R. G. Sundaram, and H. Gupta, "Efficient execution of multiple quantum circuits over a quantum network," in *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2025.

[20] L. Schrage, "A proof of the optimality of the shortest processing remaining time discipline," *Operations Research*, vol. 16, no. 3, pp. 678–690, 1968.

[21] E. M. Arkin, R. Hassin, and M. Sviridenko, "Approximating the maximum quadratic assignment problem," *Information Processing Letters*, vol. 77, no. 1, pp. 13–16, 2001.

[22] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels," *Physical review letters*, vol. 70, no. 13, p. 1895, 1993.

[23] J. Eisert, K. Jacobs, P. Papadopoulos, and M. B. Plenio, "Optimal local implementation of nonlocal quantum gates," *Physical Review A*, vol. 62, no. 5, p. 052317, 2000.

[24] A. Yimsiriwattana and S. J. Lomonaco, "Generalized ghz states and distributed quantum computing," *arXiv: Quantum Physics*, 2004.

[25] M. Ghaderibaneh, C. Zhan, H. Gupta, and C. Ramakrishnan, "Efficient quantum network communication using optimized entanglement swapping trees," *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–20, 2022.

[26] R. Sundaram and H. Gupta, "Optimized generation of entanglement by real-time ordering of swapping operations," 09 2024, pp. 1973–1979.

[27] P. Andres-Martinez and C. Heunen, "Automated distribution of quantum circuits via hypergraph partitioning," *Physical Review A*, vol. 100, no. 3, p. 032308, 2019.

[28] M. Zomorodi-Moghadam, M. Houshmand, and M. Houshmand, "Optimizing teleportation cost in distributed quantum circuits," *International Journal of Theoretical Physics*, vol. 57, pp. 848–861, 2018.

[29] F. Zhan, Y. Zhao, and C. Qiao, "Towards high-performance distributed quantum computing with qubit placement and provisioning," in *Proceedings of the IEEE/ACM International Symposium on Quality of Service (IWQoS)*, 2024.

[30] C. Cicconetti, M. Conti, and A. Passarella, "Request scheduling in quantum networks," *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 2–17, 2021, art. no. 4101917.

[31] T. Coopmans, R. Knegjens, A. Dahlberg *et al.*, "Netsquid, a network simulator for quantum information using discrete events," *Communications Physics*, 2021.

[32] B. M. Waxman, "Routing of multipoint connections," *IEEE journal on selected areas in communications*, vol. 6, no. 9, pp. 1617–1622, 1988.