# Full Exploitation of Limited Memory in Quantum Entanglement Switching

Panagiotis Promponas*, Víctor Valls†, and Leandros Tassiulas*
{panagiotis.promponas@yale.edu, victor.valls@ibm.com, leandros.tassiulas@yale.edu}
* *Department of Electrical Engineering and Institute for Network Science, Yale University, USA*
† *IBM Research Dublin*

***Abstract*—We study the problem of operating a quantum switch with memory constraints. In particular, the switch has to allocate quantum memories to clients to generate link-level entanglements (LLEs), and then use these to serve end-to-end entanglements requests. The paper's main contributions are (i) to characterize the switch's capacity region, and (ii) to propose a memory allocation policy (MEW) that is throughput optimal. The worst-case time complexity of MEW is exponential on the system parameters. However, when the requests are bipartite and the LLE attempts are always successful, we propose a variant of MEW (MEW2) that has polynomial time complexity. We evaluate the proposed policies numerically and illustrate their performance depending on the requests arrivals characteristics and the time available to obtain a memory allocation.**

## I. Introduction

Quantum computing will transform the world by allowing us to solve problems that are too complex for classical computers [1] (e.g., Shor's algorithm [2]). However, we are still nowhere near that day. Quantum programs of meaningful size require quantum computers with thousands of qubits [3], which is far from the number of qubits that quantum computers currently have [4], [5].

One way to increase the number of qubits of a quantum computer is to connect multiple quantum processors [6], [7], [8] with a quantum switch. In brief, a quantum switch is analogous to a classic packet switch, but its task is to create end-to-end entanglements with the clients it is connected. Figure 1 shows an example of how a quantum switch operates. The switch first generates link-level entanglements (LLEs)[1] with the clients/processors (Figure 1b), and then it uses these to create end-to-end entanglements (Figure 1c & d).[2] The end-to-end entanglements are used by the quantum applications to, for example, teleport information qubits or carry out distributed quantum operations (via non-local CNOT gates [9]).

Quantum networking is in its infancy since single-hop communications are still challenging [12]. However, the building blocks of how quantum networks will operate already exist, prompting researchers to start designing the algorithms that will run the networks when the hardware becomes available [13], [14], [15], [16], [17]. Regarding quantum switches, previous work has studied their operation under a variety of settings [18], [19], [20], [21]. In brief, [18] and [19] study an idealized switch with bipartite and tripartite end-to-end entanglements requests when the request arrivals are symmetric and decoherence [22] is negligible. The work in [20] studies a quantum switch with bipartite requests when there is no memory decoherence and LLE attempts succeed probabilistically. The contributions of [20] are to characterize the switch capacity region and to propose on-demand policies that are throughput optimal. Similarly, the recent work in [21] extends the setting in [20] to capture that LLEs expire (i.e., "decohere") after some time in practical systems. In sum, previous work has focused *primarily* on studying quantum switches for different decoherence models. However, none of them consider that quantum memory is a scarce resource that must be managed. In practice, quantum switches can only store a limited number of qubits (in analogy to quantum computers), constricting the LLEs that can exist at a given time and, therefore, the requests that the switch can serve.

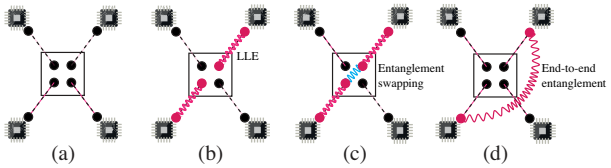



Figure 1. Illustrating the operation of a quantum switch. (b) LLEs are created between the switch and the clients (e.g., quantum processors). (c) The switch performs an entanglement swapping operation. (d) An end-to-end entanglement is created as a result. An entanglement swapping operation consists of performing a joint BSM measurement with the qubits at the switch.

In this paper, we study the problem of operating a quantum switch when it can store a limited number of qubits. In particular, the switch has fewer quantum memories than the number of clients it is connected, and so it has to decide how to allocate quantum memories to generate LLEs. Studying this problem is important because memory is a scarce resource in practical quantum systems. To this end, this paper makes the following contributions:

- We present the first mathematical model of a quantum switch that has to operate with fewer quantum memories

The research work was supported by the Army Research Office MURI under the project number W911NF2110325 and by the National Science Foundation under project numbers EEC-1941583 CQN ERC and CNS 1955744.


[1]Also known as EPR pairs. A LLE or EPR pair consists of two entangled qubits [9]. One qubit at the switch and the other qubits at the client.

[2]An end-to-end entanglement is created by performing a measurement (BSM or GHZ) on the qubits at the switch [10]. The process is also known as entanglement swapping when the requests are bipartite [11], [12].

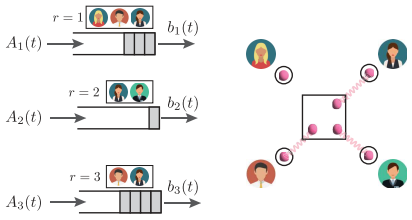



Figure 2. Example of a quantum switch with three types of requests. The switch is connected to three users, which only allows it to serve requests of type 2 and 3.

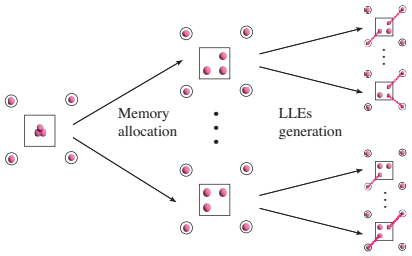



Figure 3. Illustrating how the quantum memory allocation results in different possible connectivities in a quantum switch with $M = 3$ and $N = 4$. Observe from the figure that different memory allocations can result in the same switch connectivity.

than the number of clients it is connected (Section II). Our model allows LLEs to decohere and end-to-end entanglement requests to be multipartite.

- We characterize the capacity region of the quantum switch with memory constraints (Section III-A), i.e., the set of end-to-end entanglement request arrival rates for which there exists a policy that can stabilize the switch.
- We propose a memory allocation policy (MEW) that stabilizes the switch when (i) the LLEs last one time slot and (ii) the arrivals of end-to-end entanglement requests are in the interior of the capacity region (Section III-B). Finding a throughput optimal policy in this setting is challenging because the admissible scheduling decisions depend on the memory allocation. Such coupling is typically not allowed in classic networking problems (e.g., wireless) where the set of available actions can vary over time, in an i.i.d. manner [23] (see discussion after Theorem 1).
- We present MEW2, a polynomial time variant of MEW tailored to the case where end-to-end entanglements are bipartite and LLE attempts are always successful. This case is important since multipartite requests can be divided into multiple bipartite requests (universality of two-qubit gates [22]) and because, with sufficient entanglement distillation, LLEs attempts succeed almost surely [24].

Finally, in Section V, we evaluate MEW and MEW2 numerically depending on the requests arrivals characteristics and the time available to obtain a memory allocation.

## II. Quantum Switch Model & Operation

### A. Switch model and operation overview

We consider a quantum switch with $M$ quantum memories and $N$ clients that operates in slotted time. In each time slot $t = 1, 2, 3, \ldots$, the switch receives a vector of requests

$$A(t) = (A_1(t), \ldots, A_R(t)),$$

where $A_r(t) \in \{0, 1\}$ for all $r \in \{1, \ldots, R\}$. A request $A_r(t)$ involves connecting two or more clients (i.e., it is multipartite), and we use set

$$\Omega(r) \subseteq \{1, \ldots, N\} \tag{1}$$

to denote the clients that participate in a request. For example, $\Omega(r) = \{1, 2\}$ if a request of type $r$ connects clients 1 and 2.

Upon arrival, the requests are stored in separate queues $Q(t) = (Q_1(t), \ldots, Q_R(t))$ to await service. The queues evolve as follows:

$$Q(t+1) = [Q(t) - b(t)]^+ + A(t), \tag{2}$$

where $[\cdot]^+ := \max\{0, \cdot\}$ and

$$b(t) = (b_1(t), \ldots, b_R(t))$$

indicates the requests served in time slot $t$. In particular, $b_r(t) = 1$ if a request $r \in \{1, \ldots, R\}$ is served, and $b_r(t) = 0$ otherwise.

The switch's task is to serve as many requests as possible subject to operational constraints. In particular, the switch can only serve a request if all the clients that participate in it have an active LLE. Figure 2 shows an example of a switch with four clients and three types of requests $r \in \{1, 2, 3\}$. Observe from the figure that the switch can serve requests of type 2 and 3, but not of type 1 because one of the clients is not connected with the switch.

In the next section, we describe how the switch allocates quantum memories to clients and how that affects the switch connectivity and the set of admissible service vectors.[3]

### B. Switch operation and decision variables

In each time slot, the quantum switch performs three types of actions. It (i) allocates quantum memories to clients; (ii) generates LLEs; and (iii) serves multipartite requests by using the LLEs. Importantly, a LLE can only be used to serve one request as this is consumed to generate an end-to-end entanglement [19]. Next, we describe the control variables that the switch can select in each time slot.

*1) Quantum memory allocation:* When $M < N$, the switch has to decide how to allocate memories to clients. We use $m_n(t)$ to denote whether the switch assigns a quantum memory to a node $n \in \{1, \ldots, N\}$ in time slot $t$, and collect these in vector

$$m(t) = (m_1(t), \ldots, m_N(t)).$$

[3] i.e., the requests that can be served given a switch connectivity.

The set of eligible memory allocations $\mathcal{M}$ is given by

$$\mathcal{M} = \Bigg\{ (m_1, \ldots, m_N) : m_n \in \{0,1\} \ \forall n \in \{1, \ldots, N\} \text{ with } \sum_{n=1}^{N} m_n \leq M \Bigg\}.$$

That is, $\mathcal{M}$ contains the binary vectors whose components' sum is smaller than or equal to the number of memories available (i.e., $M$).

*2) LLEs generation and switch connectivity:* After the memory allocation, the switch has to generate LLEs with the clients that are connected to a memory. The switch attempts to create LLEs by sending entangled qubits (e.g., photons) over a fiber-optical channel, but only a fraction of the LLE attempts are successful due to interference [25]. Also, LLEs last for a limited amount of time due to a phenomenon known as decoherence [26].

We model the switch connectivity in a time slot as follows. Let $p_n \in [0,1]$, $n \in \{1, \ldots, N\}$ be the probability that a LLE attempt succeeds. Vector

$$k(t) = (k_1(t), \ldots, k_N(t))$$

with

$$k_n(t) = \begin{cases} 0, & m_n(t) = 0, \\ 0, & m_n(t) = 1 \ \text{ w.p. } \ 1 - p_n \\ 1, & m_n(t) = 1 \ \text{ w.p. } \ p_n \end{cases} \tag{3}$$

denotes the collection of successful LLEs in a time slot, i.e., the switch's connectivity. We use set

$$\mathcal{K}(m(t)) \subseteq \{0,1\}^N \tag{4}$$

to capture all the possible switch connectivities for a given memory allocation $m(t) \in \mathcal{M}$. Note that a memory allocation has a total of $|\mathcal{K}(m)| = 2^M$ possible switch connectivites if all the memories are used.[4] Figure 3 shows how the switch connectivity depends on different memory allocations and the successful LLEs. Also, observe from the figure that different memory allocations can result in the same connectivity due to some LLE attempts failing.

The duration of a switch connectivity depends on how long LLEs last. In this paper, we make the following assumption:

**Assumption 1.** *LLEs last for one time slot.*

This assumption is standard (e.g., [21]), and it allows us to align the duration of a LLE with the frequency in which the switch allocates quantum memories and serves requests.

*3) End-to-end entanglement requests service:* The switch connectivity in a time slot affects the set of available service vectors. Let $k(t) \in \mathcal{K}(m(t))$ with $m(t) \in \mathcal{M}(t)$ be the switch connectivity at time slot $t$. The set of admissible service vectors is given by:[5]

$$\mathcal{B}(m(t), k(t)) = \Bigg\{ b_r \in \{0,1\}, r \in \{1, \ldots, R\} : \text{there exists a matrix } S \in \{0,1\}^{R \times N} \text{ s.t. } s_{rn} = 1 \text{ for all } n \in \Omega(r) \text{ iff } b_r = 1, \text{ and } \sum_{r=1}^{R} s_{rn} \leq k_n(t) \ \forall n \in \{1, \ldots, N\} \Bigg\}.$$

That is, $\mathcal{B}(m(t), k(t))$ contains a collection of binary vectors, where the $r$'th entry of a vector is equal to one if and only if (i) all the clients involved in a request of type $r$ have an active LLE with the switch, and (ii) a LLE is only used to serve one request.

## III. Capacity Region and Throughput Optimal Policy

In this section, we present the main contributions of the paper: the characterization of the capacity region of the quantum switch (Section III-A), and a memory allocation policy that is throughput optimal (Section III-B). In Section III-C, we discuss the scalability of the proposed policy.

### A. Capacity region

Before designing an algorithm, we need to characterize the set of arrival rates that the switch can support. To start, let

$$\lambda := \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} A(t) \tag{5}$$

be the long-term arrival rate of requests at the quantum switch. We say an arrival vector $\lambda$ is admissible (or, it can be supported) if there exists a policy $\pi$ that can generate a sequence of service rate vectors $\{b^\pi(t)\}_{t=1}^{\infty}$ such that

$$\lambda_r \leq f_r^\pi := \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} b_r^\pi(t) \qquad \forall r \in \{1, \ldots, R\}. \tag{6}$$

That is, for a given vector $\lambda$, the switch must be able to generate a long-term service vector $f^\pi$ that is equal to or larger than $\lambda$ component-wise. Hence, by characterizing all the service vectors $f^\pi$ that the switch can generate, we know the arrival vectors that the switch can support (i.e., the switch capacity region).

To define the switch capacity region, we decouple the decision variables from the time slot index $t$ and express them as the fraction of time they can occur. In short, let $\theta_m$ denote the fraction of time a memory allocation $m \in \mathcal{M}$ is used, and $\mathbb{P}(k; m)$ the probability that a switch connectivity $k \in \mathcal{K}(m)$ occurs for a given a memory allocation $m \in \mathcal{M}$. Similarly, let $\delta_b^{k,m}$ be the fraction of time that each service vector $b \in \mathcal{B}(m, k)$ is used for a given switch connectivity and memory allocation. We have the following proposition.

[4] Otherwise, the switch has $2^{\sum_{r=1}^{R} m_r(t)}$ possible connectivities.

[5] Although the set $\mathcal{B}(m(t), k(t))$ depends only on the network connectivity, $k(t)$, we parameterize it with $m(t)$ as well to emphasize that the service vectors are picked after the memory allocation.

**Algorithm 1** (MEW)

1: **Set:** $t = 0$
2: **while** switch is operating **do**
3: $t \leftarrow t + 1$
4: **(S1) Quantum memory allocation:** Select the memory allocation

$$m(t) \in \arg\max_{m \in \mathcal{M}} \sum_{r=1}^{R} Q_r(t)\mu_r(m, Q(t)), \tag{8}$$

where

$$\mu(m, Q(t)) := \sum_{k \in \mathcal{K}(m)} \mathbb{P}(k) w(k, Q(t)) \tag{9}$$

$$w(k, Q(t)) \in \arg\max_{u \in \mathcal{B}(m,k)} \sum_{r=1}^{R} Q_r(t) u_r. \tag{10}$$

5: **(S2) LLEs generation:** The switch attempts to create LLEs with the clients that have a memory connected. The successful LLEs determine the switch connectivity $k(t)$ and the action set $\mathcal{B}(m(t), k(t))$.
6: **(S3) Requests service:** Select a service vector with the update

$$b(t) \in \arg\max_{u \in \mathcal{B}(m(t),k(t))} \sum_{r=1}^{R} Q_r(t) u_r$$

7: **(S4) Queue update:** Serve end-to-end entanglement requests and update the queues with arrivals $A(t)$:

$$Q(t+1) = [Q(t) - b(t)]^+ + A(t) \tag{11}$$

8: **end while**

**Proposition 1** (Quantum switch capacity region)**.** *The capacity region of the quantum switch is:*

$$\Lambda := \Bigg\{ f^\pi : f^\pi = \sum_{m \in \mathcal{M}} \theta_m \sum_{k \in \mathcal{K}(m)} \mathbb{P}(k; m) \sum_{b \in \mathcal{B}(m,k)} \delta_b^{m,k} b, \\ \sum_{m \in \mathcal{M}} \theta_m = 1, \ \sum_{b \in \mathcal{B}(m,k)} \delta_b^{m,k} = 1, \\ \theta_m \geq 0, \ \delta_b^{m,k} \geq 0, \\ \text{for all } b \in \mathcal{B}(m,k), \ k \in \mathcal{K}(m), \ m \in \mathcal{M} \Bigg\}. \tag{7}$$

*Proof sketch:* The full proof is omitted due to space constraints. However, it follows the same methodology as in [23], [27]: writing the fraction of time that the service vectors can be generated—depending on the memory allocations and switch connectivities in our case. ∎

Note that if $\lambda \in \Lambda$ (i.e., the long-term average of requests arrivals is in the capacity region), then there exists a vector $f^\pi$ that satisfies (6). Having $\lambda \in \Lambda$ is usually known as the *necessary* condition for having stable queues [23].

### *B. MEW: A max-weight algorithm for allocating quantum memory and serving requests*

We present *Maximum Expected Weight (MEW)*, an algorithm that stabilizes the queues when the long-term arrival of requests is in the *interior* of the capacity region. MEW (Algorithm 1) consists of three steps. The first step **(S1)** allocates the quantum memories to clients using (8), which consists of maximizing the sum of the expected service in each queue (i.e., $\mu_r$) multiplied by the queue occupancies (i.e., $Q_r$). This update can be regarded as an "expected" max-weight maximization, where the updates in (9) and (10) are intermediate steps to compute the expected rate vectors $\mu(m, Q(t)) = (\mu_1(m, Q(t)), \ldots, \mu_R(m, Q(t)))$ used in (8). The second step **(S2)** generates the LLEs with the clients that have a memory connected. Only some LLE attempts succeed due to interference, which affects the network connectivity and the set of admissible requests service vectors, i.e., set $\mathcal{B}(m(t), k(t))$. The third step **(S3)** consists of finding the service vector $b(t)$ that maximizes the dot product with the vector of queues $Q(t)$. Once all the decision variables have been made, the queues are updated as indicated in (11). We have the following theorem.

**Theorem 1.** *Consider the quantum switch model in Section II, and suppose that the long-term arrival rate of requests $\lambda$ is in the interior of the capacity region $\Lambda$. That is, there exists a vector $\hat{b} \in \Lambda$ such that*

$$\lambda_r + \epsilon \leq \hat{b}_r \qquad \forall r \in \{1, \ldots, R\}$$

*for some $\epsilon > 0$. Then, MEW (Algorithm 1) ensures that*

$$\lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{r=1}^{R} \mathbb{E}[Q_r(t)] \leq \frac{N^2}{\epsilon},$$

*i.e., the queues are strongly stable.*

*Proof:* See Section VII-A. ∎

Strong stability implies that all the requests that arrive are eventually served (i.e., (6) is satisfied), but also that the queues are bounded [28]. The result in Theorem 1 is based on max-weight techniques widely employed in network scheduling problems [27], [23], and the novelty of our contribution resides in the fact that the switch connectivity is random and depends on how we assign quantum memories to links/clients. The latter is different from wireless network models with time-varying connectivity since the allocation of quantum memories affects the switch's connections and, therefore, the set of admissible service vectors. Such coupling is typically not allowed in max-weight or backpressure approaches where the set of available actions can vary over time; however, usually in an i.i.d. manner [29]. In our problem, the action sets $\{\mathcal{B}(m(t), k(t))\}_{t=1}^{\infty}$ are not i.i.d. because they depend on the memory allocation decisions $\{m(t) \in \mathcal{M}\}_{t=1}^{\infty}$. Our approach to tackle this problem is to exploit the linearity of (8), (9), and (10), and evaluate all the possible scheduling decisions for every connectivity. However, enumerating all the cases can be computationally expensive sometimes, as we discuss next.

### *C. MEW scalability*

The step with higher computational cost is the allocation of quantum memories **(S1)**, which involves computing (8), (9), and (10). In brief, the maximization in (8) is over the set of all possible memory allocations, which has cardinality $|\mathcal{M}| = \binom{N}{M}$.[6] Furthermore, we need to compute (9) and

[6]Assuming we allocate all the memories to clients.

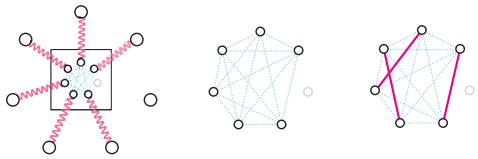


Figure 4. Quantum switch with $N = 7$ clients and $M = 6$ memories. When requests for end-to-end entanglements involve only two clients, the update in (10) reduces to finding a maximum weighted matching in a complete graph.

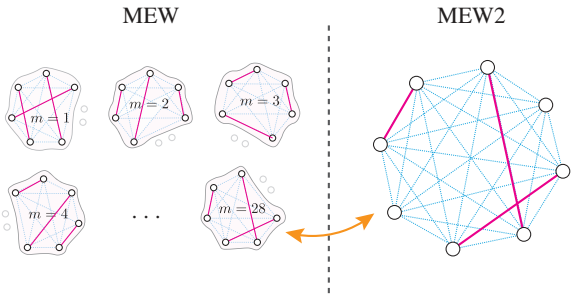



Figure 5. Schematic illustration of how MEW compares to MEW2 for a switch with $N = 8$ clients and $M = 6$ quantum memories. MEW has to find a maximum weighted matching in each of the $\binom{8}{6} = 28$ different complete graphs (with 6 nodes each). MEW2 selects a special type of matching with at most $M/2$ edges in the 8-complete graph.

(10) for every memory allocation $m \in \mathcal{M}$ and network connectivity $k \in \mathcal{K}(m)$ respectively. We could compute (10) only once per switch connectivity since a switch connectivity can be obtained by different quantum memory allocations (see example in Figure 3). However, the number of possible switch connectivities increases exponentially with the number of memories since $|\mathcal{K}(m)| = 2^M$. In addition, the update in (10) requires finding a maximum-weighted matching in a complete hypergraph,[7] which is known to be an NP-hard problem [30].

In sum, MEW does not scale well with $N$ and $M$ since it solves, in the worst case, an exponential number of NP-hard problems for every memory allocation. However, MEW can be used effectively when $N$, $M$ and $R$ are not very large. In the next section, we focus on a special case where we can derive a variant of MEW (MEW2) that has polynomial complexity.

## IV. MEW2: Efficient Scheduling for Bipartite Requests and Successful LLEs

In this section, we study the case where:

- (i) LLE attempts are always successful (i.e., $p_n = 1$, $\forall n \in \{1, \dots, N\}$), and
- (ii) Requests involve only two clients.[8]

This case is important because of two reasons. First, we can extend the duration of a time slot to perform entanglement distillation and increase the probability that every LLE succeed.[9] Second, every multipartite request between clients can be divided into (multiple) bipartite ones. That is because two-qubit gates are universal, i.e., every quantum program can be implemented with two-qubit gates [22].

This case allows us to derive a variant of MEW (MEW2) that has lower computational cost. Specifically, we can allocate memories and select which requests to serve by finding a special type of matching with at most $M/2$ edges in an $N$-complete graph (see Algorithm 2).[10]

### A. Motivation: Complexity of MEW

When end-to-end entanglement requests involve only two clients, (10) corresponds to finding a maximum weighted matching in the complete graph of the clients with an active LLE (see Figure 4). Finding such matching has polynomial time complexity (see, for example, the survey in [31]). The assumption that LLEs are always successful is useful to reduce the number of times we need to call (10). In particular, we have that a memory allocation is associated with a *single* switch connectivity. Hence, $|\mathcal{K}(m(t))| = 1$ for all $m(t) \in \mathcal{M}$ and

$$\mu(m(t), Q(t)) = w(k(t), Q(t))$$

since $m(t) = k(t)$. In sum, we can solve (10) in polynomial time and only once for every admissible memory allocation. Yet, that can still be too much in some cases. For example, if $N = 16$ and $M = 8$, we need to find a maximum weighted matching of $\binom{16}{8} = 12,870$ different graphs to make a single memory allocation decision.

### B. Maximum Expected Weight 2 (MEW2)

We propose *Maximum Expected Weight 2 (MEW2)*, a policy that selects a memory allocation by obtaining a special type of matching in the $N$-complete graph (Algorithm 2). The intuition behind MEW2 is shown in Figure 5 for a switch with $N = 8$ clients and $M = 6$ memories. Recall that **(S1)** in MEW computes a maximum weighted matching in $\binom{N}{M}$ different $M$-complete graphs, and picks one that has maximum weight. Observe from the figure that such matching is also a (non-maximal) matching in the $N$-complete graph. Thus, we can replace step **(S1)** in MEW by *directly* computing a matching with maximum weight among the matchings that have at most $M/2$ edges. We have the following corollary of Theorem 1.

**Corollary 1** (Theorem 1)**.** *Consider the setup of Theorem 1 where the LLE attempts are always successful (i.e., $p_n = 1, \forall n \in \{1, \dots, N\}$). Also, suppose that requests involve connecting two clients and that $M$ is even. Then, MEW2 ensures that the queues are strongly stable.*

*Proof:* See Appendix (Section VII-B). ■

[7] A hypergraph is a generalization of a graph in which an edge can connect any number of vertices.

[8] Without loss of generality we assume that $M$ is even. With bipartite requests, having an odd number of memories means that there will be an unused memory.

[9] Note that there is a trade-off between reducing the duration of a time slot (thus increasing the number of service requests per unit of time), and increase the probability that LLEs succeed.

[10] An $N$-complete graph is a graph with $N$ nodes, in which each pair of graph vertices is connected with an edge. The edges' weights are the queue backlogs.

**Algorithm 2** (MEW2)

1: **Set:** $t = 0$
2: **while** switch is operating **do**
3: $t \leftarrow t + 1$
4: **(S1b) Quantum memory allocation:** Select a matching with at most $M/2$ edges in the $N$-complete graph with maximum possible weight:

$$l(t) \in \arg\max_{u \in O} \sum_{r=1}^{R} Q_r(t)\ u_r, \tag{12}$$

where $O := \{u \in \mathcal{P}_N : \sum_{r=1}^{R} u_r \leq M/2\}$ and $\mathcal{P}_N$ the set of matchings in the $N$-complete graph.
Assign a memory to every client/node that is connected to an edge in $l(t)$, i.e.,

$$m(t) \in \{m \in \mathcal{M} : l(t) \in \mathcal{P}(\mathcal{C}(m))\}, \tag{13}$$

where $\mathcal{C}(m)$ is the complete graph of the clients $n \in \{1, \ldots, N\}$ with $m_n = 1$.
5: **(S2b) LLE generation:** Generate LLEs with the clients that have a memory connected.
6: **(S3b) Requests service:** Select

$$b(t) = l(t)$$

7: **(S4b) Queue update:**

$$Q(t+1) = [Q(t) - b(t)]^+ + A(t) \tag{14}$$

8: **end while**

Finding the matching described in (12), which characterizes the complexity of MEW2, can be done in polynomial time. In particular, we can find such matching by augmenting the $N$-complete graph and then computing a maximum weighted matching. Specifically, the augmented graph has $n-M$ virtual nodes connected to the others with edges that have *infinite* weight. The solution to (12) corresponds to a maximum weighted matching of the augmented graph, which can be found in polynomial time [32], [31].

## V. Numerical Evaluation

In this section, we perform three different simulations to evaluate the performance of MEW and MEW2. Our goal is to illustrate the algorithms' behavior in different scenarios (e.g., request load, LLEs generation) and to study MEW when the update in (8) is carried out approximately. In particular, when MEW uses only $l$ memory allocations out of all the $\binom{N}{M}$ possibilities (Sections V-B and V-C). We refer to such algorithm as $l$-Approximate MEW.

### *A. Simulation 1: Performance of MEW under different arrival rates*

This simulation evaluates the performance of MEW when the requests arrive with three different intensities: 70%, 99%, and 120% of the total load that the switch can support.[11] For the simulation, we set $N = 6$, $M = 3$, and $R = 8$, where all the requests involve connecting three clients (i.e., the requests are tripartite). The probability of the LLE attempts being successful is fixed to $p_n = 0.9$ for all clients and loads.

[11] An intensity of 100% is at the boundary of the capacity region.

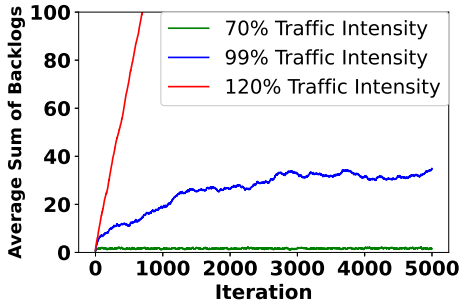



Figure 6. Illustrating the simulation in Section V-A: The evolution of MEW for a quantum switch with 6 clients and 3 memories for different arrival rates. LLE attempts succeed with probability 0.9. Each line is the average of 10 different realizations.

We run MEW for the three different loads and show the evolution of the queue occupancies over time in Figure 6. Observe from the figure that when the arrival rates are in the interior of the capacity region (70% and 99%), the backlogs remain bounded. However, note that the "saturation" points are different, which is in line with the queue stability bound in Theorem 1. Higher intensity (i.e., smaller $\epsilon$ in Theorem 1) implies larger backlogs. Finally, observe from the figure that when the request arrival intensity is equal to 120% (outside of the capacity region), the queues are not stable since their occupancy increases linearly.

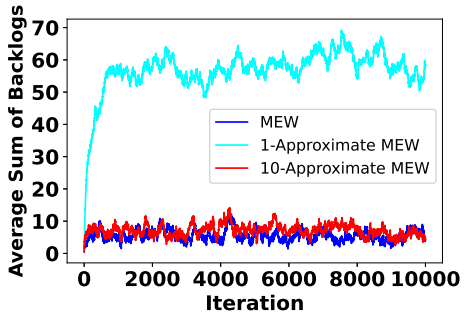
**Conclusions:** MEW stabilizes the queues when the arrivals are in the interior of the capacity region. The average queue occupancies depend on how close the long-term arrival rates are to the boundary of the capacity region (Theorem 1).

### *B. Simulation 2: MEW with memory allocation decision deadlines*

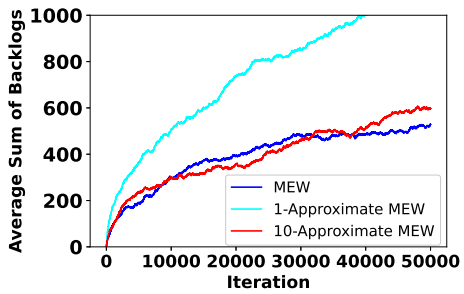
Recall from Section III-C that MEW needs to solve (10) multiple times for every memory allocation. However, we may not be able to evaluate all possible memory allocations since in practice we need to select one within a time deadline. To capture that, we reduce the search space of the problem in (8).

As in Section V-A, we consider $N = 6$ clients, $M = 3$ memories, and probabilities for successful LLE attempts equal to $p_n = 0.9$ for all clients. However, we now consider all types of bipartite and tripartite requests (i.e., $R = \binom{6}{2} + \binom{6}{3} = 35$), hence the time needed to compute (10) increases.

We run MEW and the $\{1, 10\}$-Approximate variant for different arrival rate intensities (70%, 99%, and 120%) and show the results in Figure 7. Observe from the figure that MEW stabilizes the queues when the arrivals are in the interior of the capacity region. However, for the $l$-approximation, the stability depends on the value of $l$ and the traffic intensity. Specifically, the 1-Approximate MEW stabilizes the system when the traffic intensity is 70% (Figure 7a), but not when the intensity is equal to 99% (Figure 7b). In contrast, the 10-Approximation keeps the queues bounded similar to MEW.




(a) 70% Traffic intensity




(b) 99% Traffic intensity

Figure 7. Illustrating the simulation in Section V-B: The evolution of MEW and $\{1, 10\}$-Approximate MEW for a quantum switch with $N = 6$, $M = 3$ and $p_n = 0.9$ for every client $n$. Each line is the average of 10 different realizations.

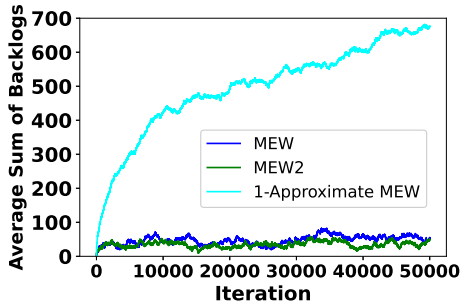



Figure 8. Illustrating the simulation in Section V-C: The evolution of MEW, MEW2 and 1-Approximate MEW for a quantum switch with $N = 7$, $M = 4$ and bipartite requests. The traffic intensity is 99% and the LLE attempts succeed with probability 0.9. Each line is the average of 10 different realizations.

**Conclusions:** The $l$-Approximate MEW can stabilize the queues when the value of $l$ is large enough. How large $l$ should be is related to how close the arrival rates are to the boundary of the capacity region. As future work, it is interesting to investigate how the capacity region scales when the memory allocation is obtained approximately (e.g., as a function of the parameter $l$).

### C. Simulation 3: Performance of MEW2

In this simulation, we compare MEW2 to MEW and 1-Approximate MEW in a switch with $N = 7$ clients and $M = 4$ memories. The traffic intensity is fixed to 99%, and we assume that the LLE attempts are always successful. Also, all the requests for end-to-end entanglements are bipartite with $R = \binom{7}{2} = 21$.

We run the three algorithms and show the results in Figure 8. Observe from the figure that MEW2 can keep the queues stable and that its behavior is similar to MEW. Nonetheless, recall that MEW has as higher computational cost than MEW2 (see discussion in Section IV). Next, observe from Figure 8 that the 1-Approximate MEW (which has a comparable cost to MEW2) cannot stabilize the queues.

**Conclusions:** The behavior of MEW2 is similar to MEW even though its complexity is significantly lower. The 1-Approximate MEW does not keep the queues stable despite having a similar computational cost to MEW2.

## VI. Conclusions

In this paper, we have studied the problem of operating a quantum switch with memory constraints. The switch has to allocate quantum memories to clients to generate link-level entanglements (LLEs), and then use these to serve end-to-end entanglements requests. The paper's main contribution is twofold: (i) to characterize the switch's capacity region, and (ii) to propose a policy (MEW) that is throughput optimal. We also present MEW2, a polynomial time variant of MEW tailored to the case where end-to-end entanglements are bipartite and LLE attempts are always successful.

## VII. Appendix

### A. Proof of Theorem 1

We prove that the queues are stable by using a quadratic Lyapunov function, and ultimately showing that the proposed policy has expected negative drift. That is, the queues at time slots $t$ and $t+1$ satisfy: $\mathbb{E}[\|Q(t+1)\|^2 - \|Q(t)\|^2] < 0$, where the expectations are w.r.t. the (i) request arrivals; (ii) successful LLEs; and (iii) all possible queue values at time $t$. To start, observe that

$$
\begin{aligned}
&\|Q(t+1)\|^2 = \\
&= \|[Q(t) - b(t)]^+\|^2 + \|A(t)\|^2 + 2\sum_{r=1}^{R}[Q_r(t) - b_r(t)]^+ A_r(t) \\
&\leq \|Q(t) - b(t)\|^2 + \|A(t)\|^2 + 2\sum_{r=1}^{R}[Q_r(t) - b_r(t)]^+ A_r(t) \\
&= \|Q(t)\|^2 + \|b(t)\|^2 + \|A(t)\|^2 - 2\sum_{r=1}^{R} Q_r(t) b_r(t) \\
&\qquad + 2\sum_{r=1}^{R}[Q_r(t) - b_r(t)]^+ A_r(t) \\
&\leq \|Q(t)\|^2 + \|b(t)\|^2 + \|A(t)\|^2 - 2\sum_{r=1}^{R} Q_r(t) b_r(t) \\
&\qquad + 2\sum_{r=1}^{R} Q_r(t) A_r(t) \\
&= \|Q(t)\|^2 + \|A(t)\|^2 + \|b(t)\|^2 + 2\sum_{r=1}^{R} Q_r(t)(A_r(t) - b_r(t)).
\end{aligned}
$$

Next, since $\|A(t)\|^2 \leq N^2$, $\|b(t)\|^2 \leq N^2$ (by assumption), and $\mathbb{E}[A(t)] = \lambda$ by assumption, we can take expectations with respect to $A_r(t)$ for a fixed queue $Q(t)$ to obtain:

$$\begin{aligned}\mathbb{E}[\|Q(t+1)\|^2 - \|Q(t)\|^2 \mid Q(t)] \\ \leq 2N^2 + 2\sum_{r=1}^{R} Q_r(t)(\lambda_r - b_r(t))\end{aligned} \tag{15}$$

We proceed to upper bound the expected value of $-\sum_{r=1}^{R} Q_r(t)b_r(t)$. To start, because $b(t)$ is a random vector that depends on the switch connectivity and memory allocation at time $t$, we have

$$\mathbb{E}\left[-\sum_{r=1}^{R} Q_r(t)b_r(t)\right] = -\sum_{r=1}^{R} Q_r(t)\mu_r(m(t), Q(t))$$

where $\mu_r(m(t), Q(t))$ is defined in (9). Note that $Q(t)$ does not depend on the switch connectivity in time slot $t$. Combining the last equation with (15), we have

$$\begin{aligned}\mathbb{E}[\|Q(t+1)\|^2 - \|Q(t)\|^2 \mid Q(t)] \\ \leq 2N^2 + 2\sum_{r=1}^{R} Q_r(t)(\lambda_r - \mu_r(m(t), Q(t))).\end{aligned} \tag{16}$$

where the expectation is with respect to the switch connectivities for a fixed memory allocation.

Next, observe that the memory allocation in (8) ensures that:

$$\begin{aligned}-\sum_{r=1}^{R} Q_r(t)\mu_r(m(t), Q(t)) \\ \leq -\sum_{r=1}^{R} Q_r(t)\mu_r(m, Q(t)) \qquad \forall m \in \mathcal{M}.\end{aligned} \tag{17}$$

since $\mu(m(t), Q(t))$ maximizes $\sum_{r=1}^{R} Q_r(t)\mu_r(m(t), Q(t))$. Now, let $\theta_m \geq 0$ with $\sum_{m\in\mathcal{M}} \theta_m = 1$ and observe that

$$\begin{aligned}-\sum_{r=1}^{R} Q_r(t)\mu_r(m(t), Q(t)) \\ = -\sum_{m\in\mathcal{M}} \theta_m \sum_{r=1}^{R} Q_r(t)\mu_r(m(t), Q(t)) \\ \overset{(a)}{\leq} -\sum_{m\in\mathcal{M}} \theta_m \sum_{r=1}^{R} Q_r(t)\mu_r(m, Q(t)), \\ \overset{(b)}{=} -\sum_{m\in\mathcal{M}} \theta_m \sum_{r=1}^{R} Q_r(t) \sum_{k\in\mathcal{K}(m)} \mathbb{P}(k;m) w_r(k, Q(t)) \\ = -\sum_{m\in\mathcal{M}} \theta_m \sum_{k\in\mathcal{K}(m)} \mathbb{P}(k;m) \sum_{r=1}^{R} Q_r(t) w_r(k, Q(t))\end{aligned}$$

where (a) follows by (17) and (b) by (9).

Now, recall $w(k, Q(t))$ maximizes $\sum_{r=1}^{R} Q_r(t)w_r(k, Q(t))$ because how we defined it in (10), and let

$$\delta_b^{m,k} \geq 0 \quad \text{for all } b \in \mathcal{B}(m,k), \qquad \sum_{b\in\mathcal{B}(m,k)} \delta_b^{m,k} = 1$$

for every $k \in \mathcal{K}(m)$ and $m \in \mathcal{M}$. Using the same strategy as before, we have

$$\begin{aligned}-\sum_{r=1}^{R} Q_r(t)w_r(k, Q(t)) \qquad k \in \mathcal{K}(m) \\ = -\sum_{b\in\mathcal{B}(m,k)} \delta_b^{m,k} \sum_{r=1}^{R} Q_r(t) w_r(k, Q(t)) \\ \overset{(a)}{\leq} -\sum_{b\in\mathcal{B}(m,k)} \delta_b^{m,k} \sum_{r=1}^{R} Q_r(t) b_r \\ = -\sum_{r=1}^{R} Q_r(t) \sum_{b\in\mathcal{B}(m,k)} \delta_b^{m,k} b_r\end{aligned}$$

where $b$ in inequality (a) holds for any vector in $\mathcal{B}(m,k)$. Combining the previous equations, we obtain

$$-\sum_{r=1}^{R} Q_r(t)\mu_r(m(t), Q(t)) \leq -\sum_{r=1}^{R} Q_r(t)\hat{b}_r$$

where

$$\hat{b} = \sum_{m\in\mathcal{M}} \theta_m \sum_{k\in\mathcal{K}(m)} \mathbb{P}(k;m) \sum_{b\in\mathcal{B}(m,k)} \delta_b^{m,k} b$$

is any vector in $\Lambda$ (Proposition 1). Hence, from (16), we have

$$\begin{aligned}\mathbb{E}[\|Q(t+1)\|^2 - \|Q(t)\|^2 \mid Q(t)] \\ \leq 2N^2 + 2\sum_{r=1}^{R} Q_r(t)(\lambda_r - \hat{b}_r(t))\end{aligned}$$

The rest of the proof follows the usual max-weight arguments. Because $\lambda_r + \epsilon \leq \hat{b}_r$ for some $\epsilon > 0$ by assumption, it holds

$$\begin{aligned}\mathbb{E}[\|Q(t+1)\|^2 - \|Q(t)\|^2 \mid Q(t)] \\ \leq 2N^2 - 2\sum_{r=1}^{R} Q_r(t)\epsilon\end{aligned}$$

Now, take expectations with respect to all the possible values of $Q(t)$ to obtain

$$\mathbb{E}[\|Q(t+1)\|^2 - \|Q(t)\|^2] \leq 2N^2 - 2\sum_{r=1}^{R} Q_r(t)\epsilon, \tag{18}$$

and observe that $\mathbb{E}[\|Q(t+1)\|^2 - \|Q(t)\|^2] < 0$ when $\sum_{r=1}^{R} Q_r(t) > \frac{2N^2}{2\epsilon}$. That is, the queue drift is negative. Finally, sum (18) from $t = 1, \ldots, T$ to obtain

$$\begin{aligned}\mathbb{E}[\|Q(T+1)\|^2] - \mathbb{E}[\|Q(0)\|^2] \\ \leq 2TN^2 - 2\epsilon\sum_{t=1}^{T}\sum_{r=1}^{R} \mathbb{E}[Q_r(t)]\end{aligned}$$

Rearranging terms and dividing by $T$ yields

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{r=1}^{R} \mathbb{E}[Q_r(t)] \leq \frac{N^2}{\epsilon} + \frac{\|Q(0)\|^2}{2T\epsilon},$$

and taking $T \to \infty$ we obtain the stated result.

### B. Proof of Corollary 1

Let $\mathcal{C}(m)$ be the complete graph that results from the clients $n \in \{1, \ldots, N\}$ for which $m_n = 1$. Moreover, let $\mathcal{P}(G)$ be the set of matchings of a graph $G$, and $\mathcal{P}_N$ be the set of matchings of the complete graph with $N$ clients. The proof of Corollary 1 relies on finding a policy that is equivalent to MEW, which we proved in Theorem 1 that stabilizes the queues.

We can rewrite step **(S1)** of MEW as follows:

$$m(t) \in \arg\max_{m \in \mathcal{M}} \max_{u \in \mathcal{P}(\mathcal{C}(m))} \sum_{r=1}^{R} Q_r(t)\ u_r. \tag{19}$$

Problem (19) summarizes MEW when LLE attempts are always successful and the requests are bipartite. That holds true because (i) its solution picks the memory allocation of **(S1)**, and (ii) the maximizer of the inner optimization problem is the final service vector of step **(S3)**. Let $M^*(t)$ be the set of solutions to problem (19) and define the sets

$$\Pi := \bigcup_{m \in \mathcal{M}} \mathcal{P}(\mathcal{C}(m)), \quad O := \{u \in \mathcal{P}_N : \sum_{r=1}^{R} u_r \leq M/2\}.$$

Recall that $M$ is even. Then, step **(S3)**, is equivalent to

$$b(t) \in \bigcup_{m^* \in M^*(t)} \arg\max_{u \in \mathcal{P}(\mathcal{C}(m^*))} \sum_{r=1}^{R} Q_r(t) u_r \tag{20}$$

$$= \arg\max_{u \in \Pi} \sum_{r=1}^{R} Q_r(t) u_r$$

$$= \arg\max_{u \in O} \sum_{r=1}^{R} Q_r(t) u_r. \tag{21}$$

Therefore, step **(S3)** of MEW can be solved immediately by solving the problem (21) (see step (**S1b**) in MEW2). Note that (21) does not depend on the optimal set $M^*$ and therefore we can solve it before allocating the quantum memory. However, to characterize the policy we have to find a memory allocation that would make $b(t)$ computed in (21) a feasible service vector. In (19) note that any memory allocation that includes the clients involved in the matching $b(t)$ (step (**S1b**)), belongs in $M^*$ and therefore would be an optimal memory allocation.

## REFERENCES


[1] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[2] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[3] Ibm unveils new roadmap to practical quantum computing era; plans to deliver 4,000+ qubit system. [Online]. Available: https://newsroom.ibm.com/2022-05-10-IBM-Unveils-New-Roadmap-to-Practical-Quantum-Computing-Era-Plans-to-Deliver-4,000-Qubit-System

[4] "IBM unveils breakthrough 127-qubit quantum processor," Available at https://newsroom.ibm.com/2021-11-16-IBM-Unveils-Breakthrough-127-Qubit-Quantum-Processor.

[5] F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[6] R. Van Meter and S. J. Devitt, "The path to scalable distributed quantum computing," *Computer*, vol. 49, no. 9, pp. 31–42, 2016.

[7] S. Guha and C. Gagatsos, "Cluster-state quantum computing methods and systems," Jul. 7 2022, uS Patent App. 17/594,874.

[8] C. Qiao, Y. Zhao, G. Zhao, and H. Xu, "Quantum data networking for distributed quantum computing: Opportunities and challenges," in *INFOCOM*. IEEE, 2022, pp. 1–6.

[9] A. Yimsiriwattana and S. J. Lomonaco Jr, "Generalized ghz states and distributed quantum computing," *arXiv: quant-ph/0402148*, 2004.

[10] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[11] J.-W. Pan, D. Bouwmeester, H. Weinfurter, and A. Zeilinger, "Experimental entanglement swapping: entangling photons that never interacted," *Physical review letters*, vol. 80, no. 18, p. 3891, 1998.

[12] R.-B. Jin, M. Takeoka, U. Takagi, R. Shimizu, and M. Sasaki, "Highly efficient entanglement swapping and teleportation at telecom wavelength," *Scientific reports*, vol. 5, no. 1, pp. 1–7, 2015.

[13] W. Dai and D. Towsley, "Entanglement swapping for repeater chains with finite memory sizes," 2021. [Online]. Available: https://arxiv.org/abs/2111.10994

[14] M. G. de Andrade, W. Dai, S. Guha, and D. Towsley, "Optimal policies for distributed quantum computing with quantum walk control plane protocol," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2021, pp. 452–453.

[15] C. Cicconetti, M. Conti, and A. Passarella, "Request scheduling in quantum networks," *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 2–17, 2021.

[16] L. Le and T. N. Nguyen, "Dqra: Deep quantum routing agent for entanglement routing in quantum networks," *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–12, 2022.

[17] N. K. Panigrahy, P. Dhara, D. Towsley, S. Guha, and L. Tassiulas, "Optimal entanglement distribution using satellite based quantum networks," *arXiv preprint arXiv:2205.12354*, 2022.

[18] G. Vardoyan, S. Guha, P. Nain, and D. Towsley, "On the exact analysis of an idealized quantum switch," *SIGMETRICS Perform. Eval. Rev.*, vol. 48, no. 3, pp. 79–80, mar 2021.

[19] P. Nain, G. Vardoyan, S. Guha, and D. Towsley, "Analysis of a tripartite entanglement distribution switch," *Queueing Systems*, 2022.

[20] W. Dai, A. Rinaldi, and D. Towsley, "Entanglement swapping in quantum switches: Protocol design and stability analysis," 2021. [Online]. Available: https://arxiv.org/abs/2110.04116

[21] T. Vasantam and D. Towsley, "A throughput optimal scheduling policy for a quantum switch," in *Quantum Computing, Communication, and Simulation II*, P. R. Hemmer and A. L. Migdall, Eds. SPIE, mar 2022.

[22] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[23] L. Georgiadis, M. J. Neely, and L. Tassiulas, *Resource Allocation and Cross-Layer Control in Wireless Networks*. Foundations and Trends in Networking, 2006, vol. 1, no. 1.

[24] J.-W. Pan, C. Simon, Č. Brukner, and A. Zeilinger, "Entanglement purification for quantum communication," *Nature*, vol. 410, no. 6832, pp. 1067–1070, 2001.

[25] Y. Yu, F. Ma, X.-Y. Luo, B. Jing, P.-F. Sun, R.-Z. Fang, C.-W. Yang, H. Liu, M.-Y. Zheng, X.-P. Xie *et al.*, "Entanglement of two quantum memories via fibres over dozens of kilometres," *Nature*, vol. 578, no. 7794, pp. 240–245, 2020.

[26] G. Bacciagaluppi, "The Role of Decoherence in Quantum Mechanics," in *The Stanford Encyclopedia of Philosophy*, Fall 2020 ed., E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2020.

[27] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, 1992.

[28] M. J. Neely, "Stability and capacity regions or discrete time queueing networks," *arXiv preprint arXiv:1003.3396*, 2010.

[29] L. Tassiulas, "Scheduling and performance limits of networks with constantly changing topology," *IEEE Transactions on Information Theory*, vol. 43, no. 3, pp. 1067–1073, 1997.

[30] J. Han and A. Treglown, "The complexity of perfect matchings and packings in dense hypergraphs," 2016. [Online]. Available: https://arxiv.org/abs/1609.06147

[31] A. Schrijver *et al.*, *Combinatorial optimization: polyhedra and efficiency*. Springer, 2003, vol. 24.

[32] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of mathematics*, vol. 17, pp. 449–467, 1965.