

ARTICLE OPEN



Effective routing design for remote entanglement generation on quantum networks

Changhao Li^{1,2,4}, Tianyi Li^{3,4}✉, Yi-Xiang Liu^{1,2} and Paola Cappellaro^{1,2}✉

Quantum network is a promising platform for many ground-breaking applications that lie beyond the capability of its classical counterparts. Efficient entanglement generation on quantum networks with relatively limited resources such as quantum memories is essential to fully realize the network's capabilities, the solution to which calls for delicate network design and is currently at the primitive stage. In this study we propose an effective routing scheme to enable automatic responses for multiple requests of entanglement generation between source-terminal stations on a quantum lattice network with finite edge capacities. Multiple connection paths are exploited for each connection request while entanglement fidelity is ensured for each path by performing entanglement purification. The routing scheme is highly modularized with a flexible nature, embedding quantum operations within the algorithmic workflow, whose performance is evaluated from multiple perspectives. In particular, three algorithms are proposed and compared for the scheduling of capacity allocation on the edges of quantum network. Embodying the ideas of proportional share and progressive filling that have been well-studied in classical routing problems, we design another scheduling algorithm, the propagatory update method, which in certain aspects overrides the two algorithms based on classical heuristics in scheduling performances. The general solution scheme paves the road for effective design of efficient routing and flow control protocols on applicational quantum networks.

npj Quantum Information (2021)7:10; <https://doi.org/10.1038/s41534-020-00344-4>

INTRODUCTION

Quantum networks¹ enable many applications that lie beyond the scope of classical data networks, such as quantum communication², clock synchronization³, and distributed quantum computing^{4–6}. Most of these applications require the generation of entangled pairs between far-away stations on the quantum network. Recent experiments⁷ have demonstrated deterministic entanglement between two spatially separated solid-state memories via optical photons. The distance in this physical (hardware) layer⁸ can be further increased with low-loss optical links, such as transporting photons at the tele-communication wavelength^{9–11}.

Still, the time required for remote entanglement generation increases exponentially with distance due to potential loss. In analogy with the functioning on the link layer of classical data networks, intermediate nodes on the quantum nodes could serve as quantum repeaters^{12–16}, by which means the exponential time cost in entanglement generation could be reduced to polynomial time scaling. Quantum repeaters (Fig. 1a) link quantum stations over longer distances by performing entanglement swapping, e.g., joint Bell state measurements at the local repeater station (see Supplemental Materials) aided by classical communication (LOCC).

Nevertheless, quantum repeaters sometimes might not generate entangled pairs with a certain desired fidelity, which is required in many applications using entangled pairs. For example, in quantum cryptography protocols (e.g., the BB84 protocol¹⁷), an entanglement fidelity beyond the quantum bit error rate (QBER) is required to ensure the security of key distribution. Entanglement purification is a technique that can increase the fidelity of entangled pairs, at the expense of a reduced number of entangled pairs, i.e., the purification process reduces the number of shared

entangled pairs along the links between adjacent nodes on the network (Fig. 1b).

On top of the link layer mentioned above, where entanglement swapping and purifications are carried out, a network layer could be implemented for robust and well-functioning network design to enable the broader capabilities of quantum networks^{18–21}. In particular, the network design determines routing principles that enable effective network responses to simultaneous requests of entanglement generation between remote stations (e.g., $S_{1(2)} - T_{1(2)}$ in Fig. 1c). Given limited quantum capacity, i.e., a small number of available quantum memories possessed by each node, it is then critical to design efficient routing protocols for simultaneous entanglement generation between multiple stations.

To tackle with this routing problem, we assume a central classical processor (scheduler) in the network which performs routing calculations and broadcasts the schedule to all network stations. Local classical processors to design the routing protocol can result in a degraded network performance²¹. At the beginning of each processing time window, upon receiving a batch of connection requests, the processor determines the paths (virtual circuits) to be initiated within the network. It then carries out the scheduling of edge capacity allocation, and determines the flow of each path according to scheduling results. Such routing information is broadcast to nodes, and local physical operations, including entanglement purification and swapping, are conducted to realize entangled connections between remote nodes. The generated entangled pairs connecting remote stations then serve as the building blocks for various quantum network applications. Entangled pairs between adjacent stations consumed in this time window are generated again to prepare for the next batch of connection requests in the consecutive time window.

¹Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. ²Department of Nuclear Science and Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. ³System Dynamics Group, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139, USA.

⁴These authors contributed equally: Changhao Li, Tianyi Li. ✉email: tianyi@mit.edu; pcappell@mit.edu

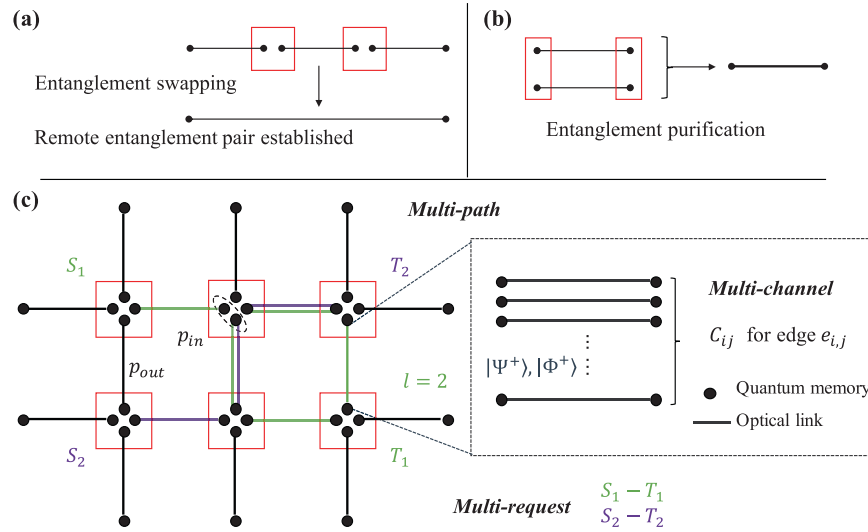


Fig. 1 Routing problem statement. **a** Diagram of entanglement swapping. Two remote stations can be entangled with the assistance of intermediate nodes. Black dots (lines) represent quantum memories (optical links). **b** Diagram of entanglement purification. With local physical operations and communication through classical channels, an entangled pair with high fidelity can be distilled from two fresh pairs with lower fidelity. **c** Entanglement routing on a 2D square lattice network. Edge (i, j) consists of multiple entangled pairs (multi-channels) such as Bell states, whose capacity is denoted as C_{ij} . Entanglement generation between $S_{1(2)}$ and $T_{1(2)}$ is requested and multiple paths are identified for each request. The probabilities of successfully building an entangled pair between adjacent nodes and performing local Bell state measurement are denoted as p_{out} and p_{in} , respectively.

Recent work by Pant et al.²¹ introduced a greedy algorithm for this routing problem on quantum lattice networks. While this algorithm works well on networks having one entangled pair shared between each pair of adjacent nodes and one connection request being processed per time window, a general routing protocol where neighboring nodes share more entangled pairs (multi-channel), and multiple user requests are simultaneously attended to (multi-request), remains an open design question. Novel architectures are to be proposed and investigated when this layer of complexity is added to the problem. Moreover, for a robust routing algorithm, it is expected that multiple paths, instead of only the shortest path, are to be utilized for each connection request, as a prevention against potential network failures.

In this study, we propose a general routing scheme which provides efficient solutions to the above entanglement generation problem, with advanced design features (multi-request, multi-path and multi-channel). We devise algorithms to effectively allocate limited quantum resources and meet the needs of entanglement generation for arbitrary connection requests, upon satisfying the fidelity threshold for generated entanglements.

Routing protocols on classical data networks have been extensively studied and are being continuously developed. Yet for quantum networks, routing schemes well-established on today's communication networks (e.g., the Internet) are not directly applicable. The quantum no-cloning theorem forbids the duplication of entangled pairs and renders unfeasible the idea of packet switching, a major routing mechanism for large-scale data networks in which the same information package is sent multiple times along indefinite routing paths. Instead, for quantum networks, one may have to go back to early routing mechanisms that appeared when computers were expensive (note these mechanisms are still in use today for specific applications), for example, the virtual circuit switching scheme²², where one establishes temporal paths between two stations and maintains such paths during a time window, before starting over again for the next round of connection. Our routing design is based on this scheme. Conceptually, the key difference between packet switching and virtual circuit switching is the manner in which information redundancy is guaranteed, a key issue in practical

routing designs in the presence of potential network failures, which are frequent and unpredictable in real applications. The no-cloning theorem decides that the redundancy of quantum information cannot be obtained through duplication of entangled pairs, while instead could be preserved in the generation of sufficient entangled pairs via virtual circuits.

The organization of the paper is as follows: we first formalize the problem statement of this study, based on which the stepwise routing scheme is introduced. Three algorithms are proposed for the scheduling of capacity allocation on network edges, in which adjacent entangled pairs are properly distributed upon connection requests. The routing performance is evaluated from multiple perspectives, and a sample routing result is demonstrated and explained in detail. We then conduct extensive simulations to discuss system parameters, compare the three scheduling algorithms, and investigate the robustness of the scheme under different conditions. Limitations, scope of usage as well as potential extensions of the current study are discussed in the end, where the paper is concluded.

RESULTS

Problem statement

Consider a lattice quantum network $G(V, E)$ of V nodes and E edges. Each node represents a quantum station with a finite number of qubit memories (here we assume first-generation quantum repeaters¹⁶). The edge capacity C_0 is defined as the maximal number of entangled pairs that can be generated between adjacent nodes. Without loss of generality, we consider maximally entangled states (Bell states) between adjacent nodes, $|\Phi^\pm\rangle = \frac{1}{\sqrt{2}}(|0\rangle|0\rangle \pm |1\rangle|1\rangle)$ and $|\Psi^\pm\rangle = \frac{1}{\sqrt{2}}(|0\rangle|1\rangle \pm |1\rangle|0\rangle)$. Since a simple local operation such as a single qubit X or Z gate can transform Bell states from one to another, this provides us the freedom of switching the states at low cost. Different Bell states or states with different entanglement entropy might serve as identities (ID) for entangled pairs for further usages in the network layer⁸. In reality, the bipartite states between adjacent nodes are non-maximally entangled states, characterized by their entanglement fidelity. To ensure this fidelity is above a certain

threshold F_{th} , the network is initialized by quantum entanglement purification on each edge, which results in a reduced number of shared entanglement pairs, thus a reduced edge capacity. We assume the fidelities of multiple entangled pairs along the same edge to be identical while the fidelities on different edges can vary, as influenced by various factors such as geographical constraints or human interventions. The average entanglement fidelity of a certain edge can be calculated by pre-characterization; here we assume the fidelities on all edges to follow a normal distribution $N(F_{mean}, F_{std})$. Note we take the fidelity to be one if it exceeds unity.

At the start of a time window, a set R of simultaneous connection requests are received. Each request $r \in R$ requires the generation of at least $\bar{F}$ entangled pairs between two remote nodes (denoted source node and terminal node) by implementing entanglement swapping (see Fig. 1a) along a given path. Local entanglement swapping operations at quantum stations are imperfect, with a probability of success $p_{in} \in (0, 1)$. This implies that the time needed to successfully entangle two remote stations along a certain path will grow exponentially with increasing path length, an important aspect in evaluating routing performances (see following sections). An optical link between two nodes will be successfully established with probability p_{out} , mainly limited by optical loss during the long-distance transmission. We assume homogeneous p_{in} and p_{out} on all network nodes and edges in simulations. We assumed first-generation quantum repeaters, that is, heralded entanglement generation (purification) is applied on

the repeaters and quantum error correction (QEC) procedures are not available to suppress the errors. Still, our model could easily be generalized to second- or third-generation repeaters, where QEC is used to replace the heralded entanglement generation (purification). This would greatly enhance the network performance, as described by the parameters p_{in}, p_{out} . In addition, the QEC protocol only needs one-way signaling and would allow higher communication rates between quantum stations.

With these preliminaries, we define the routing problem on small-capacity regular (lattice) quantum networks as the following:

Given a quantum network with non-uniform edge capacity and variable topology due to imperfect initialization and limited quantum memory on stations, design an effective algorithm that provides routing solutions to arbitrary entangled-pair requests, able to ensure above-threshold entanglement fidelity and to process multiple connection requests during the same time window.

To efficiently utilize network resources within each processing window, multiple connection paths L' are identified and established for each request r , as opposed to only using the single shortest path between the request pair. Specifically, the solution scheme determines the flow (allocated capacity) $f_{ij}^{r,l}$ of each path l for each connection request r , on every edge (i, j) in the network, such that the aggregated flow of each request $f^r = \sum_{l \in L'} f_{ij}^{r,l}$ is able to meet the demand $\bar{F}$ to a sufficient extent.

This problem formulation considers multiple paths on the network for a specific connection request, as opposed to only

Routing Protocol

- 1: (Global $F_{mean}, F_{std}, F_{th}, p_{in}, p_{out}$).
- 2: **New processing window: open**
- 3: **Step 0 Initialization**
- 4: Resume lattice network $G = (V, E)$. Initialize edge capacity C_0 .
- 5: **Perform entanglement generation between adjacent nodes.**
- 6: Receive requests of entanglement generation (connection) $R = \{r\} = \{[S_r, T_r], \bar{F}^r, w_r\}$.
- 7: **Step 1 Entanglement Purification**
- 8: **Conduct entanglement purification on edges with fidelity below required threshold F_{th} .**
- 9: Realize capacity $C_{ij} \leq C_0$ on each edge (i, j) .
- 10: Determine system parameters $X = \{l_{max}, k, \alpha, \beta\}$ and f_{min} .
- 11: Disactivate edges with insufficient capacity $C_{ij} < l_{max}$. $E \rightarrow E'$. $G \rightarrow G'$.
- 12: **Step 2 Path Determination**
- 13: **for each $r \in R$ do**
- 14: **if** No path exists between S_r and T_r **then**
- 15: **Continue**
- 16: Identify k shortest paths $L^r = \{l\}_k$ between S_r and T_r .
- 17: **for each $l \in L^r$ do**
- 18: Append tuple $h = [r, l, d, o]$ to path information set H on each edge in l .
- 19: **Step 3 Capacity Allocation Scheduling** (Algorithm 1-3 in Supplemental Materials)
- 20: Algorithm 1: **Proportional Share** (PS)
- 21: Algorithm 2: **Progressive Filling** (PF)
- 22: Algorithm 3: **Propagatory Update** (PU)
- 23: **Step 4 Flow Determination and Performance Evaluation**
- 24: **for each $r \in R$ do**
- 25: **for each $l \in L^r$ do**
- 26: Determine $f^{r,l} = \min C_{ij}^{r,l}$ for $(i, j) \in l$ on path l of request r .
- 27: Compute $F, u_{ij}, \gamma_r, J_{req}, J_{path}$. Evaluate routing performance.
- 28: Output scheduling result $\{R, G'\} \Rightarrow \{C_{ij}^{r,l}\}$.
- 29: **Step 5 Remote Entanglement Establishment**
- 30: **Perform entanglement swapping according to scheduling results.**
- 31: Compare realized flow $f^r = \sum_{l \in L^r} f^{r,l}$ with demand $\bar{F}^r$. Queue unsatisfied demand to next window.

Fig. 2 Illustration of the routing protocol. Physical steps (in which quantum operations are physically conducted) are shown in red.

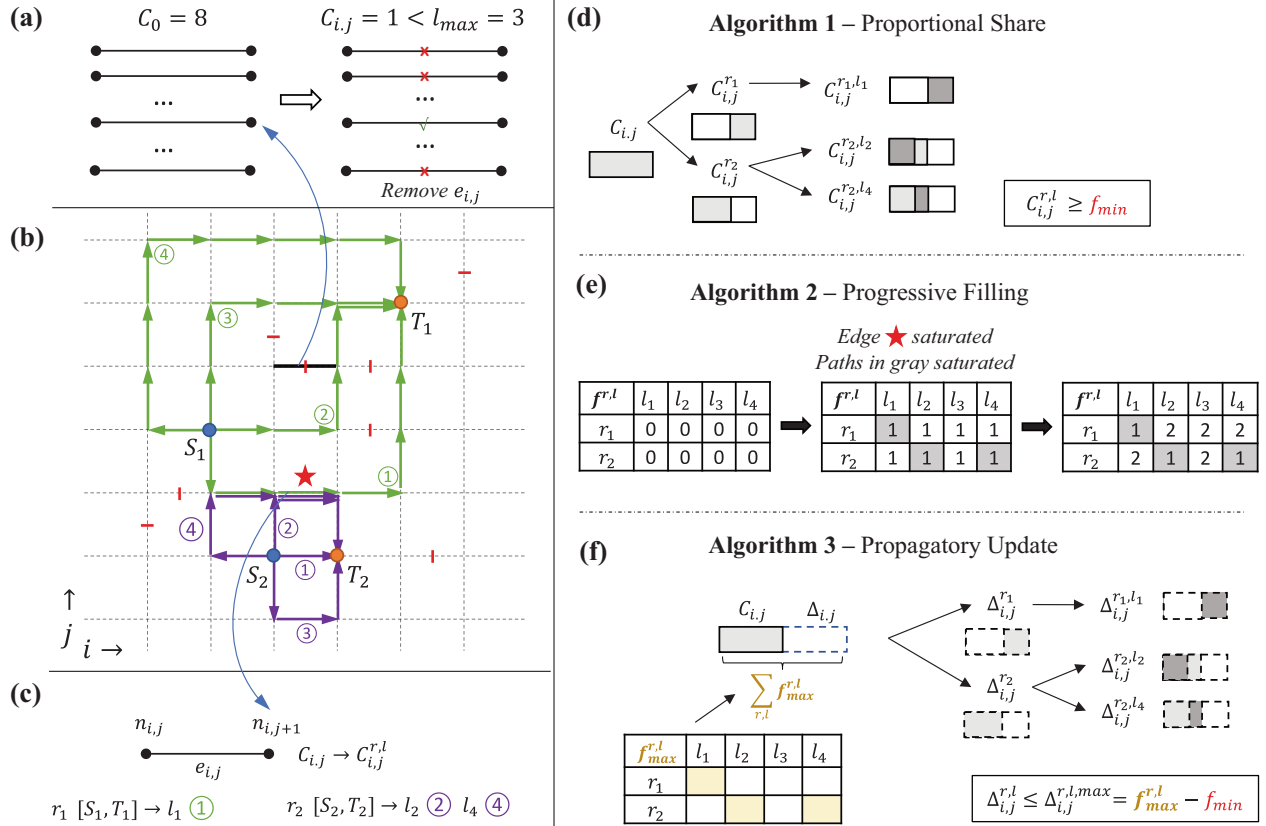


Fig. 3 Illustration of the routing protocol. **a** Initialization of network topology. On a specific edge, purification realizes one entanglement pair out of eight available entangled pairs ($C_{ij} = 1$, $C_0 = 8$), which falls below the maximum number of paths $l_{\max}$ allowed on each edge. To ensure that each path has at least one channel allocated, this edge with insufficient capacity is then deactivated (indicated by the red bar). **b** A sample routing scenario. Two connection requests ($r_1: S_1 \leftrightarrow T_1$; $r_2: S_2 \leftrightarrow T_2$) are received; four-shortest paths are identified for each request, utilizing active edges (edges without red bars) on the square lattice network. Each edge may be utilized by multiple paths of a certain connection request, and by multiple requests. Different paths are shown with arrows for clarity, but there is no directionality along the paths. **c** Capacity allocation scheduling. Three paths for two requests (l_1 for r_1 ; l_2, l_4 for r_2) utilize the starred edge (in **b**), on which the three scheduling algorithms (**d–f**) are illustrated. For proportional share and propagatory update, entangled pairs on this edge are distributed among the three paths through a two-stage allocation: the quota is first determined for each request, then for each path of different requests. In proportional share, the edge capacity is directly allocated; propagatory update calculates the summarized desired capacity on this edge from all paths that utilize it, using the information in the global schedule table, then allocates the unmet capacity demand to these paths. Corresponding deductions are made from the desired flows whose sum exceeds the real edge capacity, and the schedule table is updated at each step. The minimum flow for each path is $f_{\min}^{r,l}$; the maximum flow deduction for path l of request r is $f_{\max}^{r,l} - f_{\min}^{r,l}$, where $f_{\max}^{r,l}$ is the desired capacity in the global routing table. For progressive filling, $f^{r,l}$ is 0 on all paths at $t = 0$ and increases by 1 at each computational interval. At $t = 1$, the starred edge is saturated and thus the three paths on it are saturated. At $t = 2$, $f^{r,l}$ keeps increasing on unsaturated paths.

utilizing the (single) shortest path as in most virtual circuit routing schemes²³. We do not aim at a multi-commodity flow optimization for routing²⁴, since the optimization is NP-hard when relaxing the shortest path constraint. Although an optimal solution for $f_{ij}^{r,l}$ is not derived here, we strive to provide efficient solutions for this routing task through sophisticated algorithmic design. Note that the queuing process on each node (station) is not considered in this study, which deals with arranging connection requests over sequential processing windows; the current scheme provides routing solutions within each processing window, assuming that a certain number of requests are processed in the same batch upon submission.

Routing scheme

Our stepwise routing scheme consists of both physical steps (in which quantum operations are physically conducted) and algorithmic steps (in which computations are performed). The overall scheme, summarizing both computational instructions and physical operations (text in red), is shown in Fig. 2.

At the start of each processing window (Step 0), the quantum network $G = (V, E)$ is reinitialized. The maximal edge capacity is C_0 on all edges. Entanglement generation is attempted between adjacent nodes. Since the entanglement generation probability p_{out} is less than one, the realized capacity on any edge is always less than C_0 . Multiple transmission requests $R = \{r\}$ are received. Each request consists of one source and one terminal node ($r = [S_r, T_r]$), the minimum demand of entanglement pairs $\bar{f}^r$, along with a weight factor w_r . There is no restriction on the choice of source/terminal nodes and the protocol supports all possible scenarios, including same S , same T , same $[S, T]$.

At Step 1, entanglement purification is carried out on each edge $(i, j) \in E$ having fidelity below the threshold F_{th} and a realized edge capacity (i.e., number of entangled pairs) $C_{ij} \leq C_0$. The initial edge fidelity (normally distributed at $N(F_{\text{mean}}, F_{\text{std}})$) and the probability of successful entanglements between adjacent stations p_{out} determine C_{ij} . Note the value of F_{th} may vary in practice for different network functionalities. To ensure enough bandwidth for each connection path, we apply a cap $l_{\max}$ to the number of possible paths traversing a single edge. Edges with residual capacity C_{ij} smaller than $l_{\max}$ are deactivated (see Fig. 3a). This

Table 1. Evaluation of routing performance.

Property	Notation	Explanation	Design feature	Comparison
Throughput	F	Total number of (successfully generated) entangled pairs	Multi-request	PU > PF ~ PS
Traffic	u_{ij}	Utilization of capacity on edge (i, j)	Multi-channel	PU > PF ~ PS
Delay	γ	Average path length stretching factor	Multi-path	PU ~ PF ~ PS
Fairness over requests	J_{req}	(Jain's index) Balance of capacity allocation over multiple requests	Multi-request	PF > PS > PU
Fairness over paths	J_{path}	(Jain's index) Balance of capacity allocation over multiple paths for one request	Multi-path	PF > PS > PU
Computational complexity	–	Of the algorithmic steps in the workflow (i.e., steps 2–4)	–	PU > PF > PS

ensures that the minimum allocated flow for a single path $f_{\min}$ (of a certain request), is always no less than one:

$$f_{\min} = \text{floor}\left(\frac{\min(C_{ij})}{I_{\max}}\right) \geq \frac{I_{\max}}{I_{\max}} = 1. \quad (1)$$

At Step 2, paths for different connection requests are identified in the revised graph G' . In order to ensure sufficient redundancy in routing in case of potential network failures, k shortest paths²⁵ are identified between each connection pair $r = [S_r, T_r]$. The value of k determines how relaxed in length the desired paths are: when $k = 1$ we only consider a single shortest path; more and longer paths will be utilized under a larger k . A practical and effective value of k should be determined based on specific network conditions, such as the number of requests $|R|$ allowed within a time window, the size of the lattice network $|E|$, and the maximum capacity C_0 . At minimum, k should be sufficiently large to satisfy the flow demand of each connection request, $k f_{\min} \geq \bar{f}^r$.

The identified paths are collected in the path information set $H = \{h_{ij}\}$. For each edge (i, j) along each path l identified for a request r , an information entry h_{ij} is added to the path information set H with the following tuple:

$$h_{ij} = [r, l, d, o], \quad (2)$$

where d is the path length and o the order of edge (i, j) in path l .

Step 3 is the core computational procedure of the routing scheme, in which the edge-wise capacity allocation scheduling is performed. In this study, we propose and compare three methods for this scheduling task: (i) proportional share (PS), (ii) progressive filling (PF) and (iii) propagatory update (PU), of which (i) and (ii) are based on well-known scheduling heuristics on classical data networks and (iii) is an algorithm combining the insights of (i) and (ii). Illustrations of the three allocation algorithms are shown in Fig. 3d–f. Below we give a qualitative overview of the three algorithms, whose details could be found in the Supplemental Materials.

In the proportional share method, the scheduling is edge-specific: the edge capacity C_{ij} is proportionally distributed to all paths that utilize (i, j) , in which case only local information on each edge is used. The progressive filling method implements the algorithm in Bertsekas and Gallager²², which guarantees max-min fairness (see sections below). All paths are treated equally, with flows starting at zero and experiencing uniform incremental increase; since in our setting quantum edges have integer capacities, the increment is 1 at each interval. Along the filling process, edges are gradually saturated, and paths utilizing saturated edges stop increasing in flow. Note our integer scheme follows a definition of edge saturation slightly different from that of non-integer flows. In our case, a saturated edge may still have a number of unallocated channels (i.e., the leftovers), but the amount is insufficient to be further allocated. The filling process terminates when no new attempt of flow increments could be realized. Note since flows increase in a fair manner, $f_{\min}$ is not functional in this method.

Combining ideas from these two conventional methods, we propose another method, the propagatory update method, which utilizes a global schedule table and assigns allocation on each edge in a backward, edge-specific manner. The algorithm calculates the summarized desired capacity on each edge from all paths that utilize it, using the information in the global schedule table, then allocates the unmet capacity demand to these paths. This requires making corresponding deductions from the desired flows whose sum exceeds the real edge capacity, and the schedule table is updated at each step. This is similar to the stepwise procedure in the progressive filling method, but there the increments in the schedule table are uniform. Unlike progressive filling, instead of treating all paths equally, proportional share and propagatory update methods differentiate between paths and adopt a similar two-stage strategy: the quota is first determined for each request, then for each path of different requests. The amount of capacities is distributed proportionally at both stages: at the request level, the allocated capacity for each request is proportional to the number of paths of such request that utilize that edge, to the power of β . At the path level, the allocated capacity for each path of a specific request is proportional to its path length, to the power of α . α and β are design features that control the exploitation-exploration tradeoff in path utilization, similar to the idea in ant colonization algorithms²⁶, and essentially determine the fairness of scheduling results. They are set as open parameters (not used in the PF method) whose values are experimented in simulations (see Simulation Results). As mentioned above, before the capacity allocation, the path list L' going through each edge is truncated to have $I_{\max}$ entries, for the PS and the PU methods. The criterion for truncation is the path length: keep short paths in the list and remove long paths; a path is always kept in the list if it is the sole path for a certain request. Details of the three scheduling methods are explained in Supplemental Materials.

The final allocated capacity flow of each path is determined at Step 4. The short-board constraint is applied: for path l of request r , the final flow $f^{r,l}$ is determined by the minimum allocated capacity for l among the values on all the edges in this path:

$$f^{r,l} = \min_{l \in L'} [C_{ij}^{r,l}], \quad \text{for } (i, j) \in l. \quad (3)$$

where $C_{ij}^{r,l}$ is determined at Step 3. Note this constraint is only necessary if proportional share scheduling is used at Step 3; for both progressive filling and propagatory update, the constraint is already implied during the allocation process.

Step 2–4 complete the algorithmic computation of the routing scheme. Based on the determined routing schedule output from Step 4, local quantum operations (e.g., entanglement swapping) are conducted accordingly and scheduled remote entanglements are established (Step 5). The realized flow for each request $f^r = \sum_{l \in L'} f^{r,l}$ is compared with the demand $\bar{f}^r$; unsatisfied requests are queued to the next processing window.

Performance measures

To evaluate the performance of our routing scheme, we consider three performance measures targeting the three features of our routing design (multi-request, multi-channel, and multi-path). In line with the performance evaluation of classical data networks, these measures characterize key properties in routing: throughput, traffic and delay. We also introduce metrics that characterize the fairness of scheduling. These measures (Table 1) target at different aspects of the routing task and are useful figures of merit in quantum network applications. Note entanglement fidelity is not considered among performance measures since our routing scheme ensures that the fidelity is always above the desired threshold.

Throughput – Besides fidelity, entanglement generation rate is another critical quantity typically used to evaluate the performance of quantum networks. In the current setting, the entanglement generation rate corresponds to the number of entangled pairs in the network that are successfully established during the fixed time window; in routing terminology, this is known as the throughput of the system. Considering the weights of different connection requests, we define system throughput as the total weighted flow aggregating over all paths and all requests:

$$F = \sum_r w_r \sum_l f^{r,l} p_{in}^{d_{r,l}-1} \quad (4)$$

Note this definition of throughput is slightly different from that on classical data networks due to a non-negligible possibility of failure (i.e. $p_{in} < 1$) during internal connections at each station (e.g., entanglement swapping).

Since our routing task is not a multi-commodity flow optimization problem, it is not guaranteed that requested demand $\vec{f}$ could always be satisfied. Therefore, it is desired in certain cases to ask to maximize the minimum flow of all requests, in which case we could use the alternative metric instead of F to evaluate system throughput: the minimum flow $F_{\min} = \min_r w_r \sum_l f^{r,l} p_{in}^{d_{r,l}-1}$. Tests suggest that in practice F and $F_{\min}$ are almost always positively correlated and thus could be used interchangeably. We thus keep F as the measure for throughput. As mentioned, throughput (of a single time window) is a straightforward figure of merit in quantum network applications (e.g., the key generation rate in quantum key distribution).

Traffic – To evaluate the amount of routed traffic on the network, we calculate the utilization u_{ij} of available capacity on each edge (i, j) :

$$u_{ij} = \frac{\sum_{r,l} f^{r,l}_{ij}}{C_{ij}} \in (0, 1]. \quad (5)$$

The overall traffic condition could be signaled via the mean U_{ave} and variance U_{var} of the utilization factors u_{ij} on all utilized edges. For an efficient routing performance, it is desired that the overall capacity utilization U_{ave} is high, i.e., little idleness, while the traffic is evenly distributed without the existence of many bottleneck edges, i.e., U_{var} is small.

Delay – To preserve sufficient transmission redundancy in routing, multiple paths of various length are exploited for each connection request. For each request r , we calculate the path length stretching factor γ_r , which is defined as the average length of all paths for this request (weighted by the flow), normalized by the shortest path length $d_{r,0}$:

$$\gamma_r = \frac{\sum_l f^{r,l} d_{r,l}}{d_{r,0} \sum_l f^{r,l}}, \quad (6)$$

where $d_{r,l}$ denotes the path length for path l under request r . The average stretching factor γ of the system is calculated from the γ_r of all requests; γ_r (and thus γ) is always no less than 1. In most

cases, quantum connections take nontrivial processing time during each hop on the path, hence longer paths can induce more delay in entanglement generation. If one assumes the same processing time on each edge in the network, then γ indicates the (relative) average delay of our routing results: $\gamma = 1$ corresponds to the situation where all flows take the shortest path, and thus the routing has minimum delay; a larger γ indicates a higher delay where circuitous paths are utilized. Delay may be an important metric in applications such as clock synchronization³.

Fairness – Jain's index²⁷ is adopted to evaluate the fairness of the scheduling scheme. The total scheduled flow for each request aggregates the amount of all paths, $f^r = \sum_{l \in L^r} f^{r,l}$, and the fairness of scheduling over requests is thus given by:

$$J_{req} = \frac{(\sum_{r \in R} w_r f^r)^2}{|R| \sum_{r \in R} (w_r f^r)^2}. \quad (7)$$

J_{req} falls between $[0, 1]$ with $J_{req} = 1$ representing complete fairness. Evidently, the progressive filling algorithm obtains the highest fairness ($J_{req} \approx 1$) among the three capacity allocation methods, guaranteeing max-min fairness²⁸; values of α and β in proportional share and propagatory update algorithms affect their fairness, which is often substantially below 1 for these two methods. A large J_{req} ensures that connection requests within a processing window are processed in a relatively fair manner, which is clearly desirable. In a way similar to equation (7), one could also calculate the fairness index with respect to each path, J_{path} :

$$J_{path} = \frac{(\sum_{r \in R} w_r \sum_{l \in L^r} f^{r,l})^2}{|R| \sum_{r \in R} w_r^2 \sum_{l \in L^r} (f^{r,l})^2}. \quad (8)$$

A high fairness of flows on different paths is also desirable, in which case one avoids the situation where projected flows are concentrated on a small number of paths. As one would imagine, the routing will be less robust to potential network failures if one or few paths dominate the flow.

Parameter determination

At the start of each processing window, new connection requests R are received, and the network is reinitialized with edge capacities C_{ij} reset according to entanglement purification results. System parameters ($X = \{l_{max}, k, \alpha, \beta\}$ for PS and PU, and $X = k$ for PF) need to be determined specifically for the current time window. As one would imagine, parameters are chosen so as to optimize a desired objective function concerning the routing performance of the network, which could be constructed from the performance measures discussed above. The specific form of the objective function could be determined according to specific requirements of a given application. For example, a general-form objective function may simultaneously consider the throughput F , traffic (in terms of U_{ave} and U_{var}), and delay γ of routing results, using π_1, π_2, π_3 to denote the relative weight of each measure. The corresponding optimization problem could then be formulated as

$$\begin{aligned} X^* &= \arg \max_{\{l_{max}, k, \alpha, \beta\}} F + \pi_1 U_{ave} - \pi_2 U_{var} - \pi_3 \gamma \\ \text{s.t.} \quad &G', R, C_{ij} (i, j) \in E' \\ &l_{max}, k \in \mathbb{Z}^+ \end{aligned} \quad (9)$$

As one could see, the objective function is flexible and may contain multiple terms, instead of using the throughput or delay as the sole objective. Note the above optimization problem is distinct from the multi-commodity flow problem in optimal routing on traditional data networks. Requested demands $\vec{f}$ do not serve as constraints of the problem, and in our setting we allow that some demands are unmet. The queuing process and the weighting mechanism are designed to deal with unsatisfied demands. Depending on the computational capacity of the

scheduler (the central processor), this optimization could be solved either through brute force searching over $\{l_{\max}, k, a, \beta\}$ (thanks to the relatively small degree of freedom), or by applying some machine learning techniques. For example, one could use supervised learning or reinforcement learning, where the connection requests R and the weighted adjacency matrix of the realized graph G' are input, and the learned parameters X' are output. Routing performance could be evaluated either in a hard manner (i.e., success/fail, for supervised learning), or in a soft manner (i.e., as rewards, for reinforcement learning). The advantage of using machine learning solvers is that now one could deal with multiple terms in the objective function more flexibly by designing sophisticated learning workflow. Essentially, this allows the learning of high-dimensional properties to evaluate the routing performance, beyond the straightforward performance measures discussed above. Since some key aspects in real quantum network applications are not modeled in the current simulation framework, notably the existence of network failures, it is then desired that an effective machine learning workflow is assembled for the parameter determination of this routing scheme; we leave this question open for future work.

Computational complexity

We briefly discuss the computational complexity of the algorithmic steps in the workflow (i.e., Steps 2–4). Determining the shortest paths incurs the largest computation cost at Step 2, with a time $O(|R|kV^2) = O(|R|kE^{1.5})$ for square lattices. Step 4 requires a cost $O(|R|k)$ to calculate the allocated capacity for each path. The

complexity of Step 3 depends on the specific method. The time is $Q_{PS} = O(E|R|l_{\max}) \sim O(E|R|k)$ for proportional share since $l_{\max} \sim k$, while it is $Q_{PF} = O(C_0)$ for progressive filling. The complexity of propagatory update is higher due to its iterative nature. In the worst case, the max flow on each path will decrease from the maximum value C_0 to the minimum value $f_{\min}$ step by step. Therefore, for $|R|$ requests and k paths of each request, the no-improvement counter will be reset to 0 at $|R|k(C_0 - f_{\min})$ times, resulting in a cost $O(|R|k(C_0 - f_{\min})E)$. Since $C_0 - f_{\min} \sim C_0$, the complexity of propagatory update is approximately:

$$Q_{PU} \sim O(E|R|kC_0) = C_0Q_{PS}, \quad (10)$$

which is in general the slowest among the three methods. Overall, summarizing the three algorithmic steps, the complexity Q_{alg} of the scheduler is:

$$Q_{alg} = O(E^{1.5}|R|k) + O(E|R|k) + O(|R|k) = O(E^{1.5}|R|k)$$

for proportional share, $O(E^{1.5}|R|k + C_0)$ for progressive filling, and $O((E^{0.5} + C_0)E|R|k)$ for propagatory update.

Simulation results

We performed extensive simulations to study the effect of design parameters $X = \{l_{\max}, k, a, \beta\}$, and to demonstrate the robustness of the routing design under different assumptions on entanglement fidelity, $F_{\text{mean}}, F_{\text{std}}, F_{\text{thr}}$, and operation success probabilities $p_{\text{in}}, p_{\text{out}}$. We characterize the performance of the simulation outcomes from multiple perspectives, using the established metrics: the throughput (total flow F), the traffic (U_{ave} and U_{var}), the delay (path

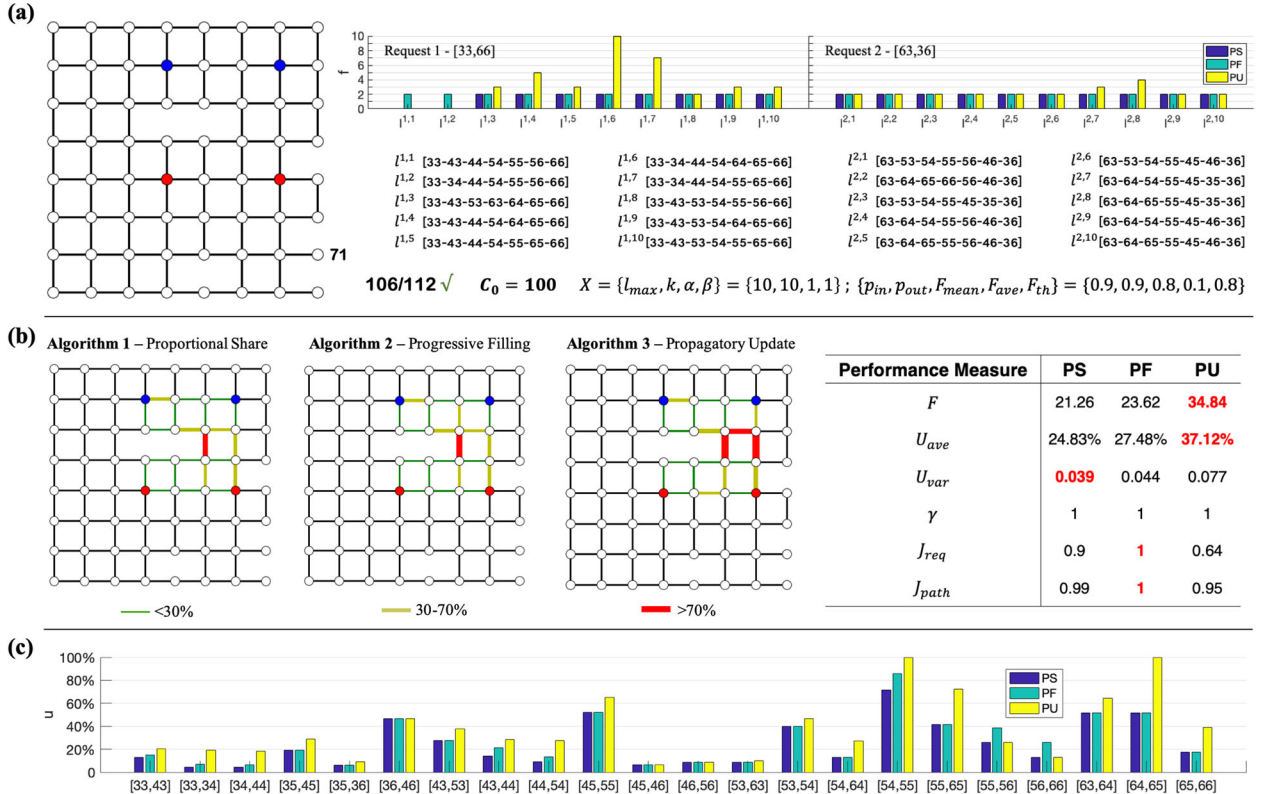


Fig. 4 Baseline case results. **a** (left) Network topology after entanglement purification; (right) 10 connection paths for each request (bottom) and the allocated flow for each path under the three scheduling schemes (top). Here the nodes are represented by a two digit number and the bottom left corner is denoted as 00. An example of node 71 is labeled in the graph plot. **b** Comparison of routing performance. Traffic plots (left) and performance measures (right). We show the traffic plot in a 8-by-8 network where edge color and width represent the capacity utilization of the corresponding edge (<30% in green, 30–70% in yellow and >70% in red, large width corresponds to larger utilization). **c** Capacity utilization on (utilized) edges under the three schemes. For simplicity sake, in all simulations hereafter we set the initial capacity of each edge to be $C_0 = 100$, and the fidelity distribution among edges follows a normal distribution with mean $F_{\text{mean}} = 0.8$ and standard derivation $F_{\text{std}} = 0.1$.

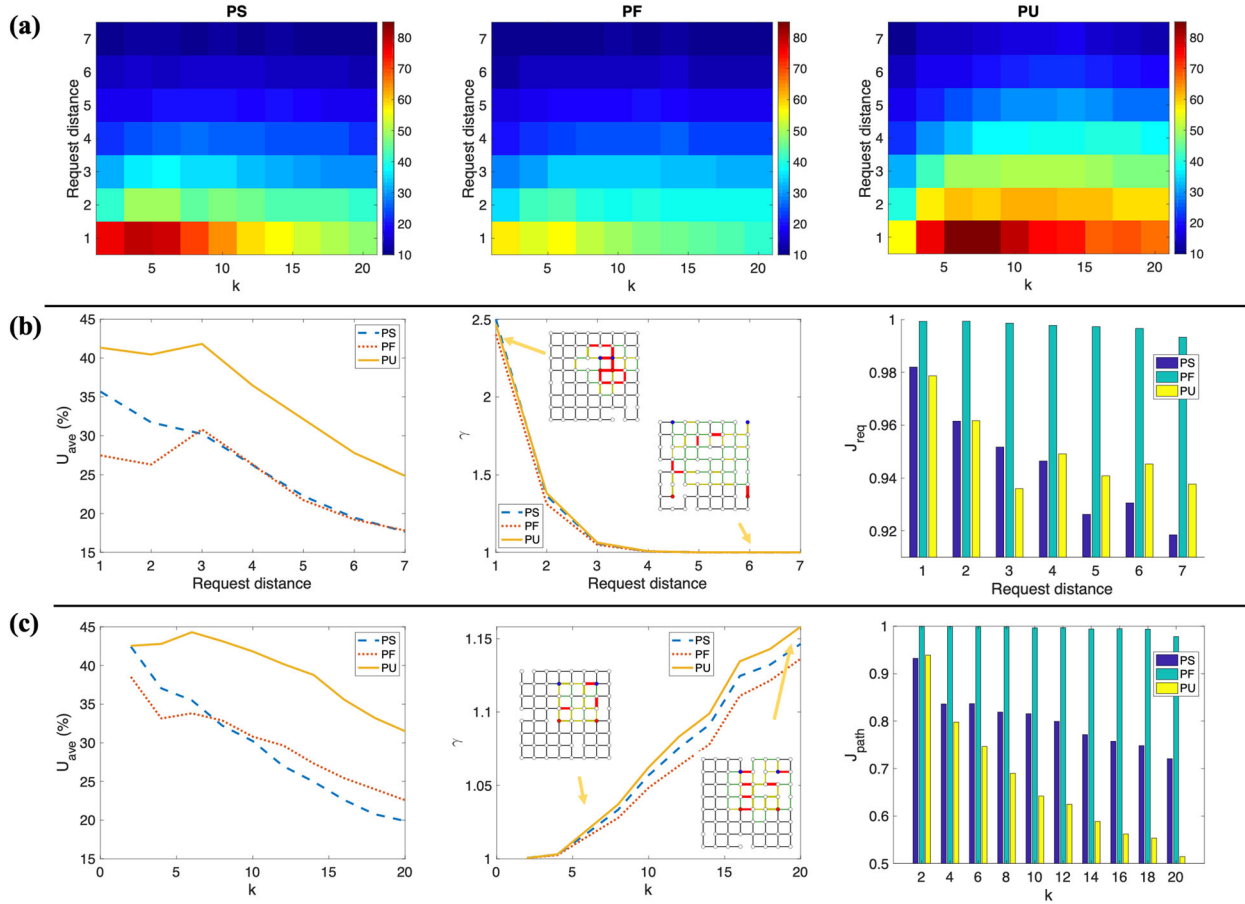


Fig. 5 Algorithm performance as a function of number of shortest paths k and request distance. a Heat map of the throughput F for the two requests. **b** Routing performance with respect to request distance at $k = 10$. U_{ave} , γ , and J_{req} plots are shown here. **c** Routing performance with respect to k at request distance 3. U_{ave} , γ , and J_{path} plots are shown here. Examples of traffic plots with different parameters for the PU algorithm are shown in the inset of γ plots. The other system parameters are $\{l_{max}, \alpha, \beta\} = \{15, 1, 0\}$.

stretching factor γ), and fairness over requests (Jain's index J_{req}) and paths (J_{path}). We focus on the comparison among the three scheduling schemes (PS, PF, PU in short). To limit the simulation complexity and better understand system parameters, we consider an 8×8 square lattice, with two connection requests $[S_i, T_i], i = 1, 2$ with same weight and identical request distance. Here the request distance is defined as the Manhattan distance along one direction (the Manhattan distance along the two directions is same in our simulations for simplicity). Under this baseline case, we first demonstrate a sample result of our general routing scheme. Then we study the choice of k , which is used in all three scheduling schemes; tests show that it is the most important design parameter in determining the performances of the three algorithms. Later we study the other three parameters $\{l_{max}, \alpha, \beta\}$ used in PS and PU, with a focus on the fairness of routing as compared to PF which guarantees the max-min fairness. Finally we study the system's robustness under different conditions of quantum parameters. For each set of parameters, 200 simulation runs are performed, and the results are averaged to account for the stochastic nature in the quantum setup. Beyond the baseline case scenario, alternative network topologies (hexagonal/triangular), arbitrary connection requests (varied request number $|R|$ and arbitrary distance) and potential network failures (edge/node failure) are studied in further simulations.

Baseline case. In Fig. 4, we demonstrate a sample result of our routing scheme. Initially, $C_0 = 100$ entangled pairs are prepared at each station. After entanglement purification (Step 1), 106/112 edges in the 8×8 square lattice are maintained, under quantum

parameters $\{F_{mean}, F_{std}, F_{thr}, p_{in}, p_{out}\} = \{0.9, 0.9, 0.8, 0.1, 0.8\}$ (left, Fig. 4a). Two connection requests ($[33, 66], [63, 36]$) are received, with source/terminal nodes shown in red/blue; the first/second coordinate indicates the horizon/vertical axis (see node 71 in Fig. 4a). System parameters are $X = \{l_{max}, k, \alpha, \beta\} = \{10, 10, 1, 1\}$. Ten paths are identified for each request (Step 2); after Step 3 and Step 4, the routing results under three scheduling schemes are compared. Figure 4b shows the traffic on the network in a graphic view (left). We compare performance measures of the three scheduling algorithms (see also Table 1): PU obtains the largest throughput, and it best exploits edge capacities; indeed, it shows that the scheduled flows on paths are almost always the highest on PU (right top, Fig. 4a, so are the capacity utilizations on different edges (Fig. 4c). PS obtains the smallest variance in edge capacity utilizations, suggesting that the routed traffic is more evenly distributed in PS than in PU or PF; yet PS has the poorest throughput F and average capacity utilization U_{ave} . PF has less throughput and capacity utilization than PU, but as expected, the scheduling is completely fair, with respect to both requests (J_{req}) and paths (J_{path}). PU is the least fair scheme among the three and it can be attributed to the large flow on few paths (e.g., $f_{1,6}, f_{1,7}, f_{2,8}$ in Fig. 4a). Various tests are simulated under different parameter sets, and results suggest that the baseline case is representative of the three schemes' performance; the comparison of performance in Fig. 4b largely hold in an aggregated sense.

Dependence on request distance and choice of k . In Fig. 5a we show the total flow F of the three algorithms. The throughput, as well as the minimum flow of the two requests, decay

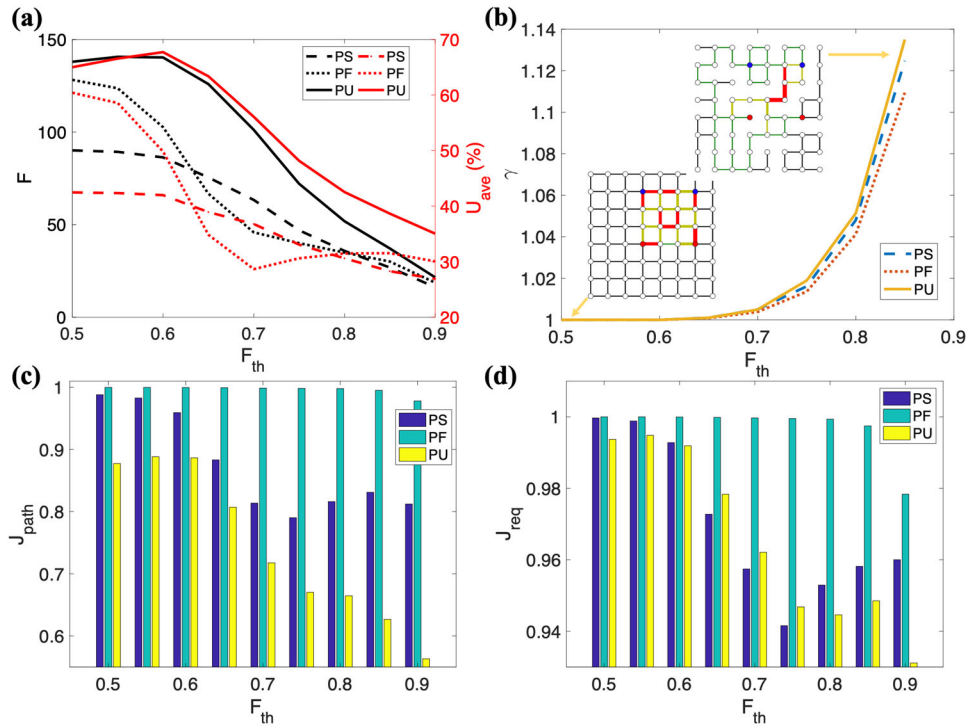


Fig. 6 Protocol performance with respect to fidelity threshold. **a** Total flow and average capacity utilization. Inset shows the minimum flow of the two requests. **b** Normalized path stretching factor. Traffic plots at two extreme case for the propagatory update algorithm are presented. **c, d** Fairness over paths (c) and requests (d). The fidelity distribution among edges follow the normal distribution $N(0.8, 0.1)$ and the request distance is fixed to be 3 here. The other system parameters are $\{l_{max}, \alpha, \beta\} = \{15, 1, 0\}$.

exponentially as the distance between request pairs grows. The PU algorithm results in obviously larger throughputs (larger number of entangled pairs), while the other two algorithms are comparable to each other. We notice that the throughput will reach a maximum at moderate k values. This can be explained as slightly higher k values provide more freedom in generating large flows, but beyond the optimum k point there will be more paths along each edge on average, leading to congestion on the utilized edges and resulting in more bottlenecks for each path thus decreasing the throughput. We also observe that with large request distance, k has less of an effect, since there are enough routes to select and less edges where congestion can occur.

Figure 5b shows the behavior of the other metrics with respect to request distance. As the number of paths between request pairs becomes exponentially large with the request distance, for fixed k the traffic is spread out on all edges of these paths, leading to decreasing capacity utilizations. Similarly, at large request distance, there is no need for the flows to take detours and all utilized paths are indeed shortest paths which yields $\gamma = 1$. This is clearly depicted in the traffic plots. We also observe that the PU method is superior in utilizing the edge channels and has better capacity utilizations due to information propagation among different edges during its iterative process. All the algorithms demonstrate excellent performance in dealing with multiple requests. In particular, the PF algorithm, due to its step-by-step filling nature, performs excellently in balancing the requests and paths.

For fixed requests as shown in Fig. 5c, choosing a larger k means that the routing algorithms tend to explore more edges, which results in a lower average capacity utilization. On the same note, circuitous paths will be utilized in routing, which increases the normalized path length thus the time delay. Meanwhile, for the PS and PU algorithm, the flow variance among different paths will increase, leading to decreasing fairness over paths. These

results point out that the parameter k is crucial in determining path properties.

Effect of fidelity threshold. Finally, we discuss the fidelity threshold that can affect the entanglement purification steps (thus the realized capacity C_{ij}). Specific tasks in quantum network would demand different error rates or fidelity thresholds and the corresponding entanglement purification will modify the topology of the network in different ways. In Fig. 6 we show the dependence of the protocol performance on fidelity threshold. As the fidelity bound raises, more purification steps are needed, monotonously decreasing the edge capacity and hence the throughput. Intuitively, one might expect that for high fidelity threshold, in general edges will have less realized capacity C_{ij} which can be fully utilized, leading to large capacity utilization. However, as shown in Fig. 6a, the average capacity utilization follows the same trend as the throughput. This is due to the fact that for fixed k , the edge removal due to repeated purification steps will force the routing scheme to explore more edges in the network thus decreasing the overall capacity utilization. This is clearly shown in the traffic plots (Fig. 6b). Similarly, for high fidelity threshold, the normalized path length is clearly larger than unity for the three algorithms and the flow among different paths tends to have large variance, thus small fairness over paths for PS and PU (Fig. 6d). Nevertheless, the two requests are well balanced with high fairness over request for all allocation algorithms (Fig. 6d).

Potential network failures. One critical feature of our routing scheme is multi-path, and the claim is that by utilizing more than one path, the routing will be more robust to potential network failures. The network failures can be induced by the breakdown of a central station, interruption of the signal transmission on physical channels, or other adversary situations in the real world. We simulate both edge failures (an edge is dead) and node failures (a station is dead; all edges linked to it are dead) to test the scheme's robustness (Fig. 7). These two failure modes are among the most common network issues, which may also take place due

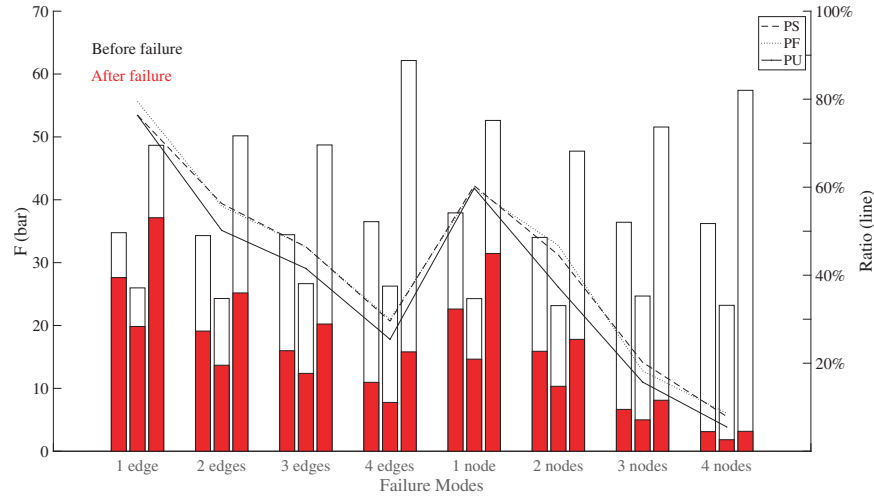


Fig. 7 Robustness of routing results against network failures. Comparing system throughput before and after the failure of the three scheduling schemes. Different failure modes (1–4 edges failure, 1–4 nodes failure) are simulated.

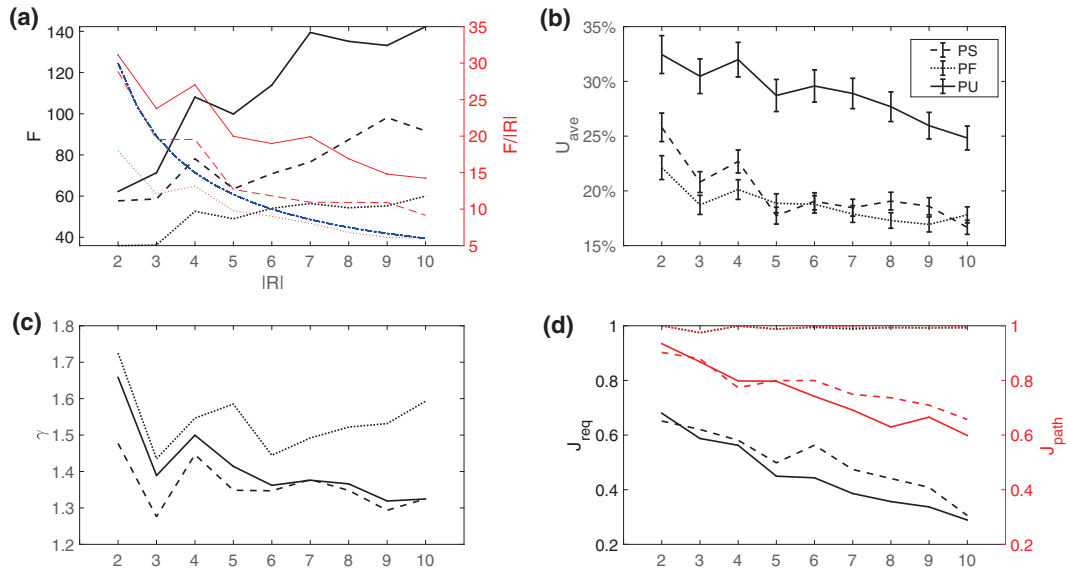


Fig. 8 Performance under different number of requests (2–10) per window. **a** Throughput F and normalized throughput $F/|R|$; blue curve shows the reference line $60/|R|$. **b** Traffic mean U_{ave} and variance U_{var} (which is shown as the error bars with reduced scale of factor 2). **c** Delay γ . **d** Fairness J_{req} and J_{path} .

to security concerns (such as external eavesdroppers). The system throughputs F before and after the failure are compared for each scheduling scheme, under different failure modes (1–4 edges failure, 1–4 nodes failure; edges and nodes are chosen from those being utilized). 20 simulations are averaged at each data point under baseline case parameters (see Fig. 7). Results show that, as expected, as the failure mode gets more and more serious, the system achieves smaller and smaller throughput, but is still working; after the shutdown of 4 out of the 16 utilized stations, the system still maintains some outputs. We also notice that of the three schemes, PS shows the greatest robustness, slightly greater than PF, whereas PU's superiority in throughput F gradually vanishes in face of network failures. Similar robustness conditions are identified on other performance measures (see Supplemental Material).

Number of requests per window. Within each processing window, we simulate 2–10 random connection requests (arbitrary $[S, T]$ pairs) and compare routing performances (Fig. 8). 20 simulations are averaged at each data point under based case

parameters (see Fig. 3). With $|R|$ increasing, the system becomes more crowded: F goes up whereas $F/|R|$ goes down, but the degradation curvature is better than inverse-proportional (blue line); the superiority of PU in throughput is always maintained. Meanwhile, the network traffic is more diluted, and the routing less fair for PU and PS, with U_{ave} , U_{var} , J_{req} , and J_{path} falling quasi-linearly. There is no consistent change in average routing delay γ ; PU seems to produce less delayed routing under more requests, but the result is not significant.

DISCUSSION

We benchmarked the performance of the routing scheme under different parameter regimes. Here we highlight differences in resource allocation for the three scheduling schemes. Performance results show that the three schemes have different advantages: progressive filling is the fairest method, whereas the fairness of the two proportional methods is compromised to a nontrivial extent, and depends on the system parameters α, β . In

terms of system throughput and capacity utilization, the new propagatory update scheme stands out by a large margin, and the superiority is largely maintained under all network conditions, even in face of network failures. However, as a tradeoff, the increased throughput are not attributed to all requests in a fair manner, and propagatory update routing results demonstrate the largest variance on edge utilizations as well. The least efficient scheme is proportional share, which generates the least throughput. However, the variance of edge utilization is smaller than in propagatory update, and it is more robust against network failures. This suggests that this scheme utilizes network resources in the most economical way. Also, since a global information table is not maintained (hence not subject to errors) and proportional allocations are performed only at the local scale, the proportional share may also be the most robust scheme against network errors on top of failures.

We now proceed to present the scope of use, limitations and potential future extensions of the routing scheme. We have demonstrated the performance of our routing scheme with a 8×8 square lattice network with edge capacity $C_0 = 10^2$ and <10 requests per time window. In reality, the implementation scale of quantum networks will be determined by various physical, geological and economic constraints. Current lab-scale quantum networks are very small ($\sim 2 \times 2$) with very limited edge capacity C_0 (only few qubits), since the preparation of entangled pairs is extremely costly; for our routing scheme to be put in use, huge leaps in quantum engineering are necessary. Practically, the lattice size and edge capacity should be decided by the specific functionality of the quantum network, and the number of requests being processed within a time window are determined by service considerations. Also note that alternative lattice topologies (e.g., hexagonal, triangular) could be experimented in real applications; current results apply to square lattice, yet it is believed that similar results could be obtained on other common regular lattice structures.

We further note that, in the space domain, we are assuming a central processor in the network and global link-state information is broadcast to each node. Under restricted cases, however, only local link-state knowledge (link with the nearest neighboring nodes) is available at each node. For example, as the size of the network $G(V, E)$ increases, the classical communication time is non-negligible and can be far beyond the quantum memory lifetime. In this scenario, instead of finding the paths with a global processor, one can perform entanglement swapping on two local quantum memories in a way such that the sum of their distance to the request pair can be minimized, as shown in Pant et al.²¹. However, in a more complex quantum network where multi-request, multi-path and multi-channel are involved, searching for an efficient routing strategy remains elusive and might be interesting for future study.

In the time domain, the current protocol is used for processing a certain number of connection requests within one active time window; it is expected that a queueing model²⁹ might be constructed to tackle the network functioning beyond the processing time window. Such queueing system should provide guidelines for coordinating the reception of incoming connection requests with the recycling of entangled pairs, in which case, processing windows could essentially overlap, and one does not need to wait for the end of one batch of requests before starting the next window. In this scenario, other concerns in routing design might arise, for example, dealing with network congestions, e.g. subsequent re-distribution of capacities after initial allocation³⁰. Since these issues have been intensively studied in classical network, discussions on such high-level infrastructures of the current protocol are not included in this paper and are left for future studies.

Next we point out that in the above protocol we consider entanglement purification only at the first step, that is, only

between adjacent nodes. As successive entanglement swapping operations are imperfect, they might degrade the fidelity of generated remote entanglement. To overcome this drawback, one can adopt a nested protocol^{14,20} where repeated purification is performed after each entanglement swapping step to ensure that the remote entangled states have regained a sufficiently large fidelity. Then an updated version of the routing scheme in which the resource consumption during successive entanglement purification is taken into account might be designed.

Finally, while here we consider entanglement between two nodes, i.e., Bell states, an extension to GHZ states and local GHZ projection measurement might bring advantages in generating remote entanglement^{31–34}. For example, even if the GHZ states are more difficult to prepare and more fragile to environment noise, they provide more configurations on a quantum network, thus creating more complex topology which can be unreachable for Bell states. The compromise between this gain and its cost can in principle benefit our entanglement routing designs. Similarly, the GHZ projection measurement in a local station can glue more than two remote stations together and modify the network topology. The routing strategy that utilize these operations will be an interesting topic to study, both mathematically and physically.

In conclusion, we have proposed an efficient protocol that tackles the routing problem on quantum repeater networks. We have shown that entanglement purification, which increases entanglement fidelity, can modify the network topology and affect the following routing design. For general multi-request, multi-path, and multi-channel problems, we developed three algorithms that can allocate limited resource to efficiently generate entangled pairs between remote station pairs.

Among the first attempts to tackle this complex problem, our routing scheme provides a solution to exploit the full capability of complex quantum networks, where efficient entanglement generation of remote station pairs are desired. The routing design here may also inspire future work that combine the quantum information science with classical network theory.

DATA AVAILABILITY

Datasets (for the simulation results) are available from the corresponding authors upon reasonable requests.

CODE AVAILABILITY

Codes (for the reproduction of simulation results) are available from T.L. at https://github.com/TimothyLi0123/Qnetwork_Routing.

Received: 8 February 2020; Accepted: 17 December 2020;

Published online: 27 January 2021

REFERENCES

1. Wehner, S. & Elkouss, D. & Hanson, R. Quantum internet: a vision for the road ahead. *Science* **362**, 9288 (2018).
2. Gisin, N. & Thew, R. Quantum communication. *Nature Photon.* **1**, 165–171 (2007).
3. Kómár, P. et al. A quantum network of clocks. *Nature Phys.* **10**, 582–587 (2014).
4. Denchev, V. S. & Pandurangan, G. *Distributed quantum computing: a new frontier in distributed systems or science fiction?* p. 77–95 (SIGACT News, 2008).
5. Beals, Robert et al. Efficient distributed quantum computing. *Proc. Royal Society A* **469**, 20120686 (2013).
6. Nickerson, N. H., Fitzsimons, J. F. & Benjamin, S. C. Freely scalable quantum technologies using cells of 5-to-50 qubits with very lossy and noisy photonic links. *Phys. Rev. X* **4**, 041041 (2014).
7. Humphreys, P. et al. Deterministic delivery of remote entanglement on a quantum network. *Nature* **558**, 268–273 (2018).
8. Dahlberg, Axel et al. A link layer protocol for quantum networks. In *Proc. ACM Special Interest Group on Data Communication, SIGCOMM'19*, p. 159–173 (ACM, New York, NY, USA, 2019).

9. Tchebotareva, A. et al. Entanglement between a diamond spin qubit and a photonic time-bin qubit at telecom wavelength. *Phys. Rev. Lett.* **123**, 063601 (2019).
10. Dréau, A., Tchebotareva, A., El Mahdaoui, A., Bonato, C. & Hanson, R. Quantum frequency conversion of single photons from a nitrogen-vacancy center in diamond to telecommunication wavelengths. *Phys. Rev. Appl.* **9**, 064031 (2018).
11. Li, C. & Cappelaro, P. Telecom photon interface of solid-state quantum nodes. *J. Phys. Commun.* **3**, 095016 (2019).
12. Briegel, H.-J., Dür, W., Cirac, J. I. & Zoller, P. Quantum repeaters: the role of imperfect local operations in quantum communication. *Phys. Rev. Lett.* **81**, 5932–5935 (1998).
13. Cirac, J. I., Duan, L.-M., Lukin, M. D. & Zoller, P. Long-distance quantum communication with atomic ensembles and linear optics. *Nature* **414**, 413 (2001).
14. Childress, L., Taylor, J. M., Sørensen, A. S. & Lukin, M. D. Fault-tolerant quantum repeaters with minimal physical resources and implementations based on single-photon emitters. *Phys. Rev. A* **72**, 052330 (2005).
15. Sangouard, N., Simon, C., de Riedmatten, H. & Gisin, N. Quantum repeaters based on atomic ensembles and linear optics. *Rev. Mod. Phys.* **83**, 33–80 (2011).
16. Muralidharan, S. et al. Optimal architectures for long distance quantum communication. *Sci. Rep.* **6**, 20463 (2016).
17. Bennett, C. H. and Brassard, G. Quantum cryptography: public key distribution and coin tossing. *Theor. Comput. Sci.* **560**, 7–11 (2014).
18. Acín, A., Ignacio Cirac, J. & Lewenstein, M. Entanglement percolation in quantum networks. *Nat. Phys.* **3**, 256–259 (2007).
19. Perseguers, S., Lewenstein, M., Acín, A. & Cirac, J. I. Quantum random networks. *Nat. Phys.* **6**, 539–543 (2010).
20. Perseguers, S., Lapeyre, G. J., Cavalcanti, D., Lewenstein, M. & Acín, A. Distribution of entanglement in large-scale quantum networks. *Rep. Prog. Phys.* **76**, 096001 (2013).
21. Pant, M. et al. Routing entanglement in the quantum internet. *npj Quant. Inf.* **5**, 25 (2019).
22. Bertsekas, D. P. and Gallager, R. G. *Data Networks*. 1992.
23. Ramakrishnan, K. G. & Rodrigues, M. A. Optimal routing in shortest-path data networks. *Bell Labs Tech. J.* **6**, 117–138 (2001).
24. Frank, H. & Chou, W. Routing in computer networks. *Networks* **1**, 99–112 (1971).
25. Yen, J. Y. Finding the k shortest loopless paths in a network. *Manag. Sci.* **17**, 712–716 (1971).
26. Dorigo and Birattari. *Ant Colony Optimization* (Springer US, 2010).
27. Jain, R. K., Chiu, D.-M. W. & Hawe, W. R. *A Quantitative Measure Of Fairness And Discrimination* (Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA, 1984).
28. Le Boudec, J.-Y. *Rate Adaptation, Congestion Control And Fairness: A Tutorial*. Web page, November, 2005.
29. Gross, D., Shortle, J. F., Thompson, J. M. and Harris, C. M. *Fundamentals of Queueing Theory*. 4th edn (Wiley-Interscience, New York, NY, USA, 2008).
30. Yan, G., Zhou, T., Hu, B., Fu, Zhong-Qian & Wang, Bing-Hong Efficient routing on complex networks. *Phys. Rev. E* **73**, 046108 (2006).
31. Pirker, A., Wallnöfer, J. & Dür, W. Modular architectures for quantum networks. *New J. Phys.* **20**, 053054 (2018).
32. Dahlberg, A. & Wehner, S. Transforming graph states using single-qubit operations. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.* **376**, 20170325 (2018).
33. Pirker, A. & Dür, W. A quantum network stack and protocols for reliable entanglement-based networks. *N. J. Phys.* **21**, 033003 (2019).
34. Hahn, F., Pappa, A. & Eisert, J. Quantum network routing and local complementation. *npj Quant. Inf.* **5**, 76 (2019).

ACKNOWLEDGEMENTS

We thank Prof. Eytan Modiano at MIT LIDS for helpful discussions. This work is supported by NSF EFRI-ACQUIRE Award 1641064.

AUTHOR CONTRIBUTIONS

C.L. and T.L. conceived the problem and T.L. developed the algorithms. C.L. and T.L. conducted the analysis and C.L. organized the manuscript. P.C. supervised the project. All authors discussed the results and contributed to the final manuscript.

COMPETING INTERESTS

The authors declare no competing interests.

ADDITIONAL INFORMATION

Supplementary information is available for this paper at <https://doi.org/10.1038/s41534-020-00344-4>.

Correspondence and requests for materials should be addressed to T.L. or P.C.

Reprints and permission information is available at <http://www.nature.com/reprints>

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

© The Author(s) 2021

Supplementary Information for “Effective routing design for remote entanglement generation on quantum networks”

Changhao Li^{1,2,4}, Tianyi Li^{*3,4}, Yi-Xiang Liu^{1,2}, and Paola Cappellaro^{†1,2}

¹*Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, MA 02139*

²*Department of Nuclear Science and Engineering, Massachusetts Institute of Technology, Cambridge, MA 02130*

³*System Dynamics Group, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139*

⁴*The authors contribute equally to the work.*

^{*}tianyi@mit.edu

[†]pcappell@mit.edu

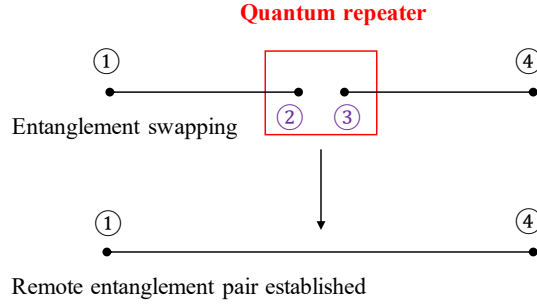
Supplementary Notes 1: Entanglement swapping and purification

We first briefly review the basic elements of entanglement swapping and purification for the convenience of readers.

A. Entanglement swapping

Due to the optical loss during information transmission, entanglement swapping would be essential in generating long distance entanglement. Here we consider memory-assisted quantum repeaters, as shown in Fig. 1. The quantum memory 2 and 3 are in the same station hence local bipartite operations can be performed at low cost. At the beginning, we create Bell states between memory 1 (3) and 2 (4), the state of which can be written as:

$$|\psi_{12}^-\rangle|\psi_{34}^-\rangle = \frac{1}{2}(|\psi_{14}^+\rangle|\psi_{23}^+\rangle - |\psi_{14}^-\rangle|\psi_{23}^-\rangle - |\phi_{14}^+\rangle|\phi_{23}^+\rangle + |\phi_{14}^-\rangle|\phi_{23}^-\rangle) \quad (1)$$



Supplementary Figure 1: Illustration of quantum repeaters. The black dots (lines) represent quantum memories (optical links).

Then we can see that a joint Bell state measurement on memory 2 and 3 will project memory 1 and 4 into Bell states. For example, the measurement result of $|\psi_{23}^-\rangle$ would reveal that we create a non-local entangled states $|\psi_{14}^-\rangle$. Technically, this entanglement can be generated with methods such as two-photon interference conditioned on the click patterns of the detectors in between. We point out that with quantum repeaters, the time needed to generate remote entanglement (with certain fidelity) would scale polynomially or sub-exponentially as a function of distance.

B. Entanglement purification

Now we introduce the principle of entanglement purification with a simple example. We consider two pairs of qubits shared by Alice and Bob, the state of which is described by the density matrix $\rho_{AB}^j = f|\Psi^+\rangle\langle\Psi^+| + (1-f)|00\rangle\langle 00|$, where $j=1,2$ and $|\Psi^+\rangle$ is the state to distill. By applying the local

C-NOT gates between the two qubits located at Alice (Bob), we can find the new density matrix:

$$\rho = f^2|\Psi^+\rangle_1\langle\Psi^+| \otimes |\Phi^+\rangle_2\langle\Phi^+| + \quad (2)$$

$$(1-f)^2|00\rangle_1\langle 00| \otimes |00\rangle_2\langle 00| + \quad (3)$$

$$f(1-f)|00\rangle_1\langle 00| \otimes |\Psi^+\rangle_2\langle\Psi^+| + \quad (4)$$

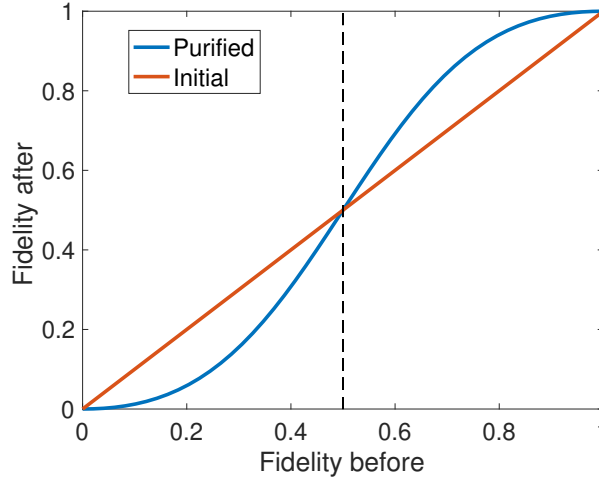
$$\frac{f(1-f)}{2}(|0101\rangle + |1010\rangle)_{1,2}(\langle 0101| + \langle 1010|) \quad (5)$$

By performing measurements on the two target qubits, the state of pair one will become $\rho_{AB}^1 = \frac{f^2|\Phi^+\rangle\langle\Phi^+| + (1-f)^2|00\rangle\langle 00|}{f^2 + (1-f)^2}$ conditioned on the same measurement result (00 or 11) of pair 2 (with success probability $P = f^2 + (1-f)^2$). The fidelity of state $|\Psi^+\rangle$ after the i -th distillation will be $f^i = \frac{(f^{i-1})^2}{(f^{i-1})^2 + (1-f^{i-1})^2}$. As shown in Fig. 2, if the initial f^{i-1} is above 0.5, highly entangled states can be generated. Note this protocol only involves inexpensive local operations. However, the purification of state fidelity is at the cost of entanglement generation rate (since $P < 1$) and multiple entanglement pairs.

In the limit of $1 - f_i \ll 1$, the purification curve will simply be $1 - f_{i+1} = (1 - f_i)^2$ while the success probability is f_i . To obtain one EPR pair with the desired threshold F_{th} , the average number of fresh EPR pairs required will be:

$$n_r = 2^m \frac{1}{\prod_i^m P_i}, m = \frac{\log(1 - F_{th})}{2\log(1 - \langle F \rangle)} \quad (6)$$

where $\langle F \rangle$ is the initial EPR fidelity and m is the number of purification steps one needs to perform.



Supplementary Figure 2: Purification curve. Above the 0.5 threshold, the purification process will gain fidelity of desired state.

Supplementary Notes 2: Capacity allocation scheduling

In *Step 3* of the routing protocol, capacity allocation scheduling is carried out. We propose and compare three methods for this task: (i) *proportional share*, (ii) *progressive filling* and (iii) *propagatory update*, of which (i) and (ii) are based on well-known scheduling heuristics on classical networks and (iii) is our invention.

A. Algorithm 1 *Proportional Share*

The difference of the *proportional share* and the *propagatory update* methods lies in the direction of managing capacity allocation. The first method forwardly divides the realized (maximum) edge capacity $C_{i,j}$ into proportions, according to the weights of divisors. At the first stage, the division is carried out on different requests. On edge (i, j) , the capacity allocated for request r is given by:

$$C_{i,j}^r = C_{i,j} \frac{w_r(\#r_{i,j})^{-\beta}}{\sum_r w_r(\#r_{i,j})^{-\beta}}, \quad (7)$$

where $\#r_{i,j}$ is the number of paths under request r passing edge (i, j) , and w_r is the weight of request r . We assume that the allocated capacity is proportional to the request weight, but is inversely proportional to the power (β) of the number of paths under the requests, i.e. $\beta > 0$; requests have fewer paths across the edge are compensated.

At the second stage, in a similar top-down manner, the allocated capacity on each request is further divided among different paths. The flow capacity for path l of request r on edge (i, j) is given by:

$$C_{i,j}^{r,l} = C_{i,j}^r \frac{d_{r,l}^\alpha}{\sum_l d_{r,l}^\alpha}. \quad (8)$$

Similarly, we assume that the division is proportional to the power (α) of the path length $d_{r,l}$. For $\alpha = 1/-1$, allocation is proportional/inverse proportional to the path length; for $\alpha = 0$, no dependence occurs. The polarity of α represents different strategies for the coordination of allocation on different paths: $\alpha > 0$ corresponds to the situation where longer paths get more flow quota as compensation, since long paths have low success rates; on the contrary, $\alpha < 0$ means shorter paths get more flow quota, as a strategy of prioritizing short paths as major paths for requests. α and β could be viewed as controlling the exploitation-exploration tradeoff of the algorithm, similar to the idea in the ant colonization algorithm. When $\alpha > 0$, the algorithm encourages the exploration of different paths by compensating longer paths, while $\alpha < 0$ indicates the exploitation on the usage of short paths. Similarly, when $\beta > 0$ the edge capacity is distributed on multiple requests in a fair manner, and $\beta < 0$ is the situation that prioritizes those requests with more paths passing through the edge. The optimal α and β is experimented in simulations and the corresponding strategies are discussed.

Finally, the allocated capacity for each path is lower-capped by the minimum flow f_{min} :

$$C_{i,j}^{r,l} = \max(C_{i,j}^{r,l}, f_{min}) \quad (9)$$

as a hard constraint. As expected, this treatment will likely raise the problem of capacity outbound, where:

$$\sum_{r,l} C_{i,j}^{r,l} > C_{i,j}. \quad (10)$$

If this happens, the overflow $\sum_{r,l} C_{i,j}^{r,l} - C_{i,j}$ is iteratively subtracted from paths receiving the largest share, until the outbound is resolved, which is guaranteed to converge under the definition of f_{min} . This forward local allocation process is summarized in Algorithm 1.

Algorithm 1 (*Proportional Share*) Capacity Allocation Scheduling

```
for  $(i, j)$  in  $E$  do
  Cut path list  $L = \{L_{i,j}^r\}$  s.t.  $|L| \leq l_{max}$  by path length. Keep short paths/sole paths for requests.
  Stage 1 Allocate  $C_{i,j}$  onto different requests  $R_{i,j} = \{r \text{ in } h \text{ for } h \in H\}$ .
  for  $r$  in  $R_{i,j}$  do
     $C_{i,j}^r = C_{i,j} \frac{w_r / (\#r_{i,j})^\beta}{\sum_r w_r / (\#r_{i,j})^\beta}$ 
  end for
  Stage 2 Allocate  $C_{i,j}^r$  onto different paths  $L_{i,j}^r = \{l \text{ in } h \text{ for } h \in H\}$  of request  $r$ .
  for  $l$  in  $L_{i,j}^r$  do
     $C_{i,j}^{r,l} = C_{i,j}^r \frac{d_{r,l}^\alpha}{\sum_l d_{r,l}^\alpha}$ 
     $C_{i,j}^{r,l} = \max(C_{i,j}^{r,l}, f_{min})$ 
  end for
  while  $\sum_{r,l} C_{i,j}^{r,l} > C_{i,j}$  do Subtract  $C_{i,j}^{r,l} - C_{i,j}$  from  $\max(C_{i,j}^{r,l})$ 
  end while
end for
```

B. Algorithm 2 *Progressive Filling*

The *progressive filling* method could guarantee max-min fairness. Unlike the other two methods, in the *progressive filling* algorithm, all paths are treated equally. For each path l of each request r , flow $f^{r,l}$ starts at zero. The unsaturated edge set E_{un} is the entire E , and the unsaturated path set L_{un} is all paths L^r for all $r \in R$.

The filling process assigns a uniform increase on each unsaturated path. Since in our setting edge capacities are integers, the increment is simply 1. Edges are gradually saturated along the filling process, and paths utilizing saturated edges stop flow increases. The integer scenario decides a slight different definition of edge saturation from the definition for non-integer flows, since it is often the case that an edge is left with a small number of unallocated channels, yet insufficient to distribute one more channel on all the unsaturated paths through it, i.e, $C_{i,j} - \sum_{r,l} f_{temp}^{i,j;r,l} < |H|_{i,j} - |L|_{un}^{i,j}$; in this case the edge is viewed as saturated (with a small leftover capacity).

Therefore, at each time step, an attempt of increment 1 is made on all $l \in L_{un}$. On each edge, we check the capacity constraint and decide if the attempt is successful, i.e., if the attempted aggregated flow is out of bound. When the attempt is successful, if the remaining capacity after deducting the attempted aggregated flow is less than the number of unsaturated paths passing through it, the edge is saturated and removed from E_{un} . Correspondingly, paths on this edge are removed from L_{un} . The filling process terminates when the new attempt is unsuccessful, i.e. it is not possible to assign one more channel uniformly to all unsaturated paths, without causing at least one remaining unsaturated edge to go off bound. Note that since flows increase in a fair manner, f_{min} is unnecessary for this method.

Algorithm 2 (*Progressive Filling*) Capacity Allocation Scheduling

```
1: Initialize flow matrix  $f^{r,l}$  for each path  $l$  of each request  $r$ .
2: Initialize unsaturated edge set  $E_{un} = E$ ; unsaturated path set  $L_{un} = \{L^r\}$  for  $r \in R$ .
3: while True do
4:    $end\ flag \leftarrow 0$ 
5:   Stage 1 Attempt an increment addition of flow uniformly on all unsaturated paths.
6:     for  $l \in L_{un}$  do  $f_{temp}^{r(l),l} = f^{r,l} + 1$ 
7:     end for
8:   Stage 2 Check the capacity constraint on each edge.
9:     for  $(i, j) \in E$  do
10:      if  $\sum_{r,l} f_{temp}^{i,j;r,l} > C_{i,j}$  then
11:         $end\ flag \leftarrow 1$ 
12:        break
13:      end if
14:      if  $C_{i,j} - \sum_{r,l} f_{temp}^{i,j;r,l} < |H|_{i,j} - |L|_{un}^{i,j}$  then
15:        Remove  $(i, j)$  from  $E_{un}$ .
16:      end if
17:    end for
18:    if  $end\ flag = 1$  then
19:      break
20:    end if
21:    Stage 3 Record succeeded attempt.
22:     $f^{r,l} \leftarrow f_{temp}^{r,l}$  for all  $r, l$ 
23:    Update  $L_{un}$  according to updated  $E_{un}$ .
24: end while
25: Obtain flow matrix  $f^{r,l}$ .
```

C. Algorithm 3 *Propagatory Update*

In the *proportional share* method, each edge decides its own capacity allocation and no communication between edges takes place. In the end, the final flow of the path is determined at *Step 4*, where the short-board constraint is applied on each edge. Intrinsically, this method will result in certain waste of edge capacity: capacity on edges will not be fully utilized, if some edges along the same path have narrower channels. Therefore the average capacity utilization of the network will be low.

This problem could be overcome if the short-board constraint is integrated into the capacity allocation process, instead of being carried out at the end. To achieve this, we store the global maximum flow information for each path, and constantly update this information (reducing the maximum flow of each path) throughout the allocation process. In fact, the allocation process on each edge could now be viewed as a backward-moving process: on a specific edge, assume that each requested path tries to claim its maximum flow, and then, we make deductions on those flows based on the current edge capacity. The gap between the sum of (desired) maximum flows over all paths passing the edge and the edge capacity is the amount to be deducted, and now we allocate the capacity deduction (instead of the capacity) to different requests, then to different paths in a similar two-stage manner as in the first method.

Algorithm 3 (*Propagatory Update*) Capacity Allocation Scheduling

```
1: Initialize max flow vector  $f_{max}^{r,l}$  for each path  $l$  of each request  $r$ .
2: Set  $count = 0$ .
3: while  $count < |E|$  do
4:     Select the next edge  $(i, j) \in E$  following the fixed edge order with startover.
5:     if  $R_{i,j} = \text{None}$  then
6:          $count++$ .
7:         Continue
8:     end if
9:     Cut path list  $L = \{L_{i,j}^r\}$  s.t.  $|L| \leq l_{max}$  by path length. Keep short paths/sole paths for
    requests.
10:    Calculate  $\Delta_{i,j} = \sum_{r,l} f_{max}^{r,l} - C_{i,j}$ ,  $f_{max}^r = \sum_l f_{max}^{r,l}$ ,  $f_{min}^r = \sum_l f_{min}^{r,l}$ 
11:    Stage 1 Allocate  $\Delta_{i,j}$  onto different requests  $R_{i,j} = \{r \text{ in } h \text{ for } h \in H\}$ .
12:    Calculate max deduction on requests  $\Delta_{i,j}^{r,max} = f_{max}^r - f_{min}^r$ 
13:    from smallest  $\Delta_{i,j}^{r,max}$  for  $r$  in  $R_{i,j}$  do
14:         $\Delta_{i,j}^r = \Delta_{i,j} \frac{\Delta_{i,j}^{r,max}/w_r}{\sum_r \Delta_{i,j}^{r,max}/w_r}$ 
15:        if  $\Delta_{i,j}^r > \Delta_{i,j}^{r,max}$  then  $\Delta_{i,j}^r = \Delta_{i,j}^{r,max}$ 
16:        end if
17:        Recalculate  $\Delta_{i,j} = \Delta_{i,j} - \Delta_{i,j}^r$ .
18:    Stage 2 Allocate  $\Delta_{i,j}^r$  onto different paths  $L_{i,j}^r = \{l \text{ in } h \text{ for } h \in H\}$  of request  $r$ .
19:    Calculate max deduction on paths  $\Delta_{i,j}^{r,l,max} = f_{max}^{r,l} - f_{min}^{r,l}$ 
20:    from smallest  $\Delta_{i,j}^{r,l,max}$  for  $l$  in  $L_{i,j}^r$  do
21:         $\Delta_{i,j}^{r,l} = \Delta_{i,j}^r \frac{\Delta_{i,j}^{r,l,max}(d_{r,l})^{-\alpha}}{\sum_l \Delta_{i,j}^{r,l,max}(d_{r,l})^{-\alpha}}$ 
22:        if  $\Delta_{i,j}^{r,l} > \Delta_{i,j}^{r,l,max}$  then  $\Delta_{i,j}^{r,l} = \Delta_{i,j}^{r,l,max}$ 
23:        end if
24:        Recalculate  $\Delta_{i,j}^r = \Delta_{i,j}^r - \Delta_{i,j}^{r,l}$ .
25:    Stage 3 Apply deduction and determine flow  $f_{i,j}^{r,l}$ .
26:     $f_{i,j}^{r,l} = f_{max}^{r,l} - \Delta_{i,j}^{r,l}$ 
27:    Stage 4 Update max flow  $f_{max}^{r,l}$  of the path.
28:
29:    if  $f_{max}^{r,l}$  updated then
30:        Reset  $count = 0$ .
31:    else
32:         $count++$ .
33:    end if
34: end while
35: Determine converged  $f_{i,j}^{r,l}$ .
```

The total amount of capacity $\Delta_{i,j}$ to be deducted is given by:

$$\Delta_{i,j} = \sum_{r,l} f_{max}^{r,l} - C_{i,j} \quad (11)$$

where $f_{max}^{r,l}$ is the desired max flow of path l for request r , whose value is set as the flow of the previous edge in the path: $f_{max}^{r,l} = f_{i,j}^{r,l}{}_{prev}$. On each path, the maximum amount of deduction (its maximum possible contribution to the total deduction) $\Delta_{i,j}^{r,l,max}$ is the difference between its maximum desired flow and its minimum flow:

$$\Delta_{i,j}^{r,l,max} = f_{max}^{r,l} - f_{min}. \quad (12)$$

Correspondingly, the maximum amount of capacity deduction a request could contribute is the sum from all its paths:

$$\Delta_{i,j}^{r,max} = f_{max}^r - f_{min} = \sum_l \Delta_{i,j}^{r,l,max}. \quad (13)$$

At the first stage, the total amount of deduction $\Delta_{i,j}$ is allocated to different requests as:

$$\Delta_{i,j}^r = \min(\Delta_{i,j} \frac{\Delta_{i,j}^{r,max}/w_r}{\sum_r \Delta_{i,j}^{r,max}/w_r}, \Delta_{i,j}^{r,max}) \quad (14)$$

where requests with large weights w_r gets small amount of deduction and the realized amount could not exceed the maximum amount $\Delta_{i,j}^{r,max}$. This allocation procedure begins from requests having the smallest $\Delta_{i,j}^{r,max}$ and proceeds until all $\Delta_{i,j}$ has been allocated, which is guaranteed since:

$$\sum_{r,l} \Delta_{i,j}^{r,l,max} = \sum_{r,l} f_{max}^{r,l} - \sum_{r,l} f_{min} \geq \sum_{r,l} f_{max}^{r,l} - C_{i,j} = \Delta_{i,j} \quad (15)$$

i.e. the total amount of desired deduction is no greater than the total amount of maximum deduction.

At the second stage, the amount of deduction on each request is allocated to different paths according to:

$$\Delta_{i,j}^{r,l} = \min(\Delta_{i,j}^r \frac{\Delta_{i,j}^{r,l,max}/(d_{r,l})^\alpha}{\sum_l \Delta_{i,j}^{r,l,max}/(d_{r,l})^\alpha}, \Delta_{i,j}^{r,l,max}). \quad (16)$$

Again, the maximum constraint $\Delta_{i,j}^{r,l,max}$ is applied. Similar to the first method, the weight of each path is assumed to be proportional to the power α of path length $d_{r,l}$. To make sure in this backward deduction process the polarity of α corresponds to the same strategies as in the forward allocation process, we put $(d_{r,l})^\alpha$ in the denominator.

After the two stages, the amount of capacity deduction on each path l for request r $\Delta_{i,j}^{r,l}$ is determined, so the flow capacity $f_{i,j}^{r,l}$ is given by:

$$f_{i,j}^{r,l} = f_{max}^{r,l} - \Delta_{i,j}^{r,l}. \quad (17)$$

$f_{i,j}^{r,l}$ is then the updated maximum flow on edges in the path, if it's smaller than the previous value. Note that not only the maximum flow values on following edges, but also on previous edges are updated. Essentially, the short-board constraint also narrows the flow on previous edges along the path, whose capacity allocation is then recalculated; since the flow on the current path is narrowed, recalculation may release extra capacity for other paths and hence the overall capacity allocation becomes more efficient.

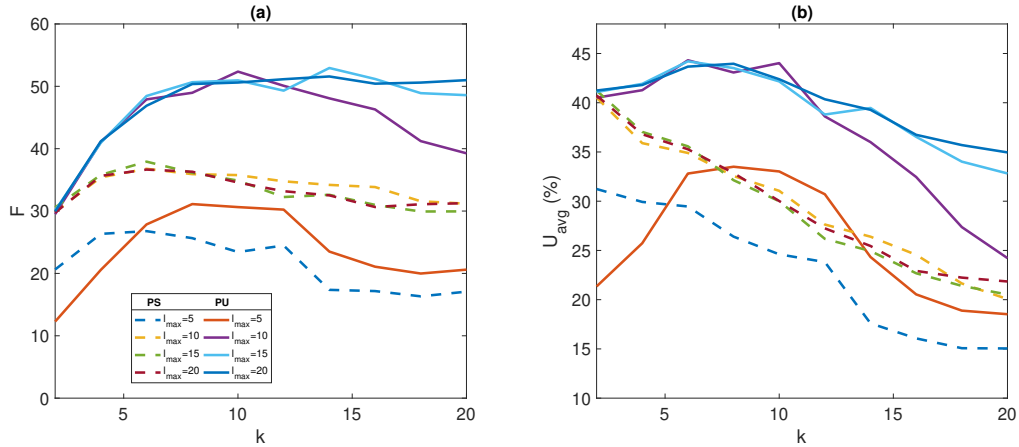
Given that the path information is required in the algorithm, this method does not follow a predefined

order in treating the edges, as opposed to the first method where all edges could be treated individually. Since lattice networks are regular and easy to be ordered, we used a fixed order in treating the edges and iterate with startovers, until no update of flow occurs on any edge, i.e. the headcount of no improvements equals the edge number. The iteration is guaranteed to converge if the (directed) paths are all non-cyclic; in practice we ensure that the two nodes in a request pair are not identical, thus the condition holds. The backward adaptive deduction process is summarized in Algorithm 3.

Supplementary Notes 3: Discussion on scheme parameters

In this section, we discuss the parameters that are important in determining the performance of capacity allocation algorithms. The parameters are $X = \{l_{max}, k, \alpha, \beta\}$ for *proportional share* and *propagatory update*, and $X = k$ for *progressive filling*. We will focus on $\{l_{max}, \alpha, \beta\}$ here.

A. l_{max} dependence



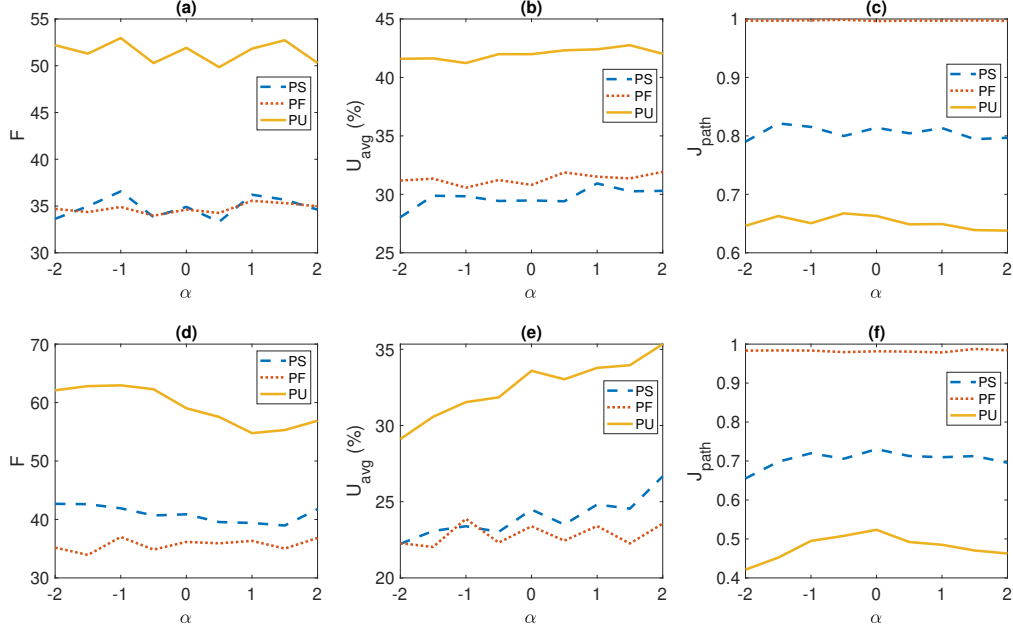
Supplementary Figure 3: Routing performance for different l_{max} and k . (a). Total flow F . (b). Average capacity utilization U_{avg} . The request distance is 3 and the other parameters are $\{\alpha, \beta\} = \{1, 0\}$ here. The PF algorithm performance is not shown here since it has no dependence on l_{max} .

The maximal allowed number of paths on each edge l_{max} characterizes the controllability of edge traffic. In Fig. 3 (a-b) we show the k dependence of the total flow and average capacity utilization with different l_{max} . For fixed k , the total flow F and average capacity utilization U_{avg} show an increase as a function of l_{max} at beginning. This performance gain can be attributed to the fact that each edge enables more paths passing through it. This increase then saturates at larger l_{max} (for example, $l_{max} = 10$ here), in which scenario the edge removal balances the gain.

B. α dependence

The parameter α determines the relation between capacity allocation and path length for the PS and PU algorithms, as we discussed in Section . Here we show the dependence of routing performance

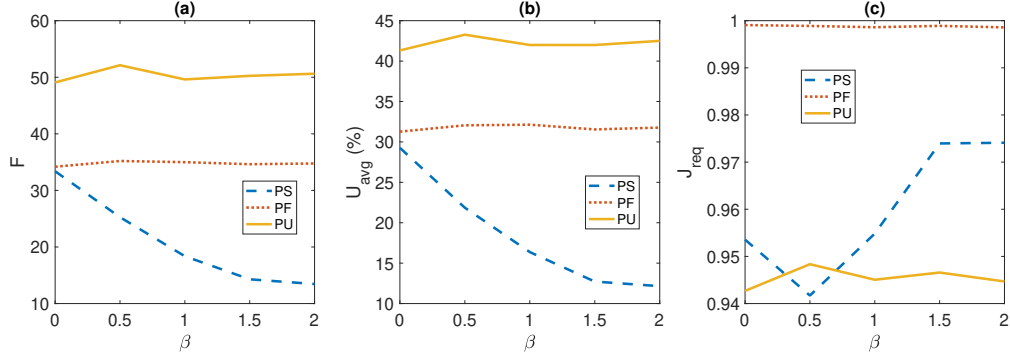
on α . In the first scenario (in Fig. 4 (a-c)), when the request distance is large and k is small, the utilized paths have identical length (shortest length) and the performance has no dependence on α . While in the second case when the request distance is large and the paths spread out the network (in Fig. 4 (d-f)), the allocation process will prefer long paths at large α and the capacity of edges outside the shortest paths can be better utilized, leading to an increase of the U_{avg} . The path fairness J_{path} seems to take the optimal value at $\alpha = 0$, in which case the capacity allocation will neither prefer short paths nor long paths.



Supplementary Figure 4: Routing performance with respect to α . **(a-c)**. Total flow F , average capacity utilization U_{avg} and path fairness J_{path} for request distance 2 and $k = 10$. **(d-f)**. The metrics for request distance 3 and $k = 20$. $\{l_{max}, \beta\} = \{15, 0\}$ The PF algorithm performance is shown here for comparison.

C. β dependence

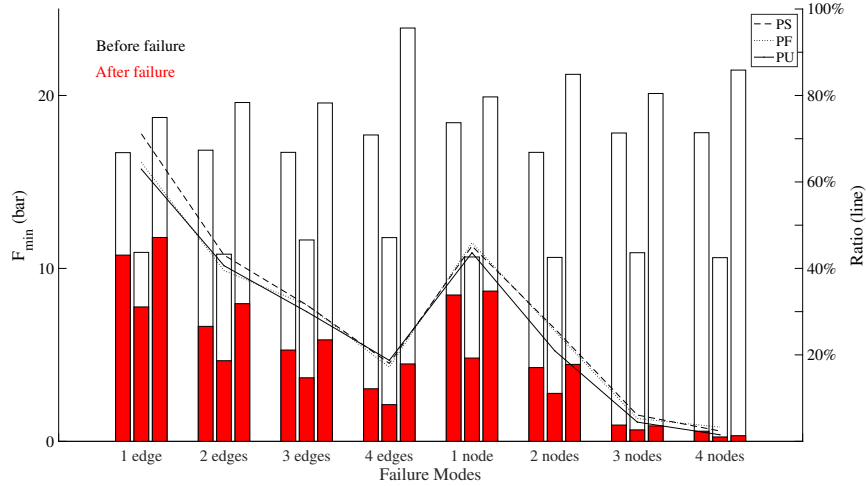
The parameter β determines the relation between capacity allocation and number of paths under different request for the PS algorithm, as we have discussed in Section . The throughput and traffic metrics (Fig. 5 (a-b)) indicates that allocation on an edge without considering the number of paths for different requests that going through that edge (*i.e.*, $\beta = 0$) is the best choice. However, the request fairness J_{req} will be degraded in this case (Fig. 5 (c)).



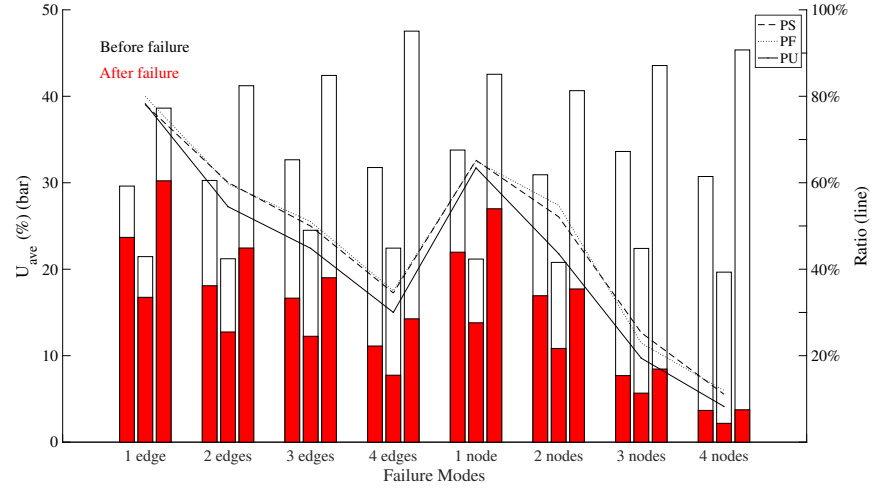
Supplementary Figure 5: Routing performance with respect to β . (a). Total flow F . (b). Average capacity utilization U_{avg} . (c). Request fairness J_{req} . The request distance is 3 and the other parameters are $\{l_{max}, k, \alpha\} = \{15, 10, 1\}$ here.

Supplementary Notes 4: Performance under network failures

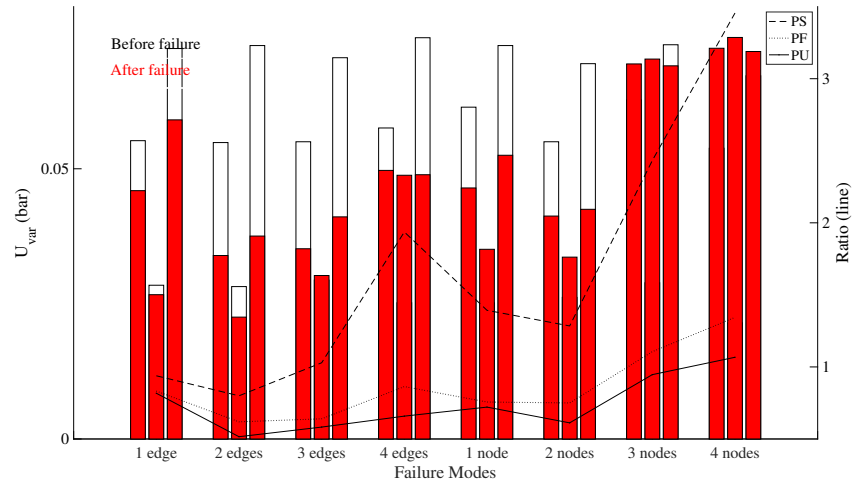
In the main text, we have shown the total flow in the presence of network failures. Besides system throughput, the changes of other performance measures before and after network failures are shown below for the convenience of readers. Note that after network failures, some paths are not accessible, so some values of γ are missing. In each figure, white/red bars indicate values before/after failures, respectively.



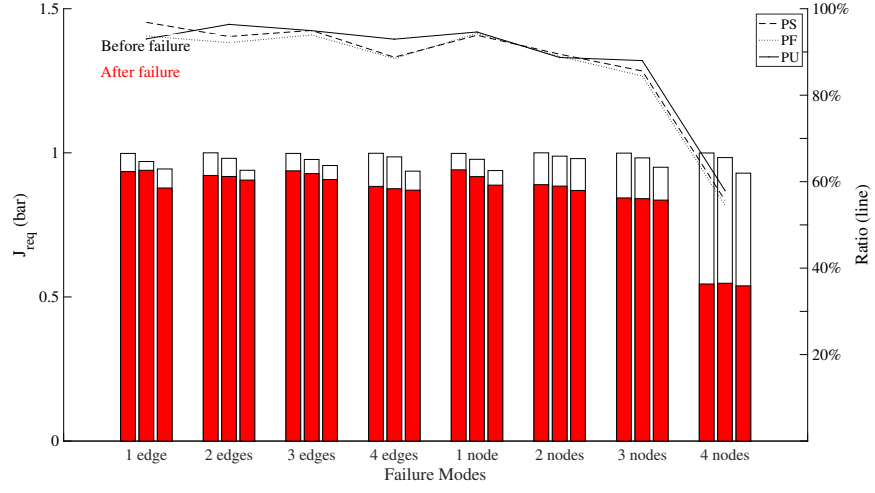
Supplementary Figure 6: Robustness in the presence of network failures: minimum flow over requests F_{min}



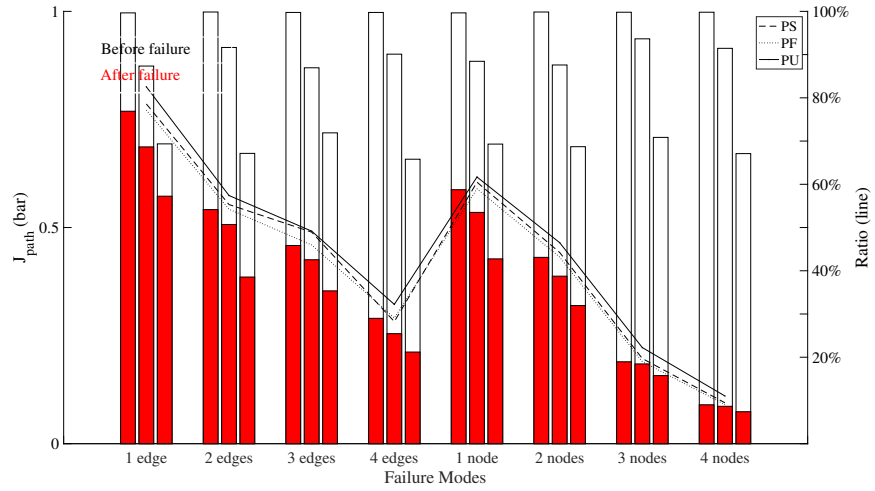
Supplementary Figure 7: Robustness in the presence of network failures: average capacity utilization U_{ave}



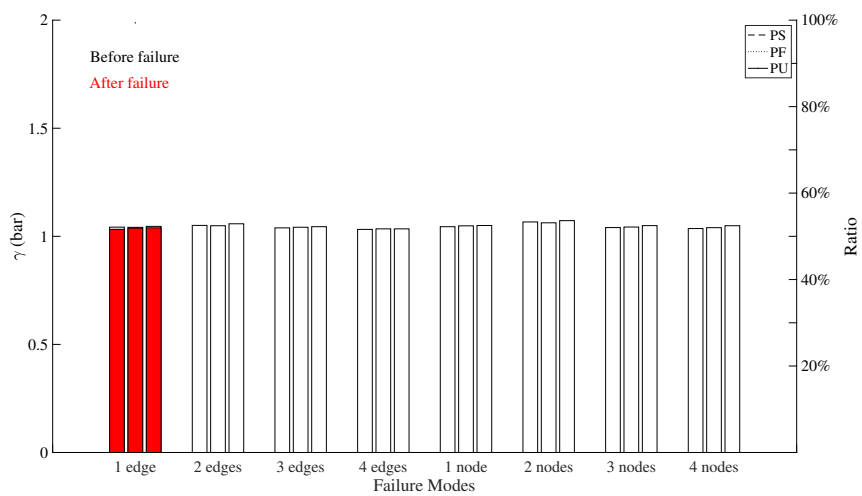
Supplementary Figure 8: Robustness in the presence of network failures: variance of capacity utilization U_{var}



Supplementary Figure 9: Robustness in the presence of network failures: fairness over requests J_{req}



Supplementary Figure 10: Robustness in the presence of network failures: fairness over paths J_{path}



Supplementary Figure 11: Robustness in the presence of network failures: γ